

24th South Symposium on Microelectronics

May 4th to 9th, 2009
Pelotas – RS – Brazil

Proceedings

Edited by

Lisane Brisolara de Brisolara
Luciano Volcan Agostini
Reginaldo Nóbrega Tavares

Promoted by

Brazilian Computer Society (SBC)
Brazilian Microelectronics Society (SBMicro)
IEEE Circuits and Systems Society (IEEE CAS)

Organized by

Universidade Federal de Pelotas (UFPel)
Universidade Federal do Pampa (UNIPAMPA)
Instituto Federal Sul-rio-grandense – Pelotas (IF)

Published by

Brazilian Computer Society (SBC)

Universidade Federal de Pelotas
Instituto de Física e Matemática
Departamento de Informática
Campus Universitário, s/n
Pelotas – RS – Brazil

Dados de Catalogação na Publicação (CIP) Internacional
Ubirajara Buddin Cruz – CRB 10/901
Biblioteca Setorial de Ciência & Tecnologia - UFPel

S726p South Symposium on Microelectronics (24. : 2009 : Pelotas, RS, Brazil).

Proceedings / 24. South Symposium on Microelectronics; edited by
Lisane Brisolara de Brisolara, Luciano Volcan Agostini, Reginaldo
Nóbrega Tavares. – Pelotas : UFPel, 2009. - 267p. : il. - Conhecido
também como SIM 2009.

ISBN 857669234-1

1. Microelectronics. 2. Digital Design. 3. Analog Design. 4. CAD Tools.
I. Brisolara, Lisane. II. Agostini, Luciano. III. Tavares, Reginaldo. IV.
Título.

CDD: 621.3817

Printed in Pelotas, Brazil.

Cover and Commemorative Stamp:	Leomar Soares da Rosa Junior (UFPel)
Cover Picture:	Luciano Volcan Agostini (UFPel)
Edition Production:	Felipe Martin Sampaio (UFPel)
	Daniel Munari Palomino (UFPel)
	Robson Sejanos Soares Dornelles (UFPel)

Foreword



Welcome to the 24th edition of the South Symposium on Microelectronics. This symposium, originally called Microelectronics Internal Seminar (SIM), started in 1984 as an internal workshop of the Microelectronics Group (GME) at the Federal University of Rio Grande do Sul (UFRGS) in Porto Alegre. From the beginning, the main purpose of this seminar was to offer the students an opportunity for practicing scientific papers writing, presentation and discussion, as well as to keep a record of research works under development locally.

The event was renamed as South Symposium on Microelectronics in 2002 and transformed into a regional event, reflecting the growth and spreading of teaching and research activities on microelectronics in the region. The proceedings, which started at the fourth edition, have also improved over the years, receiving ISBN numbers, adopting English as the mandatory language, and incorporating a reviewing process that also involves students. The papers submitted to this symposium represent different levels of research activity, ranging from early undergraduate research assistant assignments to advanced PhD works in cooperation with companies and research labs abroad.

This year SIM takes place at Pelotas together with the 11th edition of the regional Microelectronics School (EMICRO). A series of basic, advanced, hands-on and demonstrative short courses were provided by invited speakers. It is going to be a very special edition because SIM will celebrate 25 years and EMICRO 10 years, exactly in the year that will be commemorated the 40th year of Federal University of Pelotas (UFPel) and the 15th year of Computer Science Course at UFPel.

These proceedings include 59 papers organized in 6 topics: Design Automation Tools, Digital Design, IC Process and Physical Design, SoC/Embedded Design and Networks-on-chip, Test and Fault Tolerance, and Video Processing. These papers came from 15 different institutions: UFRGS, UFPel, FURG, UFSM, IF Sul-rio-grandense, UFSC, UNIPAMPA Bagé and Alegrete, PUC-RS, UCPel, UNIVALI, ULBRA, UNIUI, UFRN and IT-Coimbra.

We would finally like to thank all individuals and organizations that helped to make this event possible. SIM 2009 was co-organized among UFPel, UNIPAMPA Bagé and IF-Sul-rio-grandense Pelotas, promoted by the Brazilian Computer Society (SBC), the Brazilian Microelectronics Society (SBMicro) and IEEE CAS Region 9, receiving financial support from CNPq and CAPES Brazilian agencies, DLP-CAS program and Open CAD Consulting Group. Special thanks go to the authors and reviewers that spent precious time on the preparation of their works and helped to improve the quality of the event.

Pelotas, May 4, 2009

Lisane Brisolara de Brisolara
Luciano Volcan Agostini
Reginaldo Nóbrega Tavares

Fonte das Nereidas

SIM 2009 Proceedings Cover Picture



SIM 2009 cover picture presents a partial view of the **Fonte das Nereidas**, the first public water fountain of Pelotas. This fountain is installed in the city downtown, in the Coronel Pedro Osório square and it is a part of the great historic collection presented in our city. Among historic buildings, monuments and waterworks, we choose the **Fonte das Nereidas** to illustrate the cover of our event. This historical patrimony, besides the beauty, is one of the most known symbols of Pelotas city.

The text bellow presents the history of the **Fonte das Nereidas** and it was extracted from the website of the project *Sanitation Museum and Cultural Space*. The original content was freely translated to English, since the original text is written in Portuguese. The complete content in Portuguese is available at (www.pelotas.com.br/sanep/museu/chafarizes.html).

“In 1871, the Government of the Província de São Pedro (currently Rio Grande do Sul state) assigned a contract with Mr. Hygino Corrêa Durão to implement the Companhia Hydráulica Pelotense in Pelotas city. This contract obliged the installation of four water fountains with four taps, with chandeliers to allow the diurnal and nocturnal water sell service. The fountains should be constructed in iron and they should have the same quality of the capital fountains.

The Companhia Hydráulica Pelotense report from 1872 informed: “The fountains models from the Durenne foundry in Paris were just delivered...”

In April 5th, 1874, three of four fountains were opened to the public, being constantly monitored by a fountain guard. Next to the fountains there were chandeliers to illuminate the local at night. A barrel with twenty five liters of water was sold by twenty Réis (the Brazilian current currency) during those days.

Based on existent documentation, all fountains in Pelotas were ordered from the Durenne iron foundry, located at Champagne Ardenne region, in France.

The fountain situated at Pedro II square (currently known as Coronel Pedro Osório square) was the first fountain established in the city. According to the records available in the city council, this fountain received permission to be installed on June 25th, 1873. During 1915, the base construction was completed.

*This fountain is very important because its model achieved a great success during the Universal Exhibition of London in 1862. It was carved by Jean Baptiste Jules Klagmann and Ambroise Choiselat. There is a replica, in a larger size, at Edinburgh city in Scotland. In Pelotas this fountain is known as **Fonte das Nereidas**.”*

SIM 2009 - 24th South Symposium on Microelectronics

Pelotas – RS – Brazil
May 4th to 9th, 2009

General Chair

Prof. Luciano Volcan Agostini (UFPel)

SIM Program Chairs

Profa. Lisane Brisolara de Brisolara (UFPel)

Prof. Reginaldo Nóbrega Tavares (UNIPAMPA-Bagé)

EMICRO Program Chairs

Prof. José Luís Almada Güntzel (UFSC)

Prof. Júlio Carlos Balzano de Mattos (UFPel)

Financial Chair

Prof. Leomar Soares da Rosa Junior (UFPel)

Local Arrangements Chair

Profa. Eliane Alcoforado Diniz (UFPel)

IEEE CAS Liaison

Prof. Ricardo Augusto da Luz Reis (UFRGS)

Program Committee

Prof. Alessandro Gonçalves Girardi (UNIPAMPA-Alegrete)

Prof. José Luís Almada Güntzel (UFSC)

Prof. Júlio Carlos Balzano de Mattos (UFPel)

Prof. Leomar Soares da Rosa Junior (UFPel)

Profa. Lisane Brisolara de Brisolara (UFPel)

Prof. Luciano Volcan Agostini (UFPel)

Prof. Marcelo Johann (UFRGS)

Prof. Reginaldo Nóbrega Tavares (UNIPAMPA-Bagé)

Prof. Ricardo Augusto da Luz Reis (UFRGS)

Local Arrangements Committee

Prof. Bruno Silveira Neves (UNIPAMPA-Bagé)

Prof. Marcello da Rocha Macarthy (UFPel)

Prof. Sandro Vilela da Silva (IF Sul-rio-grandense)

Roger Endrigo Carvalho Porto (UFRGS)

Carolina Marques Fonseca (UFPel)

Daniel Munari Palomino (UFPel)

Felipe Martin Sampaio (UFPel)

Gabriel Sica Siedler (UFPel)

Mateus Grellert da Silva (UFPel)

Robson Sejanos Soares Dornelles (UFPel)

List of Reviewers

Adriel Ziesemer Junior, Msc., UFRGS,
Alessandro Girardi, Dr., UNIPAMPA
Alexandre Amory, Dr., CEITEC
André Borin, Dr., UFRGS
Andre Aita, Dr. UFSM
Antonio Carlos Beck F., Dr., UFRGS
Bruno Neves, Msc., UNIPAMPA
Bruno Zatt, Msc UFRGS
Caio Alegretti, Msc., UFRGS
Carol Concatto, UFRGS
Cesar Prior, Msc., UNIPAMPA
Cesar Zeferino, Dr., UNIVALI
Cláudio Diniz, UFRGS
Cristiano Lazzari, Dr., INESC-ID
Cristina Meinhardt, Msc., UFRGS
Dalton Colombo, Msc., CI-Brasil
Daniel Ferrão, Msc., CEITEC
Debora Matos, UFRGS
Denis Franco, Dr., FURG
Digeorgia da Silva, Msc. UFRGS
Edson Moreno, Msc., PUCRS
Eduardo Costa, Dr., UCPel
Eduardo Flores, Msc., Nangate do Brasil
Eduardo Rhod, Msc., UFRGS
Elias Silva Júnior, Dr., CEFET-CE
Felipe Marques, Dr., UFRGS
Felipe Pinto, UFRGS
Gilberto Marchioro, Dr., ULBRA
Gilson Wirth, Dr., UFRGS
Giovani Pesenti, Dr., UFRGS
Guilherme Flach, UFRGS
Gustavo Girão, Msc., UFRGS
Gustavo Neuberger, Dr., UFRGS
Gustavo Wilke, Dr., UFRGS
Helen Franck, UFRGS
Júlio C. B. Mattos, Dr., UFPel
José Rodrigo Azambuja, UFRGS
Jose Luis Guntzel, Dr., UFSC
Leandro Rosa, UFRGS
Leomar Soares da Rosa Junior, Dr., UFPel
Leonardo Kunz, UFRGS
Lisane Brisolara, Dr., UFPel
Lucas Brusamarello, Msc., UFRGS
Luciano Volcan Agostini, Dr., UFPel
Luciano Ost, Msc., PUCRS
Luis Cleber Marques, Dr., IF Sul-rio-
grandense
Luis Fernando Ferreira, Msc. UFRGS
Marcello da Rocha Macarthy, Msc. UFPEL
Marcelo Porto, Msc., UFRGS
Marcio Oyamada, Dr., UNIOESTE
Marcio Eduardo Kreutz, Dr., UFRN
Marco Wehrmeister, Msc., UFRGS
Marcos Hervé, UFRGS
Margrit Krug, Dr., UFRGS
Mateus Rutzig, Msc., UFRGS
Mauricio Lima Pilla, Dr., UFPel
Monica Pereira, Msc., UFRGS
Osvaldo Martinello, UFRGS
Paulo Butzen, Msc., UFRGS
Reginaldo Tavares, Dr., UNIPAMPA
Renato Hentschke, Dr., Intel Corporation
Roger Porto, Msc., UFPel
Ronaldo Husemann, Msc., UFRGS
Sandro Sawicki, Msc., UFRGS
Sandro Silva, Msc., IF Sul-rio-grandense
Sidinei Ghissoni, Msc., UNIPAMPA
Tatiana Santos, Dr., CEITEC
Thaísa Silva, Msc., UFRGS
Thiago Assis, UFRGS
Tomás Moreira, UFRGS
Ulisses Corrêa, UFRGS
Vagner Rosa, Msc., UFRGS
Vinicius Dal Bem, UFRGS, Brazil

Table of Contents

Section 1 : DESIGN AUTOMATION TOOLS	13
An Iterative Partitioning Refinement Algorithm Based on Simulated Annealing for 3D VLSI Circuits	
Sandro Sawicki, Gustavo Wilke, Marcelo Johann, Ricardo Reis	15
3D Symbolic Routing Viewer	
Érico de Moraes Nunes and Reginaldo da Nóbrega Tavares	19
Improved Detailed Routing using Pathfinder and A*	
Charles Capella Leonhardt, Adriel Mota Ziesemer Junior, Ricardo Augusto da Luz Reis	23
A New Algorithm for Fast and Efficient Boolean Factoring	
Vinicius Callegaro, Leomar S. da Rosa Jr, André I. Reis, Renato P. Ribas	27
PlaceDL: A Global Quadratic Placement Using a New Technique for Cell Spreading	
Carolina Lima, Guilherme Flach, Felipe Pinto, Ricardo Reis	31
EL-FI – On the Elmore “Fidelity” under Nanoscale Technologies	
Tiago Reimann, Glauco Santos, Ricardo Reis.....	37
Automatic Synthesis of Analog Integrated Circuits Using Genetic Algorithms and Electrical Simulations	
Lucas Compassi Severo, Alessandro Girardi	41
PicoBlaze C: a Compiler for PicoBlaze Microcontroller Core	
Caroline Farias Salvador, André Raabe, Cesar Albenes Zeferino	45
Section 2 : DIGITAL DESIGN	49
Equivalent Circuit for NBTI Evaluation in CMOS Logic Gates	
Nivea Schuch, Vinicius Dal Bem, André Reis, Renato Ribas	51
Evaluation of XOR Circuits in 90nm CMOS Technology	
Carlos E. de Campos, Jucemar Monteiro, Robson Ribas, José Luís Güntzel.....	55
A Comparison of Carry Lookahead Adders Mapped with Static CMOS Gates	
Jucemar Monteiro, Carlos E. de Campos, Robson Ribas, José Luís Güntzel.....	59
Optimization of Adder Compressors Using Parallel-Prefix Adders	
João S. Altermann, André M. C. da Silva, Sérgio A. Melo, Eduardo A. C. da Costa	63
Techniques for Optimization of Dedicated FIR Filter Architectures	
Mônica L. Matzenauer, Jônatas M. Roschild, Leandro Z. Pieper, Diego P. Jaccottet, Eduardo A. C. da Costa, Sergio J. M. de Almeida.	67
Reconfigurable Digital Filter Design	
Anderson da Silva Zilke, Júlio C. B. Mattos	71
ATmega64 IP Core with DSP features	
Eliano Rodrigo de O. Almança, Júlio C. B. Mattos	75

Using an ASIC Flow for the Integration of a Wallace Tree Multiplier in the DataPath of a RISC Processor

Helen Franck, Daniel S. Guimarães Jr, José L. Güntzel, Sergio Bampi,
Ricardo Augusto da Luz Reis, ... 79

Implementation flow of a Full Duplex Internet Protocol Hardware Core according to the Brazil-IP Program methodology

Cristian Müller, Lucas Teixeira, Paulo César Aguirre, Leandro Zafalon Pieper,
Josué Paulo de Freitas, Gustavo F. Dessbesel, João Baptista Martins 83

Wavelets to improve DPA: a new threat to hardware security?

Douglas C. Foster, Daniel G. Mesquita, Leonardo G. L. Martins, Alice Kozakevicius 87

Design of Electronic Interfaces to Control Robotic Systems Using FPGA

André Luís R. Rosa, Gabriel Tadeo, Vitor I. Gervini, Sebastião C. P. Gomes, Vagner S. da Rosa 91

Design Experience using Altera Cyclone II DE2-70 Board

Mateus Grellert da Silva, Gabriel S. Siedler, Luciano V. Agostini, Júlio C. B. Mattos 95

Section 3 : IC PROCESS AND PHYSICAL DESIGN..... 99

Passivation effect on the photoluminescence from Si nanocrystals produced by hot implantation

V. N. Obadowski, U. S. Sias, Y.P. Dias and E. C. Moreira 101

The Ionizing Dose Effect in a Two-stage CMOS Operational Amplifier

Ulisses Lyra dos Santos, Luís Cléber C. Marques, Gilson I. Wirth 105

Verification of Total Ionizing Dose Effects in a 6-Transistor SRAM Cell by Circuit Simulation in the 100nm Technology Node

Vitor Paniz , Luís Cléber C. Marques, Gilson I. Wirth 109

Delay Variability and its Relation to the Topology of Digital Gates

Digeorgia N. da Silva, André I. Reis, Renato P. Ribas..... 113

Design and Analysis of an Analog Ring Oscillator in CMOS 0.18µm Technology

Felipe Correa Werle, Giovano da Rosa Camaratta, Eduardo Conrad Junior,
Luis Fernando Ferreira, Sergio Bampi 119

Loading Effect Analysis Considering Routing Resistance

Paulo F. Butzen, André I. Reis, Renato P. Ribas..... 123

Section 4: SOC/EMBEDDED DESIGN AND NETWORK-ON-CHIP 127

A Floating Point Unit Architecture for Low Power Embedded Systems Applications

Raphael Neves, Jeferson Marques, Sidinei Ghissoni, Alessandro Girardi 129

MIPS VLIW Architecture and Organization

Fábio Luís Livi Ramos, Osvaldo Martinello Jr, Luigi Carro 133

SpecISA: Data and Control Flow Optimized Processor

Daniel Guimarães Jr., Tomás Garcia Moreira, Ricardo Reis, Carlos Eduardo Pereira 137

High Performance FIR Filter Using ARM Core

Debora Matos, Leandro Zanetti, Sergio Bampi, Altamiro Susin 141

Modeling Embedded SRAM with SystemC Gustavo Henrique Nihei, José Luís Güntzel.....	145
Implementing High Speed DDR SDRAM memory controller for a XUPV2P and SMT395 Sundance Development Board Alexsandro C. Bonatto, Andre B. Soares, Altamiro A. Susin	149
Data storage using Compact Flash card and Hardware/Software Interface for FPGA Embedded Systems Leonardo B. Soares, Vitor I. Gervini, Sebastião C. P. Gomes, Vagner S. Rosa	153
Design of Hardware/Software board for the Real Time Control of Robotic Systems Dino P. Cassel, Mariane M. Medeiros, Vitor I. Gervini, Sebastião C. P. Gomes, Vagner S. da Rosa	157
HeMPS Station: an environment to evaluate distributed applications in NoC-based MPSoCs Cezar R. W. Reinbrecht, Gerson Scartezini, Thiago R. da Rosa, Fernando G. Moraes	161
Analysis of the Cost of Implementation Techniques for QoS on a Network-on-Chip Marcelo Daniel Berejuck, Cesar Albenes Zeferino	165
Performance Evaluation of a Network-on-Chip by using a SystemC-based Simulator Magnos Roberto Pizzoni, Cesar Albenes Zeferino.....	169
Traffic Generator Core for Network-on-Chip Performance Evaluation Miklécio Costa, Ivan Silva	173
Section 5 : TEST AND FAULT TOLERANCE.....	177
A Design for Test Methodology for Embedded Software Guilherme J. A. Fachini, Humberto V. Gomes, Érika Cota, Luigi Carro	179
Testing Requirements of an Embedded Operating System: The Exception Handling Case Study Thiago Dai Pra, Luciéli Tolfo Beque, Érika Cota	183
Testing Quaternary Look-up Tables Felipe Pinto, Érika Cota, Luigi Carro, Ricardo Reis.....	187
Model for S.E.T propagation in CMOS Quaternary Logic Valter Ferreira, Gilson Wirth, Altamiro Susin	193
Fault Tolerant NoCs Interconnections Using Encoding and Retransmission Matheus P. Braga, Érika Cota, Marcelo Lubaszewski.....	197
Section 6 : VIDEO PROCESSING.....	201
A Quality and Complexity Evaluation of the Motion Estimation with Quarter Pixel Accuracy Leandro Rosa, Sergio Bampi, Luciano Agostini	203
Scalable Motion Vector Predictor for H.264/SVC Video Coding Standard Targeting HDTV Thaísa Silva, Luciano Agostini, Altamiro Susin, Sergio Bampi.....	207
Decoding Significance Map Optimized by the Speculative Processing of CABAD Dieison Antonello Deprá, Sergio Bampi.....	211

H.264/AVC Variable Block Size Motion Estimation for Real-Time 1080HD Video Encoding Roger Porto, Luciano Agostini, Sergio Bampi	215
An Architecture for a T Module of the H.264/AVC Focusing in the Intra Prediction Restrictions Daniel Palomino, Felipe Sampaio, Robson Dornelles, Luciano Agostini.....	219
Dedicated Architecture for the T/Q/IQ/IT Loop Focusing the H.264/AVC Intra Prediction Felipe Sampaio, Daniel Palomino, Robson Dornelles, Luciano Agostini	223
A Real Time H.264/AVC Main Profile Intra Frame Prediction Hardware Architecture for High Definition Video Coding Cláudio Machado Diniz, Bruno Zatt, Luciano Volcan Agostini, Sergio Bampi, Altamiro Amadeu Susin.....	227
A Novel Filtering Order for the H.264 Deblocking Filter and Its Hardware Design Targeting the SVC Interlayer Prediction Guilherme Corrêa, Thaísa Leal, Luís A. Cruz, Luciano Agostini.....	231
High Performance and Low Cost CAVLD Architecture for H.264/AVC Decoder Thaísa Silva, Fabio Pereira, Luciano Agostini, Altamiro Susin, Sergio Bampi.....	235
A Multiplierless Forward Quantization Module Focusing the Intra Prediction Module of the H.264/AVC Standard Robson Dornelles, Felipe Sampaio, Daniel Palomino, Luciano Agostini	239
H.264/AVC Video Decoder Prototyping in FPGA with Embedded Processor for the SBTVD Digital Television System Márlon A. Lorencetti, Fabio I. Pereira, Altamiro A. Susin.....	243
Design, Synthesis and Validation of an Upsampling Architecture for the H.264 Scalable Extension Thaísa Leal da Silva, Fabiane Rediess, Guilherme Corrêa, Luciano Agostini, Altamiro Susin, Sergio Bampi	247
SystemC Modeling of an H.264/AVC Intra Frame Video Encoder Bruno Zatt, Cláudio Machado Diniz, Luciano Volcan Agostini, Altamiro Susin, Sergio Bampi.....	251
A High Performance H.264 Deblocking Filter Hardware Architecture Vagner S. Rosa, Altamiro Susin, Sergio Bampi	255
A Hardware Architecture for the New SDS-DIC Motion Estimation Algorithm Marcelo Porto, Luciano Agostini, Sergio Bampi.....	259
SDS-DIC Architecture with Multiple Reference Frames for HDTV Motion Estimation Leandro Rosa, Débora Matos, Marcelo Porto, Altamiro Susin, Sergio Bampi, Luciano Agostini.....	263
Author Index	267



Design Automation Tools

An Iterative Partitioning Refinement Algorithm Based on Simulated Annealing for 3D VLSI Circuits

^{1,2}Sandro Sawicki, ¹Gustavo Wilke, ¹Marcelo Johann, ¹Ricardo Reis
{sawicki, wilke, johann, reis}@inf.ufrgs.br

¹UFRGS – Universidade Federal do Rio Grande do Sul
Instituto de Informática

²UNIJUI – Universidade Regional do Noroeste do Estado do Rio Grande do Sul
Departamento de Tecnologia

Abstract

Partitioning algorithms are responsible for dividing random logic cells and ip blocks into the different tiers of a 3D design. Cells partitioning also helps to reduce the complexity of the next steps of the physical synthesis (placement and routing). In spite of the importance of the cell partitioning for the automatic synthesis of 3D designs it been performed in the same way as in 2D designs. Graph partitioning algorithms are used to divide the cells into the different tiers without accounting for any tier location information. Due to the single dimensional alignment of the tiers connections between the bottom and top tiers have to go through all the tiers in between, e. g., in a design with 5 tiers a connection between the top and the bottom tiers would require 4 3D vias. 3D vias are costly in terms of routing resources and delay and therefore must be minimized. This paper presents a methodology for reducing the number of 3D vias during the circuit partitioning step by avoiding connections between non-adjacent tiers. The proposed algorithm minimizes the total number of 3D vias and long 3D vias while respecting area balance, number of tiers and I/O pins balance. Experimental results show that the number of 3D-Vias was reduced by 19%, 17%, 12% and 16% when benchmark circuits were designed using two, three, four and five tiers.

1. Introduction

The design of 3D circuits is becoming a reality in the VLSI industry and academia. While the most recent manufacturing technologies introduces many wire related issues due to process shrinking (such as signal integrity, power, delay and manufacturability), the 3D technology seems to significantly aid the reduction of wire lengths [1-3] consequently reducing these problems. However, the 3D technology also introduces its own issues. One of them is the thermal dissipation problem, which is well studied at the floorplanning level [4] as well as in placement level [3]. Another important issue introduced by 3D circuits is how to address the insertion of the inter-tier communication mechanism, called 3D-Via, since introduces significant limitations to 3D VLSI design. This problem has not been properly addressed so far since there some aspects of the 3D via insertion problem that seem to be ignored by the literature: 1) all face-to-back integration of tiers imply that the communication elements occupy active area, limiting the placement of active cells/blocks ; 2) the 3D-Via density is considerably small compared to regular vias, which won't allow as many vertical connections as could be desired by EDA tools; 3) timing of those elements can be bad specially considering that a vertical connection needs to cross all metal layers in order to get to the other tier ; 4) 3D-Vias impose yield and electrical problems not only because of their recent and complex manufacture process but also because they consume extra routing resources.

The 3D integration can happen in many granularity levels, ranging from transistor level to core level. While core level and block level integration are well accepted in the literature, there seem to exist some resistance to the idea of placing cells in 3D [6]. One of the reasons is that finer granularity demands higher 3D-Via demand, which might fail to meet the physical constraints imposed by them. On the other hand, the evolution of the via size is going fast and is already viable (for most designs) to perform such integration [2, 5, 11, 13] since we already can build 0.5 μm pitch face-to-face vias [6] and 2.4 μm pitch on face-to-back [5]; we believe that this limitation is more in the design tools side, since those are still not ready to cope with the many issues of 3D-Vias [7, 13].

While it is known that the addition of more 3D-Vias improves wire length [5], this design methodology might fail to solve the issues highlighted above. We believe that EDA tools can play a major role to enable random logic in 3D by minimizing 3D-Vias to acceptable levels. The number of 3D-Vias required in a design is determined by the tier assignment of each cell, which is performed during the cell partitioning. The cell partitioning is usually performed by hypergraph partitioning tools (since it is straightforward to map a netlist into a hypergraph) such as hMetis [8] as done in [2] for the same purpose that is addressed here. On the other hand, hypergraph tools do not understand the distribution of partitions in the space (in 3D circuits they are

distributed along in a single dimension) and fail to provide optimal results. It is important to understand that the amount of resources used is proportional to of the vertical distance of the tiers; in fact, considering that the path from a tier to an adjacent involves regular vias going through all metal layers plus one 3D-Via, it is clear that any vertical connection larger than adjacency might be too expensive in terms of routing resources and delay.

In this paper we want to demonstrate that using a Simulated Annealing based algorithm we can refine the partitioning produced by traditional hypergraph partitioning algorithms further minimize the 3D-Via count while maintain nets vertically shorter and keeping good I/O and area balancing. The remainder of the paper is organized as follow. In Section 2 presents the problem being addressed, section 3 presents how we draft a solution to this problem while comparing it to similar approaches. Section 4 presents the details of our Simulated Annealing algorithm and finally section 5 presents conclusions and directions for the future work.

2. Problem Formulation

Consider a random logic circuit netlist and a target 3D circuit floorplan (including area and number of tiers), compute the partitioning of the I/O pins as well as the partitioning of cells into tiers such that the 3D-Vias count is minimized; be constrained by keeping a reasonable balance of both, I/Os and cells, along the tiers.

3. Partitioning Algorithm and Related Work

Our previous work [9] presents a solution for the proposed problem. In that work, we concentrated solely in improving the I/O pins partitioning and the cells partitioning were naturally following the improved structure producing cut improvements in the order of 5,33% (2 tiers), 8,29 (3 tiers), 9,59% (4 tiers) and 16,53% (5 tiers). We have used hMetis for cell partitioning the while the I/Os were fixed by our method. Experimental results have shown to that the initial I/O placement improves the ability of hMetis to partition cells. In this paper, we will tackle the cells partitioning method as well.

We now propose an iterative improvement heuristic to handle the proposed problem (section 2). The algorithm is inspired on Simulated Annealing, but instead of accepting uphill solutions to avoid local minima [14], our heuristic understands that we can obtain a good initial solution (sufficiently close to the optimal point) using our previous work and now focus on improving it disregarding any possible local minima.

The main difference between our new approach and a hypergraph partitioner such as hMetis is that our approach accounts for the location of the partitions. In fact, in a 3D circuit, the partitions are organized in a line, which implies in the notion of adjacent partitions (that are cheap in terms of cut) and distant partitions (that are expensive since demand extra 3D-Vias). Having the goal of minimizing the 3D-Vias as a whole, we naturally want to overpenalize the cut of distant partitions, which is not done in standard hypergraph partitions. For example, the algorithm in [2, 9] employs hMetis to partition the cells into groups, and in a second stage employs a post-processing method to distribute the partitions in the 1D space (line) such that the number of 3D-Vias is minimized (as illustrated in fig. 1.a). While this approach targets the same goals, it clearly has limited freedom since partitions cannot be changed. The algorithm proposed in this paper allows cells to move from one partition to the other as long as the final cost is reduced, as illustrated in fig. 1.b. The definition of the cost function is responsible for keeping a good I/O pins and cells balance among the different tiers.

4. Improvement Method for Heuristic Partitioning

The proposed algorithm picks an initial solution (the solution obtained from our previous work [9]) and improves it iteratively using random perturbations of the existing solution. The perturbations might be accepted or rejected depending on the cost variation. Any perturbation that improves the current state is accepted and all perturbations that increase the cost is rejected (greedy behavior).

4.1. Perturbation Procedure

The perturbation function designed for our application attempts to move cells across partitions. Although they are random in nature, we perform two different kinds of perturbations for better diversity: single movement or double exchange (swap). The double and single perturbations are alternated with 50% probability. They work as follows:

- The single perturbation can randomly pick a cell or an I/O pin (with 50% probability each) and move it to a different tier (also chosen randomly).
- The double perturbation randomly selects a pair of elements to switch partitions. Each element can be either a cell or I/O pin with 50% of selecting each, totaling 4 different double perturbation combinations, each having 25% probability of happening.

4.2. Cost Function

Any intermediate state of the partitioning process can have its quality measured by a cost function. In the cost function, we model all metrics of interest in a single value that represents the cost.

Our cost function is divided in three distinct parts: a cost v associated to the usage of 3D-Via resources, a value a for the area balance and finally a cost p for the I/O pin balance. The cost reported is a combination of the three parcels; in order to be able to add them together, we normalize each parcel by dividing them from their initial values v_i , a_i and p_i (computed before the first perturbation). In addition, we also impose weights (w_v , w_a and w_p) in order to be able to fine tune the cost function to vias the optimization process, as shown in equation 1.

The values v , a and p are computed as follows.

- For each net, compute the square of the via count; add the computed number of each net to obtain v . The square is applied to highly punish nets having high 3D-Via counts and to encourage short vertical connections.
- To compute a , we first compute the cell area of all tiers; the unbalance cost is a subtraction of the largest by the smallest area.
- The value p is computed similarly to a .

$$c = \frac{(w_v \times v)}{v_i} + \frac{(w_a \times a)}{a_i} + \frac{(w_p \times p)}{p_i} \quad (1)$$

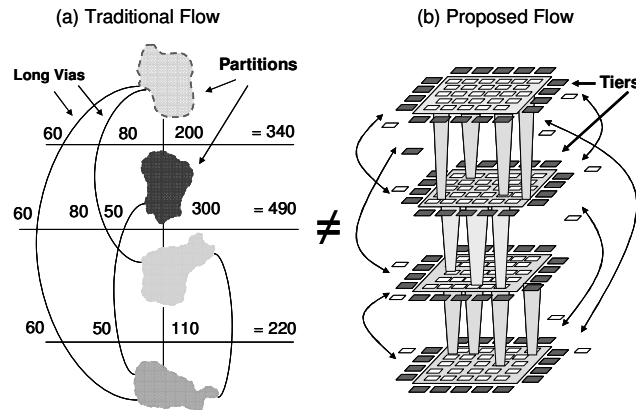


Fig. 1 – Fixed Tiers Method

5. Experimental Results

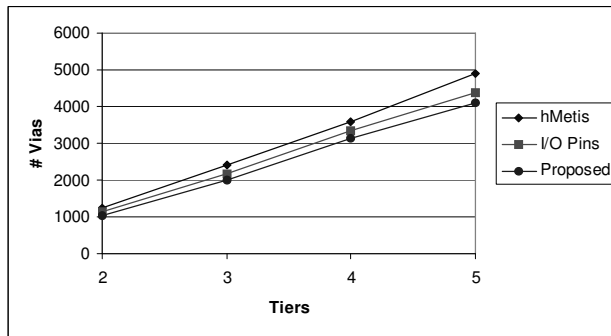


Fig. 2 – Number of 3D-Vias

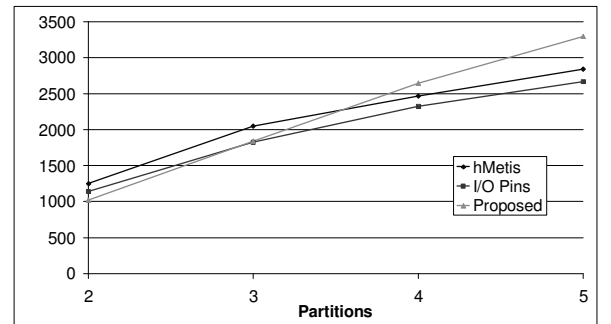


Fig. 3 – Cut quality

Our partition refinement algorithm was compared to a state-of-the-art hypergraph partitioner called hMetis. We have used hMetis to partition the design netlist (including cells and I/O pins) freely into n partitions, where n is also the number of tiers. In a subsequent stage, we would assign the partitions into tiers, as illustrated in fig. 1 inspired in the method performed by [2]. Our previous work [9] would perform a slightly different approach; it would first partition the I/O pins of the block in a way that the I/O balance could be controlled; also, we have a heuristic that is able to cleverly partition the I/Os in a way that they can aid the cell partitioning to reduce the cut. Following the I/O partitioning, we fix the pins and perform hMetis to partition the cells. Since the method is concentrated on the I/Os, we now call it I/O Pins. The method presented here is referred as proposed in the tables and graphs below. Please note that this method starts from the I/O Pins solution and runs our greedy improvement heuristic to refine both the cells and the I/O partitioning.

The experimental setup is as follows. We picked circuits from the ISPD 2004 benchmark suite [12] and partitioned the design into 2, 3, 4 and 5 tiers. The three referred methods (hMetis, I/O Pins and proposed) were attempted. All methods were constrained to distribute area evenly, which resulted in a worst case of 0.1%

unbalance. The I/O balancing is not imposed in the hMetis since it would overconstraint the method. For this reason, hMetis has the worst I/O balancing while the I/O Pins is the best since the proposed method uses a little freedom on the I/O balancing to improve the 3D-Via count.

Fig. 2 shows the total 3D-Via count comparison between the methods. The proposed method obtains the average least amount of 3D-Vias. More specifically, the proposed method produced 3D-Via count average improvement of 19% and 11% compared to hMetis and I/O Pins respectively for 2 tiers, 17% and 8% for 3 tiers, 12% and 6% for four and finally 16% and 7% for 5 tiers. Fig. 3 shows the final partitioning from a hypergraph perspective. The y-axis shows the average cut among the different partitions when the circuits are divided into 2, 3, 4 and 5. It can be observed that the proposed algorithm presents a higher cut value when the number of partitions is larger, however when only two partitions are created the proposed algorithm achieves the smaller cut value than hMetis and I/O pins. The behavior is explained by the cost function used to optimized the final partition set. The proposed algorithm, unlike hMetis and I/O Pins, do not tackle at the cut reduction but in reduction of the total number of vias. When only two partitions are considered the number of vias given by the cut of the partitioning algorithm. When more partitions are created the proposed algorithm increases the number of connections between adjacent partitions in order to reduce the number of connections between non-adjacent ones. This behavior leads to a increase in the partitioning cut number while the total number of vias can be further reduced.

6. Conclusions

This paper presented a method to refine the I/O Pins and cells partitioning into a 3D VLSI circuit. We were able to develop a heuristic iterative improvement algorithm that achieves better partitioning than a simple application of hypergraph partitioners. The method demonstrates that hypergraph partitioners are not well suited for cell and I/O partitioning into 3D circuit because they do not handle long connections properly, affecting the total 3D-Via count. We demonstrated that our heuristic was able to improve the 3D-Via count by considering the positions of each tier within the 3D chip while partitioning the cells among the tiers. Finally, we highlight that our heuristic was able to perform partitioning while keeping area and I/O pin count balanced across tiers.

7. References

- [1] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer and P. D. Franzon; Demystifying 3D ICs: The Pros and Cons of Going Vertical. IEEE Design and Test of Computers – special issue on 3D integration; pp 498-510, Nov.-Dec. 2005.
- [2] C. Ababei, Y. Feng, B. Goplen, H. Mogal, T. Zhang, K. Bazargan and S. Sapatnekar. Placement and Routing in 3D Integrated Circuits. IEEE Design and Test of Computers – special issue on 3D integration; pp 520-531, Nov.-Dec. 2005.
- [3] B. Goplen; S. Sapatnekar; Efficient Thermal Placement of Standard Cells in 3D ICs using Forced Directed Approach. In: International Conference on Computer Aided Design, ICCAD'03, November, San Jose, California, USA, 2003.
- [4] E. Wong; S. Lim. 3D Floorplanning with Thermal Vias. In: DATE '06: Proceedings of the Conference on Design, Automation and Test in Europe, 2006. p.878–883.
- [5] Das, S.; Fan, A.; Chen, K.-N.; Tan, C. S.; Checka, N.; Reif, R. Technology, performance, and computer-aided design of three-dimensional integrated circuits. In: ISPD'04: Proceedings Of The 2004 International Symposium On 59 Physical Design, 2004, New York, NY, USA. Anais. . . ACM Press, 2004. p.108–115.
- [6] Patti, R. Three-dimensional integrated circuits and the future of system-on-chip designs. Proceedings of IEEE, [S.l.], v.94, p.1214–1224, 2006.
- [7] Hentschke, R. et al. 3D-Vias Aware Quadratic Placement for 3D VLSI Circuits. In: IEEE Computer Society Annual Symposium on VLSI, ISVLSI, 2007, Porto Alegre, RS, Brazil. Proceedings. . . Los Alamitos: IEEE Computer Society, 2007. p.67–72.
- [8] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI domain. In Proceedings of 34th Annual Conference on Design Automation, DAC 1997, pages 526–529, 1997.
- [9] Sawicki, S.; Hentschke, Renato ; Johann, Marcelo ; Reis, Ricardo . An Algorithm for I/O Pins Partitioning Targeting 3D VLSI Integrated Circuits. In: 49th IEEE International Midwest Symposium on Circuits and Systems, 2006, Porto Rico. MWSCAS, 2006.
- [10] K. Bernstein; P. Andry; J. Cann; P. Emma; D. Greenberg; W. Haensch; M. Ignatowski; S. Koester; J. Magerlein; R. Puri; A. Young. Interconnects in the Third Dimension: Design Challenges for 3D ICs. In: Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE. 2007 Page(s):562 - 567
- [11] K. Banerjee and S. Souri and P. Kapur and K. Saraswat. 3D-ICs: A Novel Chip Design for Improving Deep Submicrometer Interconnect Performance and Systems on-Chip Integration. Proceedings of IEEE, vol 89, issue 5, 2001.
- [12] ISPD 2004 - IBM Standard Cell Benchmarks with Pads. [http:// www. public. iastate .edu/~nataraj/ISPD04_Bench.html#Benchmark_Description](http://www.public.iastate.edu/~nataraj/ISPD04_Bench.html#Benchmark_Description). Access on Mar 2008.
- [13] R. Hentschke, G. Flach, F. Pinto, and R. Reis, "Quadratic Placement for 3D Circuits Using Z-Cell Shifting, 3D Iterative Refinement and Simulated Annealing," Proc. Symp. on Integrated Circuits and Syst. Des. '06, 220-225.
- [14] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, Science, 1983, 220, pages 671-680.

3D Symbolic Routing Viewer

Érico de Moraes Nunes and Reginaldo da Nóbrega Tavares

nunes.erico@gmail.com, regi@unipampa.edu.br

Engenharia de Computação - UNIPAMPA/Bagé

Abstract

This paper describes a tool for visualization of the symbolic cell routing of digital integrated circuits. The tool allows 3D view of the routing scene from any view angle. Some features such as identification of congested areas and the edition of wires segments are also implemented. It is currently written in C language using OpenGL API for graphics. For user interface, the tool uses the GLUT library.

1. Introduction

Visualization of logic circuits is an important research area in design automation of digital integrated circuits. High-quality graphical tools are used during the circuit project not only to verify the results but also to find design errors and improve design strategies. This paper presents a visualization tool that can be used to help the design of digital integrated circuits. This computer program will be integrated in a package of didactic tools. The didactic package will have a set of logic and physical synthesis tools.

The design of a digital integrated circuit is a very complex task. A huge number of transistors and wires are necessary to construct the circuit [2]. Physical synthesis is one step of the circuit design. Many important tasks can occur during the physical synthesis. For instance, the physical cell placement and the cell routing. The cell placement has to determine the position of the logic cells. The average wire length should be minimized during the execution of the cell placement [1]. Logic cells that have connections among them should be positioned near of each other [1]. Then, the router has to connect all cells of the circuit. An efficient router is able to connect all logic cells without increasing the circuit area [3].

This paper describes a computer tool that can be used to visualize the symbolic routing of digital integrated circuits. This symbolic viewer does not show details about the physical implementation of the circuit, but it can give a 3D view of the routing. The user may visualize the circuit from a different angle of vision during the circuit traversal. The viewer is also able to compute congested areas of the circuit. The computed congested areas are highlighted in red. The tool not only can be used to visualize the circuit in a 3D scenery, but also can be used to insert or remove wire segments between source and destination points. Therefore, this 3D visualization tool is also an editor.

2. Visualization Tool

Wire segments are represented in the 3D viewer by a data structure which represents a connection between two points. The entire circuit is represented in a matrix of points. Each wire segment is loaded, positioned and oriented. Any element can be placed in different wire levels. Each level is represented with a different color. Level bridges, or contacts, are also shown and they have their own color. The integrated editor allows the user to freely edit the result. The tool shows two independent editor instances. Each instance shows the connections of a single level and the vertical connection with the upper level. Some display options can be changed in the configuration text file. Fig. 1 shows the tool flowchart.

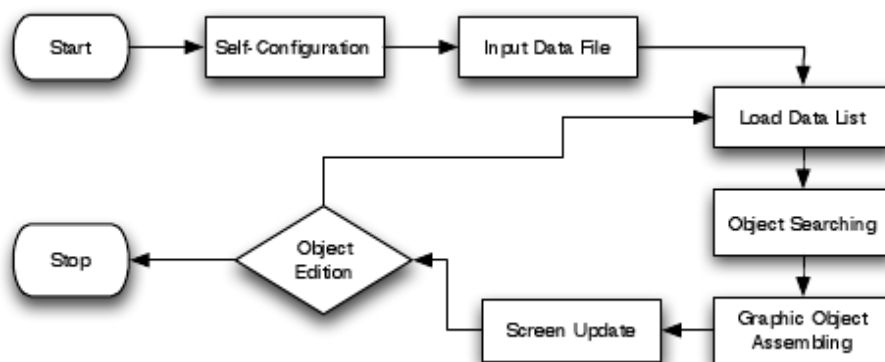


Fig. 1 – Simplified tool flowchart.

At first, the configuration file is loaded and all display options are set. The data structure is allocated as the data input file is read. The base matrix auto adjusts to the required matrix size. The rendering block runs through the structure and draws the routing description within a buffer. Then the buffer is showed to the user. When an event occurs, the tool runs through the structure and draws the objects again. Fig. 2 shows a screen shot of the tool running.

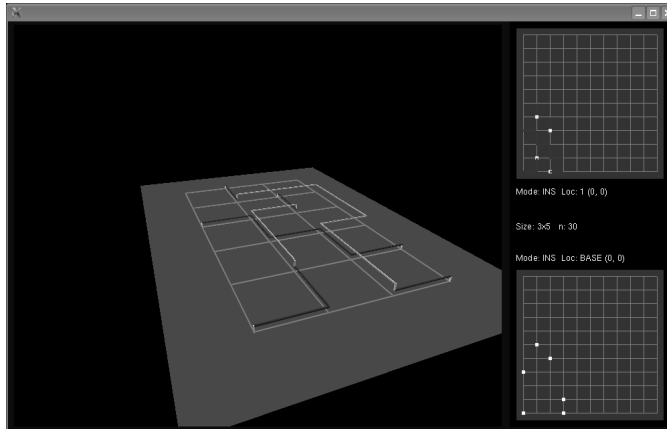


Fig. 2 – Viewer screen shot.

Currently, the input file format accepted is only an own format. It is a raw binary file containing each wire segment position and orientation. The description of the wire is maintained by four integer values: x, y and z displacements and a last value which is interpreted as the orientation. This last value can be -1, 0 or 1. For 0 and 1, the wire is either horizontal or vertical, parallel to the base. For -1, the wire is a bridge between levels, perpendicular to the base.

This structure was chosen because it eliminates impossible wires that could appear if a start point and an ending point were specified. The output file generated by the viewer follows the same file format.

3. Highlighted Paths

This tool allows the user to select a path to highlight. One segment must be selected and all other segments connected to it will be highlighted. This functionality is shown in fig. 3. This feature is useful to check a path and the point it reaches. The highlighted wire is marked with a brighter color.

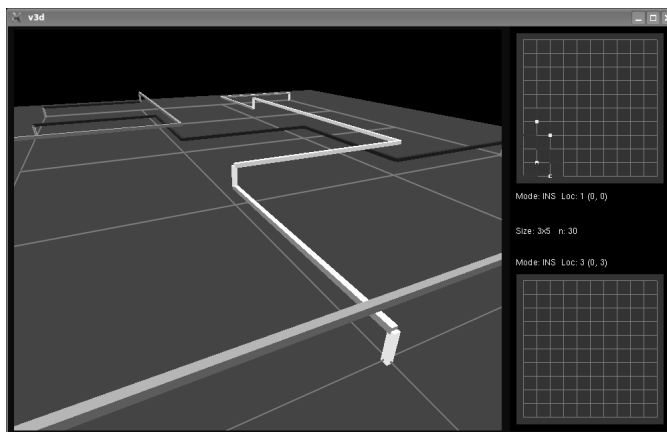


Fig. 3 – Path highlighted feature.

The tool allows the user to identify congested areas. These areas are represented by scaling the segments color according with the connection density at that area. White color represents low density and red color represents high density. It is shown in fig.4. There are two employed methods to measure congested areas: a fast and an accurate. The fast method checks each point and measures the number of connections on that point. The accurate method also visits each point and measures the color using the number of segments on that point and the number of segments at each connected point.

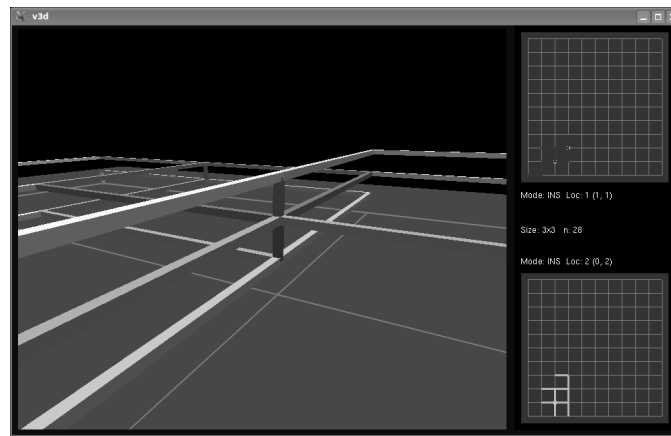


Fig. 4 – Congested zones highlight feature.

The graphic implementation does not interfere in the method used for defining segments attributes as it is done in both highlighting features. This allows the tool to easily get additional features implemented, also allowing for features to be adapted according to the needs of the user.

Other smaller built in features can be listed as self-adjustable, virtually infinite matrix size and displaying of wires number and matrix size for comparison reasons.

With all these features, the tool already offers reasonable number of resources to compare different routing and placing solutions.

4. Data structure

The viewer currently uses an adjacency list as main structure. This structure was chosen due to its ability of defining relations between objects. From any segment, it is possible to have an information of other segment connected to that one. Also, it is interesting because it can grow dynamically. As the viewer does not have a set maximum matrix size, a dynamically growing structure is needed.

Over this structure, algorithms such as Depth-first search can be executed. These algorithms are extensively used for the highlighting features.

5. Camera Management

The most important feature of the tool is the ability of showing the routing as 3D view from any point. In order to have such a feature it is necessary to have a camera management system. The camera management can be defined as a resource that allows the user to modify the point of view. Camera management is not provided by OpenGL [4]. Because of that, it was necessary to find out a mathematical model to implement the 3D camera movement.

The program uses a point to store the camera position and three vectors to specify the camera orientation, as it can be seen in fig. 5. The orientation vector points towards the direction the camera is currently looking at. The up vector says which is the way up and the right vector points towards the right of the camera. These three vectors are used for movimenting around the scenery and changed by user input.

To change the orientation vector for the 3D scenery, one can imagine there is a sphere around the camera position point, such as the one in fig. 5. By having the angle that the camera makes with both the horizontal and vertical, it is possible to calculate a vector inside the sphere as follows [6]:

$$\begin{aligned}x &= \sin(\text{angVert}) * \cos(\text{angHoriz}); \\z &= \sin(\text{angVert}) * \sin(\text{angHoriz}); \\y &= \cos(\text{angVert});\end{aligned}$$

This is used to calculate the orientation vector.

The right vector is used to implement the side movement. This vector results from a cross product between the orientation and the up vector.

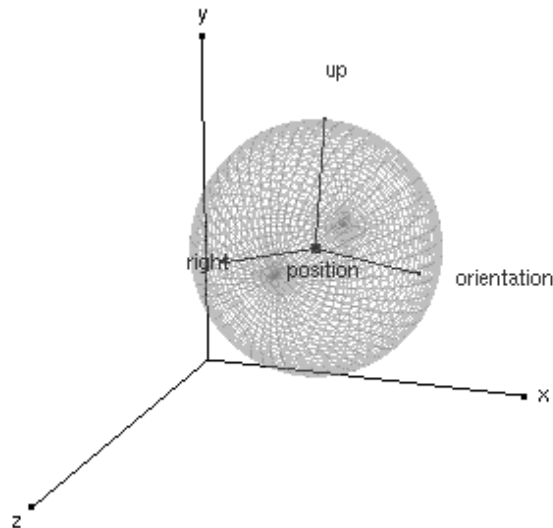


Fig. 5 – Camera representation with spheric coordinates.

6. Conclusions

This tool intends to be working on every platform. OpenGL was chosen to render the routing scenery because it is the fastest platform independent 3D renderer option and is widely supported. This tool also makes use of GLUT, the OpenGL Utility Toolkit. GLUT is a window system independent toolkit, written to hide the complexities of differing window system APIs [5].

It provides OpenGL capable window and input management via keyboard and mouse in a simple and fast way to learn and use. Using GLUT instead of OS specific window and event managers greatly reduces the task of porting the application, in case this gets ever needed.

The 3D symbolic routing viewer is a capable software tool developed to aid integrated circuit design. The tool allows the user to obtain routing information which can be also be used to compare routing and positioning algorithms, and offers a simple editing tool to test different manual solutions. Considering the tool uses symbolic viewing, it also offers good performance gains and cleanness by not being intended to show layout details.

7. References

- [1] R.A. Reis, “Sistemas Digitales: Síntese Física de Circuitos Integrados”, CYTED, 2000.
- [2] J.M. Rabaey, A. Chandrakasan and B. Nikolic, “Digital Integrated Circuits”, Prentice-Hall, 2003.
- [3] N. Sherwani, S. Bhingarde and A. Panyam, “Routing in the Third Dimension: From VLSI Chips to MCMs”, IEEE Press, 1995.
- [4] R.S. Wright, B. Lipchak and N. Haemel, “OpenGL Superbible Fourth Edition”, Addison-Wesley, 2007.
- [5] I. Neider, T. Davis and N. Haemel, “The Official Guide to Learning OpenGL, Version 1.1”, Addison-Wesley, 1997.
- [6] J. Stewart, “Cálculo Vol. 2”, Pioneira, 2009.

Improved Detailed Routing using Pathfinder and A*

Charles Capella Leonhardt, Adriel Mota Ziesemer Junior, Ricardo Augusto da Luz Reis

{ccleonhardt,amziesemer,reis}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul

Abstract

The goal of this work is to improve the previous implementation of the detailed routing algorithm of our physical synthesis tool by obtaining circuits with similar wirelength in less time. The current implementation is based on the Pathfinder algorithm to negotiate routing issues and uses Steiner points for optimization. Because the complexity of the Steiner problem, it was necessary to improve the performance of the signal router to solve larger problems in a feasible time. Our approach was to replace the graph search Lee algorithm by the more efficient A algorithm in the implementation of the shortest path method. The results showed that the execution time is reduced by approximately 11% on average and the wirelength is similar to the previous approach.*

1. Introduction

In detailed routing there is so many options of nodes to use that the routing process become time consuming. The problem is more distinguished in the optimization step because the signal router is used several times to test the candidates for being Steiner nodes.

Since the previous implementation of the routing algorithm [1] was primarily designed for intracell routing, it presented very long execution times when we tried to use it for detailed routing. The goal of his work was to optimize the path search algorithm using specialized algorithms that takes advantage of the regularity of the gridded graph used for detailed routing. So to deal with that was clear that the signal router must be changed from Lee[2] to A*[3], but by doing that, it was also necessary to change the data structure of our previous implementation. The nodes of the graph were arranged in a structure not suitable to the use of the A* because it was impossible to get the distance to the target nodes without a lookup table and the use of a lookup table would add extra processing and usage of memory. Therefore the structure had been changed to a grid that allows simpler implementation of the A*.

The article is organized in 5 sections. Section 2 explains the A* algorithm. In section 3 the algorithm is divided in two parts and explained. The results are presented in section 4. Finally, in section 5 we have the conclusions and future works.

2. A* algorithm

The algorithm is used to find the least-cost path from an initial node to a goal. The key of the algorithm is the cost function $F(x)$ that takes into account the distance from the source $G(x)$ and the estimated cost to the goal $H(x)$ as shown below:

$$F(x)=G(x)+H(x)$$

By using this formula, the neighbor nodes closest to the goal are expanded first, reducing, in average, the number of visited nodes needed to find the target when compared to the Lee approach.

$H(x)$ must be an admissible heuristic, for this reason it can't be overestimated. This is necessary to guarantee that the smallest path can be found. As close $H(x)$ gets to the real distance to the goal, less iterations will be necessary to find the smallest path between the nodes. This cost is calculated using the Manhattan distance between the current node and target node. The Manhattan distance is obtained by the sum of the distances in the 3 axes of the tridimensional grid between the nodes. For our implementation, we considered the distance between two adjacent nodes as unitary.

For example, consider Fig. 1 that shows two searches using Lee and A*. The square identified as S is the source node and the one identified as T is the target node. The gray squares are the nodes that were expanded during the execution of the search algorithms. Using Lee, a total of 85 nodes were expanded until the target be found. Using A*, only 33 nodes were visited until the target node be reached.

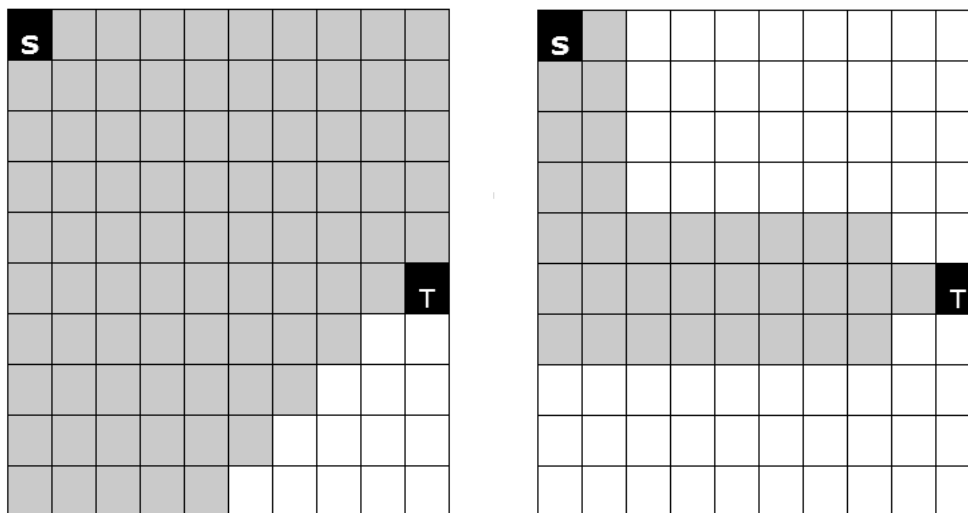


Fig. 1 – Lee search and A* search

3. The algorithm

It's divided in two independent parts, the first one makes a regular routing using Pathfinder and the second one is the optimization phase which makes use of the 1-Steiner Node Algorithm [4].

3.1. Pathfinder routing

In the first phase all the nets are routed using the Pathfinder algorithm [5] to deal with congestion nodes between nets. The A* algorithm is used in the signal router to perform interconnections between nodes that belong to the same net.

The global router calls the signal router many times to connect all nodes from every net of the circuit. If a conflict happens, the global router adjusts the cost of the use of the congested node accordingly to the formula below to make its use more expensive by the signal router.

$$C_n = (B_n + H_n) * P_n$$

This formula takes in account the base cost B_n that is the $F(x)$ cost from A*, the congestion history in the previous iterations H_n and the amount of nets using this point in the current iteration P_n . It makes the signal router look for alternative paths to reach the target and then make the algorithm converge to a feasible solution.

The process happens incrementally and it ends when the global router finds no more conflicts between the nets.

3.2. Optimization phase

After a feasible solution is found during the Pathfinder routing, an optimization phase is executed in the attempt of reducing the wirelength of each net by looking for special nodes called Steiner Nodes. Those nodes, when added to the original net, have the property of reducing the cost of the interconnection by guiding the signal router to use paths that can be shared when connect different sinks. The approach we used was the Iterated 1-Steiner algorithm[4] which iteratively calculates the best Steiner nodes adding them to the original net.

The algorithm applies rip-up and reroute with the addition of the Steiner Nodes. In this step the nodes that belong to other nets aren't considered. With this restriction it's not necessary to deal with conflicts because they don't happen at all. The algorithm ends after no improvement is obtained after re-route all nets.

With the change to a gridded structure, the task of finding Steiner Nodes became easier because heuristics can be used to reduce the number of Steiner Node candidates, like the Hanan grid that is used in our algorithm.

The Hanan grid is obtained by constructing lines through each point that belongs to the net in the 3 dimensions of the grid. In the picture below is shown an example of a hanan grid of three points net in two dimensions. The black nodes are the net nodes and white nodes are the Steiner Node candidates of this net.

In our previous approach, all nodes where considered as candidates.

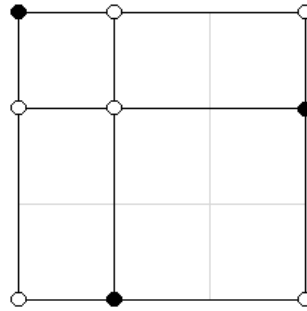


Fig. 2 – Hanan Grid

4. Results

In order to illustrate the results, two tables are presented. Tab. 1 compares the results between a regular Pathfinder routing and other with the optimization step using Steiner nodes. The last one compares the results using A* and Lee as signal router for the same test cases.

For comparison two measures were used, total interconnection cost and execution time. In the case of the use of Steiner technique, the results for time are the sum of regular routing and optimization.

For generating the results was used the tool ICPD[6] in a PowerPC G5 2GHz with 4GB DDR SDRAM.

Tab. 1 - Comparison between regular router and optimization step

Circuit	NrNets	NetSize	Grid	Pathfinder Routing		Optimized Routing		Comparison	
				Cost	Time(s)	Cost	Time(s)	Cost(OR/PR)	Time(OR/PR)
test1	5	4	10x10	691	0.43	682	0.92	0.987	2.15
test2	6	8	30x30	2977	15.50	2920	127.40	0.981	8.22
test3	10	7	30x30	4068	19.40	4012	88.59	0.986	4.57
test4	12	9	30x30	5684	61.52	5592	177.96	0.984	2.89
test5	17	10	40x40	10855	325.70	10534	806.72	0.970	2.48
test6	20	13	40x40	15835	1109.87	15516	1601.18	0.980	1.44
								0.981	3.62

From tab. 1 it's noticeable that the total interconnection cost decreased by 1.9% in average and execution time increased by 262%. As expected, the optimization step improved the quality of the result but because of its high complexity the execution time also significantly increased.

Another conclusion is that for the same grid size as more the number of nets it has, less likely to be optimized it is. The reason is that each net produces obstacles to the others reducing the number of nodes that can reduce its wirelength.

Tab. 2 - Comparison between the signal routers

Circuit	NrNets	NetSize	Grid	Time(s)		Comparison
				Lee	A*	A*/Lee
test1	5	4	10x10	0.92	0.92	1.00
test2	6	8	30x30	127.51	127.40	1.00
test3	10	7	30x30	101.64	88.99	0.88
test4	12	9	30x30	199.76	177.96	0.89
test5	17	10	40x40	901.67	806.72	0.89
test6	20	13	40x40	2475.90	1601.18	0.65
						0.88

From tab. 2, we notice that the time demanded for executing of the regular routing and then the optimization step is 12% smaller using A* than using Lee as the signal router for this set of tests. Another thing that came from the results is that as bigger is the circuit, greater is the difference between the signal routers.

In fig. 3 is shown the routing result of the tool using A* with optimization step.

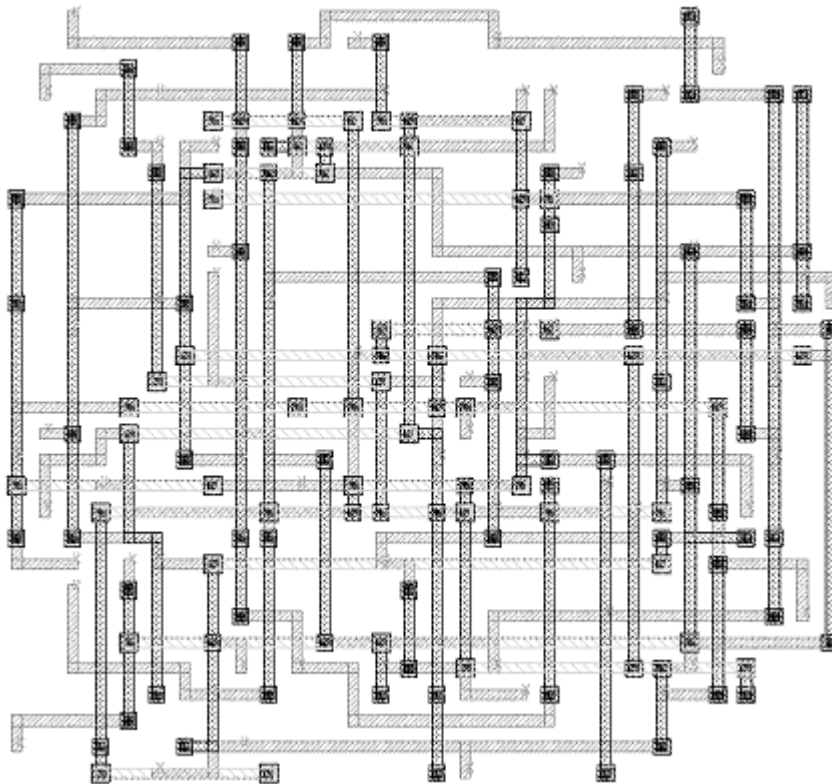


Fig. 3 – The final result of the routing using the ICPD tool

5. Conclusions and future works

In this work we presented an improved algorithm for detailed routing using A* as signal router from an initial intracell router. The algorithm keep the good results in decreasing the interconnection cost and achieve substantially better results in decreasing the execution time of the previous approach.

The next step is to include new techniques of routing like those implemented in FastRoute[7]. Another future work is to test the implementation using academic benchmarks to compare with recognized tools.

6. References

- [1] L. Saldanha, C. Leonhardt, A. Ziesemer, R. Reis, "Improving Pathfinder using Iterated 1-Steiner algorithm", in South Symposium on Microelectronics, Bento Gonçalves, May 2008.
- [2] C. Y. Lee, "An algorithm for path connections and its applications," in IRE Transactions on Electronic Computer, vol. EC-10, number2, Pp. 364-365, 1961.
- [3] P. E. Hart, N. J. Nilsson, "A Formal basis for the heuristic determination of minimum cost paths", in IEEE Transactions of Systems Science and Cybernetics, vol. SSC-4, number2, July 1968, Pp. 100-107.
- [4] A. Kahng, "A new class of iterative Steiner tree heuristics with good performance," IEEE, 1992, Los Angeles.
- [5] L. McMurchie, C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in International Symposium on Field Programmable Gate Arrays, Monterrey, Feb. 1995.
- [6] A. Ziesemer, C. Lazzari, R. Reis, "Transistor level automatic layout generator for non-complementary CMOS cells", in Very Large Scale Integration – System on Chip(VLSI-SOC), 2007, Pp. 116-121.
- [7] M. Pan, C. Chu, "FastRoute: A step to integrate global routing into placement", in International Conference on Computer Aided Design, 2006, Pp. 464-471.

A New Algorithm for Fast and Efficient Boolean Factoring

¹Vinicius Callegaro, ²Leomar S. da Rosa Jr, ¹André I. Reis, ¹Renato P. Ribas
 {vcallegaro, andreis, rpribas}@inf.ufrgs.br, leomarjr@ufpel.edu.br

¹Instituto de Informática – Nangate/UFRGS Research Lab, Porto Alegre, Brazil

²Departamento de Informática – IFM – UFPel, Pelotas, Brazil

Abstract

This paper presents a new algorithm for fast and efficient Boolean factoring. The proposed approach is based on kernel association, leading to minimum factored forms according to a given policy cost during the cover step. The experimental results address the literal count minimization, showing the feasibility of the proposed algorithm to manipulate Boolean functions up to 16 input variables.

1. Introduction

Factoring Boolean functions is one of the fundamental operations in logic synthesis. This process consists in deriving a parenthesized algebraic expression or factored form representing a given logic function, usually provided initially in a sum-of-products (SOP) form or product-of-sums (POS) form. In general, a logic function can present several factored forms. For example, the SOP $f = a*c + c*d + b*d$ can be factored into the logically equivalent forms $f = c*(a+d) + b*d$ and $f = a*c + d*(b+c)$. The problem of factoring Boolean functions into more compact logically equivalent forms is one of the basic operations in the early stages of logic synthesis. In some design styles (like standard CMOS) the implementation of a Boolean function corresponds directly to its factored form. In other words, each literal in the expression will be converted into a pair of switches to compose the transistor network that will represent the Boolean function. Thus, it is desired to achieve the most economical expression regarding the number of literals in order to obtain the most reduced transistor network. This step guarantees, for instance, that final integrated circuit will not present area overhead [1]. Other benefits of this optimization step may be the delay and power consumption minimization [2].

Generating an optimum factored form (a shortest length expression) is an NP-hard problem. According to Hachtel and Somenzi [3], the only known optimality result for factoring (until 1996) is the one presented by Lawler in 1964 [4]. Heuristic techniques for factoring achieved high commercial success. This includes the *quick_factor* and *good_factor* algorithms available in SIS tool [5]. Recently, a factoring method that produces exact results for read-once factored forms has been proposed [6] and improved [7]. However, the *IROF* algorithm [6,7] fails for functions that cannot be represented by read-once formulas. The *Xfactor* algorithm [8,9] is exact for read-once forms and produces good heuristic solutions for functions not included in this category. Another method for exact factoring based on Quantified Boolean Satisfiability (QBF) [10] was proposed by Yoshida [11]. The *Exact_Factor* [11] algorithm constructs a special data structure called eXchange Binary (XB) tree, which encodes all equations with a given number n of literals. The XB-tree contains three different classes of configurable nodes: internal (or operator), exchanger and leaf (or literal). All classes of nodes can be configured through configuration variables. The *Exact_Factor* algorithm derives a QBF formula representing the XB-tree and then compares it to the function to be factored by using a miter [12] structure. If the QBF formula for the miter is satisfiable, the assignment of the configuration variables is computed and a factored form with n literals is derived. The exactness of the algorithm derives from the fact that it searches for a read-once formula and then the number of literals is increased by one until a satisfiable QBF formula is obtained.

This paper presents an exact factoring algorithm based on kernel association. The experimental results address the literal count minimization, showing the feasibility of the proposed algorithm to manipulate Boolean functions up to 16 input variables. The main contribution of this work over the above mentioned heuristic solutions [5,6,7,8,9] is the ability of delivering shorter length expressions in terms of literals. When compared to Yoshida's approach [11], this algorithm is able to deliver similar solutions without using QBF. The straightforward process only consists in generating and combining kernels to feed a cover table that will provide sub-expressions that could be used to compose factored forms.

The remaining of this paper is organized as follows. Section 2 presents the proposed algorithm for factoring. The results are presented in Section 3. Finally, Section 4 discusses the conclusions and future works.

2. Proposed Algorithm

The proposed algorithm to achieve minimum factored forms is divided in a sequence of well defined execution steps. The next subsections will describe the algorithm and illustrate its functionality.

2.1. Converting the Input Expression to a SOP Form

The first step consists in converting any Boolean expression to a SOP form. This is required because the proposed algorithm uses the product terms from a SOP to find portions with identical literals that will be manipulated in the next step. The conversion is done through a BDD (Binary Decision Diagram) using well established algorithms [13]. The input Boolean expression is used to create a BDD structure and, afterward, all relevant paths on the BDD are extracted to compose product terms. The SOP is built using sum of these products.

2.2. Terms Grouping

From a SOP form it is possible to perform the identification of equal portions between products. Considering the example of Eq.1, the literal e is common for the products $e*f$ and $e*g*h$. In this case a new term $e*(f+g*h)$ can be built representing the same logical functionality of those two original products. The same occurs between products $b*c$ and $a*c$, where a new term $c*(a+b)$ can be generated. Notice that the literal i cannot be grouped to others since it is unique on the Eq.1.

$$f = b*c + a*c + e*f + e*g*h + i \quad (\text{Eq.1})$$

This step of the algorithm is executed recursively because when generating new terms other groupings become possible. Considering the example of Eq.2, on the first pass the Eq.3 will be returned. Applying the algorithm recursively in the sub product $(c*e + c*f*g + c*f*h + d*e + d*f*g + d*f*h)$ from Eq.3, the method will return Eq.4. At this point, the optimized found terms will be used to compose equivalent expressions of the original one (Eq.2), as illustrated by Eq.5.

$$f = b*c*e + b*c*f*g + b*d*e + b*d*f*g + b*d*f*h \quad (\text{Eq.2})$$

$$f = b*(c*e + c*f*g + c*f*h + d*e + d*f*g + d*f*h) \quad (\text{Eq.3})$$

$$f = (f*(h + g) + e)*(d + c) \quad (\text{Eq.4})$$

$$f = b*((f*(h + g) + e)*(d + c)) \quad (\text{Eq.5})$$

All set of terms returned by this step will be used in the sequence. The number of returned terms will depend on the possibility of grouping different portions of the input equation.

2.3. Kernels Sharing

In order to perform a fine optimization, all terms returned by the previously step are analyzed and shared when it is possible. Taking into account a set of terms $\{(a*(c+d)), (b*(c+d)), (c*(a+b)), (d*(a+b))\}$, each term is divided in kernels as illustrated in Tab.1.

Tab. 1 – Kernels from a set of terms.

#	Terms	Kernel 1	Kernel 2
1	$a*(c+d)$	a	$(c+d)$
2	$b*(c+d)$	b	$(c+d)$
3	$c*(a+b)$	c	$(a+b)$
4	$d*(a+b)$	d	$(a+b)$

By evaluating the lists of kernels, the algorithm tries to find equivalent ones that are candidates to be shared with others. Considering kernel a , for instance, it is possible to observe that it cannot be shared with other kernel. However, kernel $(c+d)$ can be shared with another identical kernel. Thus, the algorithm shares all candidates and builds new terms. In this example a new term $(a+b)*(c+d)$ is obtained. The set of new terms generated during this step is $\{a*(c+d), b*(c+d), c*(a+b), d*(a+b), (c+d)*(a+b)\}$. Notice that if some repeated term is generated, then it is not added into the set.

2.4. Covering Step

The last step of the proposed algorithm receives all terms (sets) generated during the steps presented in subsections 2.1, 2.2 and 2.3. All terms are put together in order to compose a cover table. After that, a standard covering algorithm [14] is applied and the best solution is delivered as the factored expression. It is important to say that in this paper the number of literals was the cost to be minimized. Nevertheless, other costs may be considered to be minimized (like number of products or number of sums in the expression, for instance).

3. Results

The algorithm described above was implemented in Java language. In order to validate the proposed method, the set of Boolean functions present in genlib.44-6 [5] was used. A total of 3321 logic functions were extracted from the library to feed the execution flow. In the sequence, the Boolean expressions were factored, one by one, using the proposed algorithm. The experiment was performed in a 1.86Ghz Core 2 Duo processor with 2Gb memory, CentOS 5.2 Linux operating system and Java virtual machine v.1.6.0.

Tab. 2 shows some factored equations obtained with the proposed approach. It is possible to see that the method is able to deliver exact expressions even for functions with a reasonable number of literals.

Tab. 2 – Results of some factored expressions.

Input SOP	Factored Expression
$!b*!d*!f*!g+!b*!d*!e+!b*!c+!a$	$!(a*(b+c*(d+e*(f+g))))$
$!b*!g*!h*!i+!b*!d*!f+!b*!d*!e+!b*!c*!f+!b*!c*!e+!a$	$!(a*(b+(c*d+e*f)*(g+h+i)))$
$!c*!d*!g*!i+!c*!d*!g*!h+!c*!d*!e*!f+!b*!g*!i+!b*!g*!h+!b*!e*!f+!a$	$!(a*(b*(c+d)+(e+f)*(g+h*i)))$
$!g*!h*!m*!n+!g*!h*!k*!l+!g*!h*!i*!j+!e*!f*!m*!n+!e*!f*!k*!l+!e*!f*!i*!j+!c*!d*!m*!n+!c*!d*!k*!l+!c*!d*!i*!j+!a*!b$	$!((a+b)*((c+d)*(e+f)*(g+h)+(i+j)*(k+l)*(m+n)))$
$!d*!n*!o*!p+!d*!k*!l*!m+!d*!h*!i*!j+!d*!e*!f*!g+!c*!n*!o*!p+!c*!k*!l*!m+!c*!h*!i*!j+!c*!e*!f*!g+!b*!n*!o*!p+!b*!k*!l*!m+!b*!h*!i*!j+!b*!e*!f*!g+!a*!n*!o*!p+!a*!k*!l*!m+!a*!h*!i*!j+!a*!e*!f*!g$	$!(a*b*c*d+(e+f+g)*(h+i+j)*(k+l+m)*(n+o+p))$

Tab. 3 shows the results concerning CPU execution time. The results are presented according to the number of different variables present in the input equations. First column describes the number of variables. Second column describes the number of expressions to be factored. Third column presents the total time for factoring. Finally, fourth column presents the average time for factoring the expressions. The results are shown in seconds.

The average time for factoring Boolean expressions up to 10 different variables was less than a second. This average time grows up when more variables are added to the expression. The average time for factoring expressions with 16 different variables was around 181 seconds.

Tab. 3 – CPU time for the set of functions from genlib 44-6.

# variables	# of expressions	Total time (s)	Average time (s)
1	1	0.032	0.032
2	2	0.121	0.060
3	4	0.278	0.069
4	9	0.707	0.078
5	21	2.064	0.098
6	52	6.956	0.133
7	113	26.193	0.231
8	224	85.028	0.379
9	369	205.222	0.556
10	523	481.078	0.919
11	602	1267.666	2.105
12	588	7923.146	13.474
13	442	20733.131	46.907
14	255	25760.855	101.022
15	94	14978.739	159.348
16	22	3985.341	181.151

4. Conclusions and Future Works

This paper presented a new algorithm for fast and efficient Boolean factoring. Experimental results demonstrated that the algorithm is feasible to manipulate Boolean expressions up to 16 input variables in very short CPU execution time. It is able to deliver minimum factored forms in terms of literal count.

As future works it is intended to expand the cover policies in order to allow the algorithm to provide factored forms concerning minimization of other costs (like products or sums length). Also, comparisons with other methods available in the literature will be performed as the next step.

5. References

- [1] Brayton, R. K. 1987. Factoring Logic Functions. IBM Journal Res. Develop., vol. 31, n. 2, pp. 187-198.
- [2] Iman, S. and Pedram, M. Logic Extraction and Factorization for Low Power. DAC'95. ACM, New York, NY, pp. 248 – 253.
- [3] Hachtel, G. D. and Somenzi, F. 2000. Logic Synthesis and Verification Algorithms. 1st. Kluwer Academic Publishers.
- [4] Lawler, E. L. 1964. An Approach to Multilevel Boolean Minimization. J. ACM 11, 3 (Jul. 1964), pp. 283-295.
- [5] Sentovich, E.; Singh, K., Lavagno; L., Moon; C., Murgai, R.; Saldanha, A., Savoj; H., Stephan, P.; Brayton, R.; and Sangiovanni-Vincentelli, A. 1992. SIS: A system for sequential circuit synthesis. Tech. Rep. UCB/ERL M92/41. UC Berkeley, Berkeley.
- [6] Golumbic, M. C.; Mintz, A.; Rotics, U. 2001. Factoring and recognition of read-once functions using cographs and normality. DAC '01. ACM, New York, NY, pp. 109-114.
- [7] Golumbic, M. C.; Mintz, A.; Rotics, U. 2008. An improvement on the complexity of factoring read-once Boolean functions. Discrete Appl. Math. Vol. 156, n. 10 (May. 2008), pp. 1633-1636.
- [8] Golumbic, M. C. and Mintz, A. 1999. Factoring logic functions using graph partitioning. ICCAD '99. IEEE Press, Piscataway, NJ, pp. 195-199.
- [9] Mintz, A. and Golumbic, M. C. 2005. Factoring boolean functions using graph partitioning. Discrete Appl. Math. Vol. 149, n. 1-3 (Aug. 2005), pp. 131-153.
- [10] Benedetti, M. 2005. sKizzo: a suite to evaluate and certify QBFs. 20th CADE, LNCS vol. 3632, pp. 369–376.
- [11] Yoshida, H.; Ikeda, M.; Asada, K., 2006. Exact Minimum Logic Factoring via Quantified Boolean Satisfiability. ICECS '06. pp. 1065-1068.
- [12] Brand, D. 1993. Verification of large synthesized designs. ICCAD 93. IEEE, Los Alamitos, CA, pp. 534-537.
- [13] Drechsler, R.; Becker, B. Binary Decision Diagrams: Theory and Implementation. Boston, USA: Kluwer Academic, 1998.
- [14] Wagner, F.R.; Ribas, R.; Reis, A. Fundamentos de Circuitos Digitais. Porto Alegre: Sagra Luzzatto, 2006.

PlaceDL: A Global Quadratic Placement Using a New Technique for Cell Spreading

Carolina Lima, Guilherme Flach, Felipe Pinto, Ricardo Reis
{cplima, gaflach, fapinto, reis}@inf.ufrgs.br

UFRGS

Abstract

In this paper, we present a global quadratic cell placer called PlaceDL using a simple diffusion-based cell spreading technique. Our results show that the new spreading technique can achieve 4% better results than Capo cell placer taking 2x less time. Comparing to FastPlace, our placer takes 5x more time to reach the same results, however our method is entirely described, which allows its easy reproduction.

1. Introduction

With the technological advances, the number of components that can be integrated inside a silicon chip is always increasing. Therefore new CAD tools must be developed to deal with thousand, even millions components at same time. The placement is the physical design step that defines the position of the components inside the chip area. A placer must scale with this huge number of components while given good results.

In this work we present PlaceDL, a fast global placer, which can tackle thousand of cells in reasonable time. PlaceDL is developed under quadratic placement philosophy, which is one of the most successful methods for cell placement of large circuits.

We based our placer on FastPlace [1], a quadratic placement, but also presenting a new technique for cell spreading. FastPlace hides some important implementation details, so the reported results cannot be reproduced. On the contrary, we present every important issues so that one can reach the same results as outlined in this paper.

The contributions of this work are:

1. To present a fast global cell placer which achieves better or similar results with respect to Capo [2], another well know quadratic placement, and FastPlace [1] in reasonable time.
2. Outline every important implementation detail to allow one to reproduced our experimental results.

The remainder of this paper is as follow. First we introduce the placement step and quadratic placement technique. We show how quadratic placement can be viewed as spring system. Next, we show our placement flow. And finally, we conclude with some results.

2. Cell Placement

Placement is one of the physical synthesis step. In this step the position of cells are defined. The input of placement problem is a circuit description that contains the components dimensions and their connectivity. The connectivity information can be viewed as a hypergraph, called netlist, where nodes represent the cells and hyperedges represent the cell connectivity. A hyperedge defines a net, which in its turns define a set of cells that must be connected by wires in final circuit project.

The placer must generate as output legal and feasible positions for every cell. We call a legal placement result, a placement in which there is no overlap among cells and all cells lie in valid positions inside the placement area. A feasible placement is not a straightforward definition, however it means that a legal placement must be done taking into account the following steps and design constraints as wire congestion and maximum delay. Notice that it is trivial to generate a legal placement randomly, but such a solution has a high probability to produce a non-feasible circuit due to the excessive wire needed to connect cells even for small circuits.

Generally, a placer aims to reduce the total wire length. Reducing the total wire length tends to reduce average path delay and average wire congestion so it is a good measure for placement quality. However, as the delay and congestion reductions are indirect, reducing wire length cannot be taken as an accurate technique for dealing with these factors.

3. Quadratic Placement

Quadratic placers formulate the placement problem using a mathematics model, which minimizes the sum of the quadratic euclidean distance between connected cells. In most cases, quadratic models ignore the overlap among cells. This assumption simplifies the problem, but generates non-legal placements results. Two constraints are required when casting the placement in the quadratic formulation:

- **Cell-to-Cell Connections:** All hyperedges must be broken into regular edges, that is, the hypergraph, which represents the netlist, must be translated to a graph. In this work, in order to broken the hyperedges, we used the hybrid model presented in FastPlace [1].
- **Fixed Components:** Without fixed components, the quadratic placement system has infinite solutions. So we must know a priori the position of some circuit components. In general, quadratic placer requires as input the position of circuit pads. Therefore the circuit components are divided in two sets: mobile and fixed. For simplicity, we will call cell the mobile components, and terminals the fixed ones. Besides, we will call node when we want to refer to a component and does not matter if it is mobile or fixed.

3.1. Quadratic Placement as a Spring System

The quadratic placement can be viewed as a spring system [3] where the spring are modeled by Hooke's law, $F = -kd$. In this case, the spring system equilibrium state gives the position of cells, which minimizes the sum of quadratic wire lengths. In the spring system, connections between cells are represented by springs, where the spring constant, k , represents the connection weight.

3.2. Linear System

Lets take a circuit composed by n cells and m terminals. We will use iterators i and j to sweep cells, i.e, $i, j = 1, 2, \dots, n$, and use iterator k to sweep nodes (cells and terminals), i.e, $k = 1, 2, \dots, n + m$.

In order to find the solution that minimizes the sum of quadratic wirelength, we need to solve two linear systems: $Qx = bx$ and $Qy = by$ where Q is a $n \times n$ matrix and b a $n \times 1$ vector. Note that we can minimize the sum of quadratic wirelength by solving separately the x and y dimensions. So hereafter we pay attention only for $Qx = bx$ system.

The matrix $Q = [q_{ij}]$ represents the cell connectivity. Each row of matrix Q is related to a cell. The off-diagonal elements represent the connection between a pair of cells. Diagonal elements are affect by both cell-to-cell connections and cell-to-node connections.

The elements of matrix Q are obtained as follows. For $i \neq j$, we have $q_{ij} = -w_{ij}$ where w_{ij} is the connection weight which connects the cell i and cell j . If there is no connection between two cells, we have $w_{ij} = 0$, and, then, $q_{ij} = 0$. For $i = j$, we have

$$w_{ii} = \sum w_{ik}$$

where w_{ik} is the connection weight between cell i and node k . That is, q_{ii} is the sum of the connection weight that cell i belongs to.

The right hand side vector, $b_x = [b_i]$, is defined as follows. For each cell i ,

$$b_i = \sum x_k w_{ik}$$

where x_k is the x position of terminal k and w_{ik} is connection weight connecting cell i and terminal k . The connection weight utilized in this work is computed in the same way as in FastPlace.

To solve the linear system we use pre-conditioned conjugate gradient [4], using Cholesky as pre-conditioner. We set its error to $1e-6$.

4. PlaceDL: Placement Flow

The main goal of PlaceDL is to generate quickly a global placement with low overlap and a good wirelength. The whole flow is composed of two main steps: diffusion under control and wirelength improvement.

5. Diffusion under Control

To solve mathematically the wirelength minimization concurrently with the overlapping problem is computationally expensive. In diffusion under control step the PlaceDL iterate two methods under control step the PlaceDL iterate two methods with opposite objectives: (1) Wirelength minimization solving the linear system. (2) Overlap minimization using a simulating diffusion algorithm, moving the cells from a pretty dense region to a less dense region.

The wirelength minimization clusters the cells, reducing the total wirelength, but increasing the cells superposition. But minimizing the overlaps the technique spread the cells, increasing the wirelength. Hereafter will be explained how these opposite methods are used in PlaceDL.

To start the cells diffusion we use the linear system solution that minimizes the wirelength. The diffusion is an iterative process. In each iteration for each region all bin utilization are calculated, and then the cells are

spreads based on density gradient (explanation in chapter 5.2). The cells are diffusing in small steps. The process stops when reach a 25% reduction in the more dense region or when occur a utilization increase in relation at before iteration. Some little increases in maximum utilization could happen because the diffusion process is calculated discretely.

5.1. Utilization Grid

The utilization grid divides the circuit area in bins with dimension binwidth x binheight how showed in fig. 1.

The number of lines and columns (n,m) is achieved taking the placement total area and dividing in squared area sizing l. Normally this squared area contents 2 cells:

$$l = \sqrt{2 \times \text{avgCell}_{\text{width}} \times \text{avgCell}_{\text{height}}}$$

Where avgCellwidth and avgCellheight means the average width and height of circuits cells. Finding the base size of the regions, we get the lines and columns number of regular grid:

$$(n, m) = \left(\left\lfloor \frac{H}{l} \right\rfloor, \left\lfloor \frac{W}{l} \right\rfloor \right)$$

Where W and H are the width and height of the placement area. Finally, to avoid regions with different sizes in the circuit border is necessary to fix the regions. The next equation fixes the size regions.

$$(\text{bin}_{\text{width}}, \text{bin}_{\text{height}}) = \left(\frac{W}{n}, \frac{H}{m} \right).$$

After the grid has been built the utilization of each region is calculated. The region utilization is defined adding the overlap area between the cells and the regions. One important point to be observed is that a cell could add the utilization of more than one region. The density of each region is obtained dividing the utilization by the area of the region.

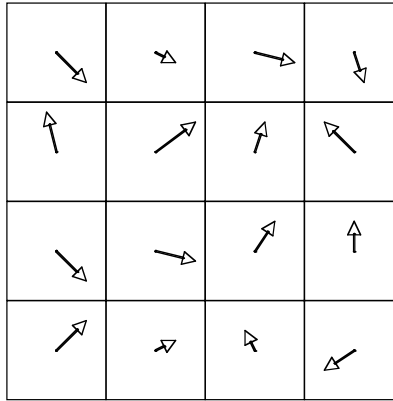


Fig. 1 – Utilization Grid and Diffusion Gradient.

5.2. Diffusion Gradient

At cell spreading, a cell must move to a place which balances the densitys of the original region and of its neighborhood.

The diffusion gradients are vectors from each region's center that point to the direction of where the cells at this region must go and with how power. In fact, the diffusion gradient is a force and is calculated as the follows:

For each one of the eight neighbors, we have one unitary vector from the region's center with the direction of the neighbor. Therefore the powerful of the vectors are calculated for the equation:

$$w = \max \left\{ \frac{U_{\text{central}} - U_{\text{vizinho}}}{\max U}, 0 \right\}.$$

Then, the weighted average of 8 vectors is calculated and the resultant vector is defined the region diffusion gradient.

5.3. Adding Spreading Forces

After the diffusion process, spreading forces need to be added to the system to avoid cells to collapse back to their old position (before diffusion). We modify the system by adding forces that push cells from their original position to the new position defined by the diffusion process.

As mentioned previously, we can view the placement problem as a string system. Therefore, to push a cell to its new position we add a string that connects the cell the circuit boundary as shown in fig. 2. The position where the spring is connected to the boundary, lie on the intersection between the circuit boundary and the line that cross the old and new cell position.

Next we explain the diffusion force, $F = (F_x, F_y)$, experienced by a cell that is moving from (x_{old}, y_{old}) to position (x_{new}, y_{new}) . Since F_x and F_y are calculated in similar way, we concentrate on F_x only. So we have

$$F_x = \mu \times (x_{new} - x_{old});$$

where μ is the diffusion coefficient, which is computed by equation:

$$\mu = 40 \times \left(1 - \exp\left(\frac{-10}{\max U}\right) \right)$$

Let $\max U$ be the maximum region density. Notice μ change as $\max U$ has low values when $\max U$ is larger and increases as $\max U$ decreases.

This variation on coefficient from low to high values avoids that cells move far away its new position at beginning of the diffusion process when regions may have high gradients. Move cells far away from its original position may increase a large amount the total wirelength. As cells spread evenly over circuit area, the distance that cells move are reduced, so the diffusion coefficient can have high values.

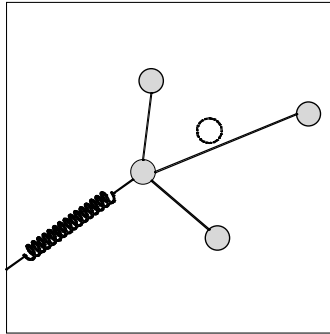


Fig. 2 – Wirelength improvement technique.

Finally the new spreading force is added to the system. Since the new spring connects a cell and a fixed position, only diagonal and the right hand side vector must be changed. The matrix diagonal is added to

$$\beta = \frac{\sqrt{F_x^2 + F_y^2}}{d};$$

where d is the euclidean distance from the original to the new cell position. To the respective element on the right hand side vector, βx_{pin} is added where x_{pin} represents the x position where the spring is connected to the circuit boundary.

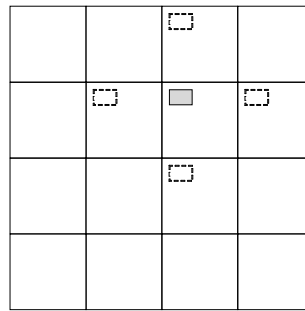


Fig. 3 – Adding Spreading Forces.

6. Improving Wirelength

The controlled diffusion process is efficient to initialize the cells spreading, defining a relative position between them. However as the cells spread over most of the area of positioning, the growth in wirelength becomes higher at each iteration, making the process of diffusion controlled ineffective.

So, another method is necessary for the spreading of the cells after they have taken most of the circuit. In PlaceDL we use a method based in the local iterative refinement, which is used in FastPlace placer.

In this technique, we use again the utilization grid presented above, the bins. Then, sequentially, each cell is moved to the same relative position in the neighboring regions to the north, south, east and west as shown in fig. 3. For each new position a score is calculated. This score takes into account the variation in the length of wire based on the semi-perimeter and the change in use. The variation of the wirelength is weighted with 0.5 and the variation of use with 0.25.

The cell is then moved to neighbor with the bigger positive score. If all scores are negative, the cell will be in the current region.

The process of improvement of wire is iterative and varies the size of the regions of regular grade used in the dissemination, using about 10x to 2x that size. This allows the cells to be moved by larger distances.

For each size of regular grade, the cells are initially moved to areas two regions distant, and then to the regions immediately adjacent. For a given size, the process stops when the improvement in wirelength is greater than 1% and there is reduction in maximum use.

Tab. 1 – Results w.r.t Capo and FastPlace.

	Half-Perimeter Wirelength					Runtime(sec)				
	Capo (X1e6)	FastPlace (X1e6)	PlaceDL (X1e6)	Capo PlaceDL	FastPlace PlaceDL	Capo	FastPlace	PlaceDL	Capo PlaceDL	FastPlace PlaceDL
ibm01	1.86	1.84	1.94	0.96	0.95	1m 54s	8s	35s	3.26x	-4.35x
ibm02	4.10	3.82	3.88	1.06	0.98	3m 24s	17s	1m 32s	2.22x	-5.56x
ibm03	5.34	5.15	5.34	1.00	0.96	4m 01s	17s	1m 18s	3.09x	-4.55x
ibm04	6.26	6.04	6.46	0.97	0.93	4m 58s	21s	1m 38s	3.04x	-4.76x
ibm05	10.59	10.56	10.25	1.03	1.03	5m 17s	15s	2m 02s	2.60x	-8.13x
ibm06	5.87	5.30	5.35	1.10	0.99	5m 49s	27s	1m 35s	3.67x	-3.52x
ibm07	9.60	8.60	9.50	1.01	0.91	8m 49s	44s	2m 54s	2.76x	-3.95x
ibm08	10.28	9.54	9.69	1.06	0.98	9m 12s	47s	3m 53s	3.02x	-4.96x
ibm09	10.46	10.46	10.68	0.98	0.98	10m 39s	54s	4m 11s	2.55x	-4.65x
ibm10	19.71	18.47	18.89	1.04	0.98	13m 22s	1m 8s	5m 48s	2.30x	-5.12x
ibm11	15.60	15.11	15.37	1.01	0.98	14m 28s	1m 4s	4m 50s	2.99x	-4.53x
ibm12	25.41	24.06	23.85	1.07	1.01	13 m 55s	1m 15s	6m 20s	2.20x	-4.27x
ibm13	19.13	18.72	16.68	1.15	1.12	17m 54s	1m 30s	6m 17s	2.85x	-4.19x
ibm14	36.01	34.91	35.73	1.01	0.98	31m 34s	2m 27s	16m 01s	1.97x	-6.54x
ibm15	44.27	43.00	43.03	1.03	1.00	39m 44s	3m 57s	26m 37s	1.49x	-6.74x
ibm16	49.69	45.43	46.36	1.07	0.98	41m 26s	4m 03s	31m 33s	1.31x	-8.53x
ibm17	69.73	66.38	64.15	1.09	1.03	44m 50s	4m 21s	42m 01s	1.07x	-10.46x
ibm18	47.10	45.40	46.40	1.02	0.98	46m 48s	4m 44s	45m 21s	1.03x	-9.58x
Average				1.04	0.99				2.41x	-5.26x

7. Results and Discussion

Analyzing the tab. 1, we observe that the new spreading technique can achieve 4% better results than Capo cell placer taking, on average, lower execution time. However, it is clear that the PlaceDL will lose the advantage of time on the Capo when the size of the circuit increases, but still having better wirelength.

The PlaceDL shows results similar to FastPlace in wirelength, but is on average 5x slower.

8. Conclusions

This work presented the PlaceDL - a fast global placer, which can tackle thousand of cells in reasonable time and generate good results. The new technique of diffusion, in addition to generate good results, is simple to implement.

Despite our placer not be faster than FastPlace, it has showed, on average, equivalent results. Unlike the paper, in which is presented FastPlace, this paper entirely describes our method, which allows one to exact reproduce our results.

9. References

- [1] Viswanathan, N.; Chu, C C-N. FastPlace: Efficient Analytical Placement using Cell Shifting, Iterative Local Refinement and a Hybrid Net Model. *IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems*, Vol 24, Issue 5. May 2005.
- [2] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection alone produce routable placements? In *Design Automation Conference*, pages 477–482, 2000.
- [3] Alpert, C. J.; Chan, T.; Huang, D. J.-H.; Markov, I.; Yan, K. Quadratic Placement Revisited. In: *DAC '97: Proceedings*. ACM Press, 1997.
- [4] Shewchuk, J. An Introduction to Conjugate Gradient without the Agonizing Pain. Technical Report, School of Computer Science, Carnegie Mellon University, 1994.

EL-FI – On the Elmore “Fidelity” under Nanoscale Technologies

Tiago Reimann, Glauco Santos, Ricardo Reis
{tjreimann,gbvsantos}@inf.ufrgs.br

Instituto de Informática - Universidade Federal do Rio Grande do Sul

Abstract

In the middle of the nineties, it have been stated that, despite some issues in the Accuracy Property of the Elmore Delay Model [7], the Fidelity Property of it, makes this model capable of properly ranking routing solutions, thus, allowing an efficient comparison between different Interconnect Design approaches, methodologies or algorithms. In this ongoing work, we dive more deeply on the Fidelity Property of Elmore Delay, by analyzing it under nanoscale interconnect parameters, showing how does it behaves while scaling down to 13nm processes. Further, we notice a high standard deviation in the Fidelity analysis.

1. Introduction

The Elmore Delay Model (EDM) is well known as an analytical model for interconnect delay. Efficient and ease to use, it is also known the inaccurate estimates it provides in some spots of the target interconnect structure, specially in the parts close to the source (or driver). Despite of this inaccuracy that it presents sometimes (due to an overestimated downstream capacitance, that is actually diminished by the resistive shield), it have been verified that the EDM provides a considerable Fidelity when ranking routing solutions. That means, once different routing solutions are given to a given routing problem, it provides a similar, or even the same, rank as electrical simulation, thus, allowing to evaluate and compare different routing techniques, then picking the best one with a fine degree of certainty. This was named as the “**Fidelity**” Property of the EDM. Such experiments were done based on a standard rank-ordering technique used in the social sciences [1]. In [2] they used the rank-ordering technique of [1] to analyze the Fidelity of the ranks provided by EDM for the whole set of possible spanning trees of randomly generated point sets (or nets), in respect to SPICE simulation ranks. Based on the **comparison of the average of the absolute difference between both ranks**, they assumed that EDM provided high Fidelity delay criterion.

In section two we try to reproduce the experiments of [2]. In section three we define more realistic and up-to-date scenarios for the new experiments, based on nowadays technologies and different grid sizes according to the scope of routing (local/detailed, long/global, system-level). In section four we describe the methodology and the current state of the work, followed by the conclusions and future work.

2. Reproduction of the previous experiments

We used the rank-ordering technique described in [1] for reproducing the experiments of [2]. Though this technique was defined as inefficient for the qualitative nature of the data used in this social sciences book, it is perceptible the insight of the authors of [2] in applying it for their quantitative measurements. First, for reproducing the experiments, we had to consider the whole set of possible spanning trees for each one of the nets. They are two sets of 50 randomly generated nets for each one of the net sizes: 4 and 5. For nets with 4 terminals there are 16 possible solutions and for nets of 5 terminals there are 125 ($|M|^{|V|-2}$). Fig. 1 shows an example with all the 16 possible spanning trees for a net with four terminals in a square disposition.

The set of solution for each net is ranked **both** by Elmore and *hspice* **delay results**. Tab. 1 then shows the results with the **difference between both ranks**, given for the best case, five best cases, and the average differences for the whole 50 nets of each net size. We collect both the delay time to a given randomly-chosen critical sink in each net, and the worst delay for each net as well. Those results were obtained with the interconnect parameters (driver resistance, wire unit resistance, wire unit capacitance, and sink load capacitance) provided in [2], for three different processes: 2.0 μ m, 1.2 μ m, and 0.5 μ m. It is not clear in

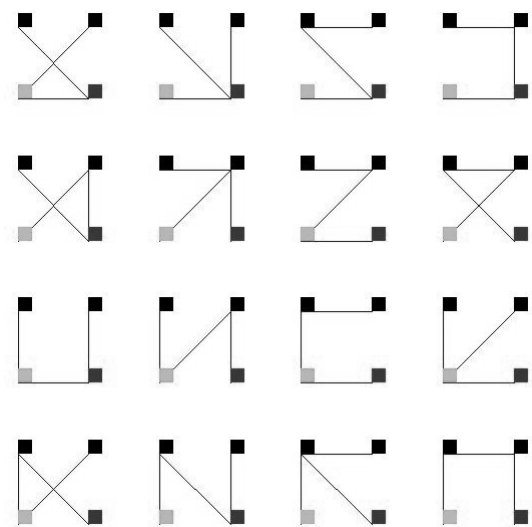


Fig. 1 – All the possible spanning trees for a four-terminals net.

Tab.1 – Average difference between Elmore and *hspice* delay rankings. Comparison of the results from [2] with our experiments on both a 1cm² and a 500x500µm grids. The rank position differences were achieved for a critical sink delay and for the worst sink delay in three cases: best case, 5 best cases, and the average of the routing solutions for each net in each net size.

CRITICAL SINK DELAY	Tech.	Case	Net size = 4			Net size = 5		
			From [2]	1 x 1 cm	500x500µm	From [2]	1 x 1 cm	500x500µm
	2.0µm	Best	0.54	0.00	0.00	5.90	0.09	0.05
		5 Best	1.02	0.21	0.10	7.20	0.38	0.15
		All	0.92	0.31	0.14	8.00	2.70	1.58
	1.2µm	Best	0.58	0.63	0.00	6.40	5.34	0.05
		5 Best	0.99	1.08	0.19	7.20	5.91	0.33
		All	0.94	0.88	0.30	7.90	7.90	2.73
	0.5µm	Best	0.58	0.58	0.58	5.60	5.81	6.08
		5 Best	0.93	1.08	1.07	6.50	6.01	6.42
		All	0.93	0.84	0.88	7.70	7.88	8.02
MAXIMUM DELAY	Tech.	Case	Net size = 4			Net size = 5		
			From [2]	1 x 1 cm	500x500µm	From [2]	1 x 1 cm	500x500µm
	2.0µm	Best	0.38	0.00	0.00	0.10	0.09	0.05
		5 Best	0.71	0.01	0.00	0.47	0.28	0.14
		All	0.65	0.09	0.10	1.39	1.97	1.19
	1.2µm	Best	0.16	0.10	0.05	0.20	0.16	0.09
		5 Best	0.51	0.20	0.05	0.53	0.89	0.26
		All	0.43	0.24	0.11	1.24	3.88	1.87
	0.5µm	Best	0.48	0.00	0.00	0.20	0.44	0.48
		5 Best	0.52	0.07	0.15	0.44	1.04	1.05
		All	0.60	0.15	0.20	1.22	3.98	3.99

[2] the area that was considered for generating the random point sets, except for saying that the random nets had "pin locations chosen from a uniform distribution over the routing area", though they provide the typical die size of 1cm² for these processes. So, once in doubt about the actual grid size we tried to enclose the actual value by using the total die size area, and a small size area of 500x500µm. This is the reason for three columns underneath each net size in Tab. 1, those are respectively, the original values from [2], the values achieved by reproducing their experiments with the large grid of 1cm², and, finally, the values achieved with the small grid. We can see that closer values are achieved with the 1cm² grid, suggesting that the original grid was closer to this one rather the 500x500µm one.

Further, in the work of [2] it is not very clear whether a given Fidelity value is much of few better than another one. On an attempt to justify that Elmore delay had a high Fidelity for the critical-sink delay criterion, (and nearly perfect Fidelity for the maximum sink delay criterion) they exemplified that, with the 5-pin nets and the 0.5µm technology, optimal critical-sink topologies under Elmore delay averaged only 5.6 rank positions (out of 125) away from optimal according to SPICE, while the best topology for maximum Elmore delay averaged only 0.2 positions away from its "proper" rank using SPICE. In our case, we can find the values 5.81 and 0.44, respectively for the same cases.

3. Definition of the Nanoscale Interconnect Scenarios

In order to observe how the Fidelity property of EDM behaves as feature sizes decrease, we are going to use technological RC parameters related to technologies ranging from 350nm down to 13nm from [3]. Those are based on the interconnects classification of the International Technology Roadmap for Semiconductors [4] providing different parameters for **short**, **intermediate** and **global** wires, according to the metal layers - fig.. 2.

Also, it became evident from the experiments of the previous section that the grid size provides significant impact on the results obtained. Thus, we are going to consider different grid sizes according to the type of routing (and related metal levels). Fig. 3 illustrates these area concepts for a 350nm technology. First we will use 200mm² as a reasonable SoC die area, based on an average of some die area values available at [5], comprising Intel and AMD processors produced in 90nm and 65nm cmos technologies. As observed in [6], the long global connections of the SoC designs will not scale in length as much as the local interconnects related to detailed routing. Therefore, we defined the same 200mm² "SoC grid" for all technologies, since this parameter do not present the same decreasing as smaller grids due to integration increase. We then define 14% of this area as a typical area for a random logic block (RLB) in a 350nm technology. Though it is not possible to specify what it would be the actual size of a RLB for any design under a given technology process, this is reasonable size for those who have some experience in backend. A fourth part of the RLB area was then defined as a fine upperbound for local networks inside a RLB, since those have their terminals approximated by the placement step of Physical Synthesis.

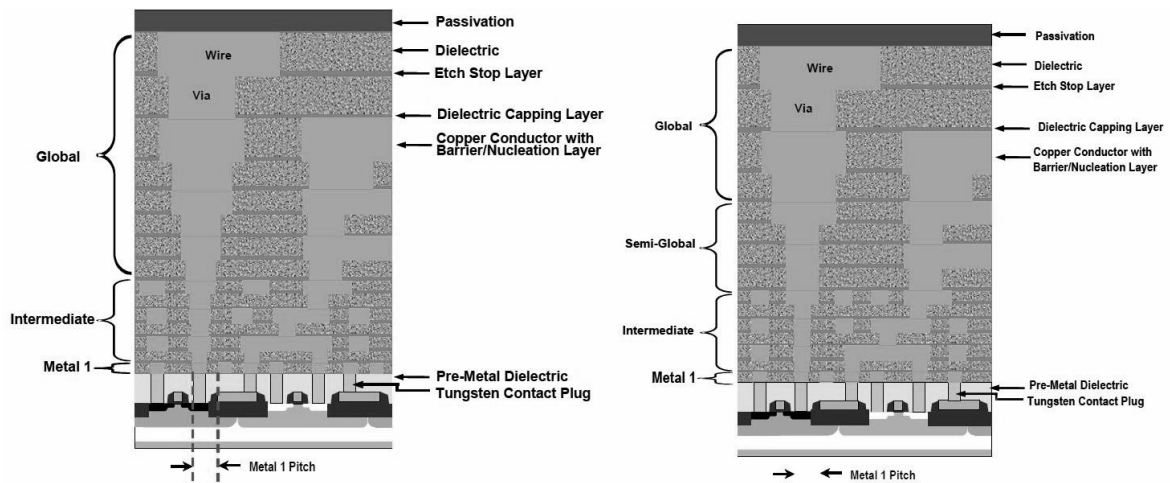


Fig. 2 - Cross-section of hierarchical scaling for ASIC (left) and MPU (right) devices from ITRS [4].

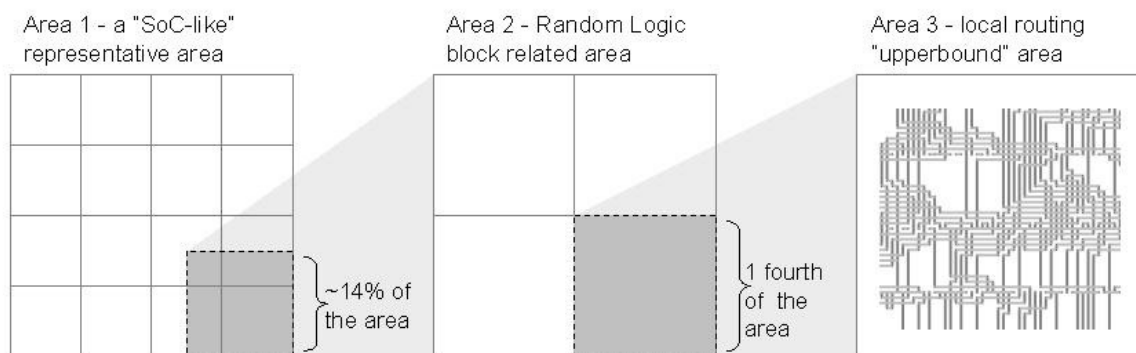


Fig. 3 – Definitions of three representative areas used for 350nm: SoC, RLB and local routing areas.

After that, assuming that the "SOC-like" representative area will be used for all technologies, we extrapolate the local routing grid to the smaller technologies in respect to the 350nm. The relation between the RLB grid and the other ones is also used for defining the other technologies' grid sizes, resulting in the grid sizes plotted in the chart of Fig 4.

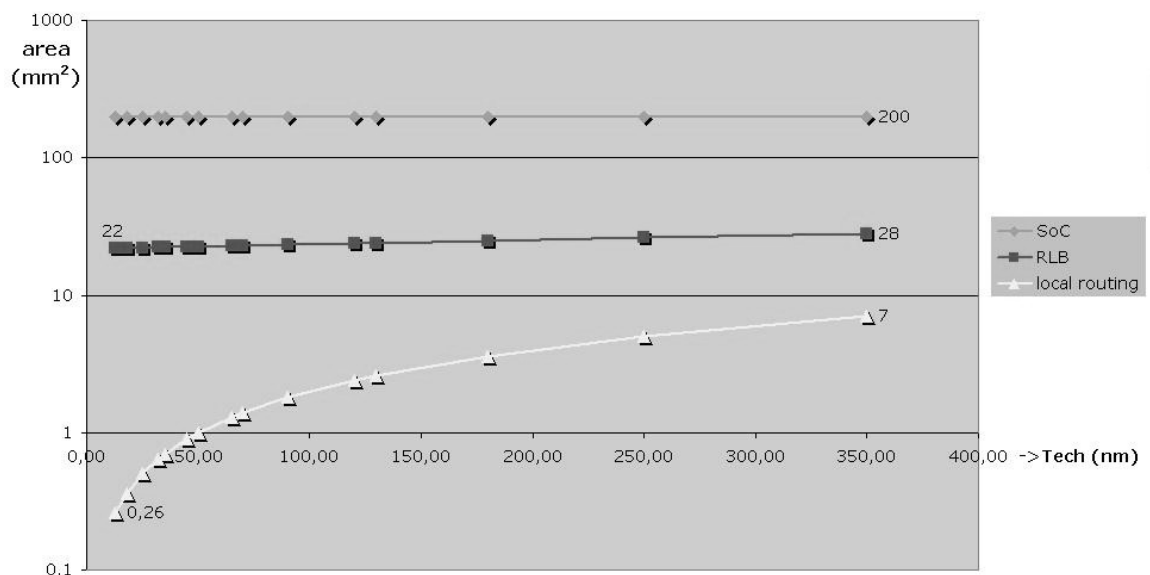


Fig. 4 – SoC, RLB and local routing grids definitions (in mm²) for the 15 technologies: 350nm, 250nm, 180nm, 130nm, 120nm, 90nm, 70nm, 65nm, 50nm, 45nm, 35nm, 32nm, 25nm, 18nm, 13nm.

4. Methodology and Current Status

We are now ready to perform the same experiments of section 2 with interconnect RC parameters corresponding to 15 technologies: 350nm, 250nm, 180nm, 130nm, 120nm, 90nm, 70nm, 65nm, 50nm, 45nm, 35nm, 32nm, 25nm, 18nm, 13nm. The experiments are currently in progress.

Besides providing the same results for nanometric scenarios, we are going also to collect the standard deviation information. We had already notice a considerable standard deviation, not yet measured, in the experiments that resulted in the tab. 1 of section two, so that we are not very sure if the average measurements ever provided so far, are enough to state the Fidelity property of EDM. We believe that these cases are due to some topologies in which large downstream capacitances impact significantly the second term of the Elmore Delay closed form expression. Such topologies will be usual in a design either to reduce cost (or wirelength) diminishing power consumption or liberate routing resources.

5. Conclusion and Future Work

In this ongoing work we address the Fidelity Property of the Elmore Delay Model, by analyzing it in nowadays scenarios. The methodology is well defined, we had already reproduced the first experiments from [2], what called us attention to a significant standard deviation, that was not considered in the former experiments and is going to be measured now.

6. References

- [1] T. G. Andrews, *Methods of Psychology*. New York: John Wiley & Sons, 1949. pp. 146.
- [2] K.D. Boese, A.B. Kahng, B.A. McCoy, G. Robins, Near-optimal critical sink routing tree constructions, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Dec 1995. Vol. 14, Issue: 12, pp. 1417-1436.
- [3] G. B. V. Santos. A Collection of Resistance and Capacitance Parameters' Sets towards RC Delay Modeling of Nanoscale Signal Networks. Technical Report/Trabalho Individual, Porto Alegre, Universidade Federal do Rio Grande do Sul, 2008.
- [4] International Technology Roadmap for Semiconductors. <http://www.itrs.net/>
- [5] Patrick Schmid, "Game Over? Core 2 Duo Knocks Out Athlon 64 : Core 2 Duo Is The New King" , July, 2006. <http://www.tomshardware.com/reviews/core2-duo-knocks-athlon-64,1282.html>.
- [6] Ho, R.; Mai, K. W.; Horowitz, M. A. The Future of Wires. *Proceedings of the IEEE*, VOL. 89, NO. 4, April, 2001.
- [7] W. C. Elmore, "The Transient Response Of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, 1948.

Automatic Synthesis of Analog Integrated Circuits Using Genetic Algorithms and Electrical Simulations

Lucas Compassi Severo, Alessandro Girardi

lucascs.eletrica@gmail.com, alessandro.girardi@unipampa.edu.br

Federal University of Pampa – UNIPAMPA
Campus Alegrete

Abstract

The goal of this paper is to present a tool for automatic synthesis of analog basic integrated blocks using the genetic algorithm heuristic and external electrical simulator. The methodology is based on the minimization of a cost function and a set of constraints in order to size circuit individual transistors. The synthesis methodology was implemented in Matlab and used GAOT (Genetic Algorithm Optimization Toolbox) as heuristic. For transistors simulation we used the Smash simulator and ACM MOSFET model, which includes a reduced set of parameters and is continuous in all regions of operation, providing an efficient search in the design space. As circuit design example, this paper shows the application of this methodology for the design of an active load differential amplifier in AMS 0.35um technology.

1. Introduction

The design automation of analog integrated circuits can be very useful in microelectronics, because it provides an efficient search for the circuit variables, among a set of design constraints, to make it more efficient as possible. Several works have been done in this theme, aiming the development of tools for the automation of time-consuming tasks and complex searches in highly non-linear design spaces [1, 2]. However, as far as we know, there is not a commercial tool capable to perform the synthesis of analog circuits with optimum results in a feasible time. An important improvement in the analog design could be the automation of some design stages, such as transistor sizing and layout generation [3], maintaining the interaction with the human designer. The large number of design variables and the consequent large design space turn this task extremely difficult to perform even for most advanced computational systems. Therefore, it is mandatory the use of artificial intelligence with great computational power to solve these problems.

In this context, we propose an automatic synthesis procedure for basic analog building blocks which can size transistors width (W) and length (L) with efficient time and ordinary computational resources. The synthesis procedure has as main strategy the global search using genetic algorithm and the evaluation of circuit characteristics through the interaction with an electric simulator. The Genetic Algorithm (GA) is a technical idealized in 1975 by scientist John Holland inspired by principles of natural evolution proposed by Charles Darwin. This evolutionary heuristic is very used for automatic design of integrated circuits [4, 5, 6].

This work is organized as follows: section 2 shows the description of the proposed methodology; section 3 presents the application of the methodology in the design of a specific analog block - the differential amplifier - with circuit description and final results; finally, section 4 shows the conclusion.

2. Automatic Synthesis

The methodology is based on the reduction of a cost function for a specific analog block. This cost function is dependent of the circuit electrical characteristic and is implemented as an equation in terms of circuit variables. The electrical characteristics can be power consumption, area, voltage gain, etc, or a combination of these. In this work we used the circuit power dissipation as cost function. The optimization is based on the reduction of a cost function and a set of constraints. This reduction is performed by genetic algorithm with values provided by electrical simulations through an external electrical simulator.

The genetic algorithm is a heuristic for non-linear optimization based on the analogy with biologic evolution theories [7]. It is a non-deterministic algorithm and it works with a variety of solutions (population), simultaneously. The population is a set of possible solutions for the problem. The size of the population is defined in order to maintain an acceptable diversity considering an efficient optimization time. Each possible solution of population is denominate chromosome, where chromosome is a chain of characters (gens) that represent the circuit variables. This representation can be in binary number, float or others. Fig.1 shows an example of binary chromosome. The quality of the solution is defined by an evaluation function (cost function). Fig.2 shows the automated design flow using genetic algorithm. The algorithm receives an initial population, created randomly, some recombination and mutation operators and the MOSFET technology model parameters.

The population is evaluated using an external commercial electrical simulator. Based on valuation and roulette method [8] the parent chromosomes are selected for generating new chromosomes. The new chromosomes are created including recombination and mutation - analogy with biology. In the recombination, the chromosomes parents are parties and the union of parts of parents make the recombination. The mutation is a random error that happens in a chromosome. The probability of mutation is defined and is compared with a random value. If this random value is less than the probability then a gene on chromosome is randomly changed. The next step is the exclusion of parents, evaluation of new chromosomes, using again the electrical simulator and a cost function. Based on these values, new chromosomes are introduced in the population. At the end of each iteration, the stopping condition is tested and, if true, then the optimization is finished. Otherwise, new parents are selected and the process is repeated. The stopping condition can be the number of generations (iterations), minimal variation between variables or cost function, or others.

The synthesis tool developed in this work was implemented in Matlab® and used the Dolphin Smash® simulator as external electrical simulator. The MOSFET model used was ACM [9], guaranteeing the correct exploration of all transistor operation regions. For the execution of genetic algorithm, we adopted GAOT (Genetic Algorithm Optimization Toolbox), an implementation of GA for Matlab developed by Cristopher R. Houck et al [10].

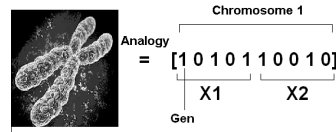


Fig. 1 – Example of a typical binary chromosome for a two variables problem.

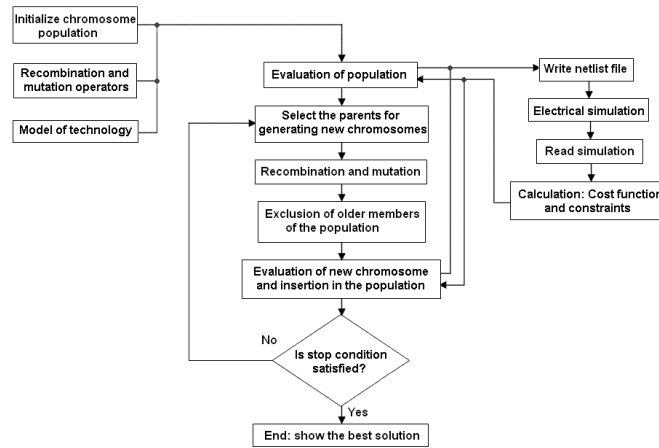


Fig. 2 – Design flow for the analog synthesis using genetic algorithms.

3. Design Example – Differential Amplifier

As an application for the proposed methodology, we implemented a tool for the automatic synthesis of a CMOS differential amplifier. The differential amplifier is one of the most versatile circuits in analog design. It is compatible with ordinary CMOS integrated-circuit technology and serves as input stage for op amps [11]. Its basic function is to amplify the difference between the input voltages. The circuit for a differential amplifier with active load is basically composed by a load current mirror (M3 and M4), a source-coupled differential pair (M1 and M2) and a reference current mirror (M5 and M6), shown in fig.3. The main electrical parameters of the circuit are low-frequency voltage gain (A_{v0}), gain-bandwidth product (GBW), slew-rate (SR), input common-mode range (ICMR), dissipated power (P_{diss}) and area (A), among others.

The low-frequency gain is the relationship between output and input voltages, defined as:

$$A_{v0} = \frac{g_{m1}}{g_{ds2} + g_{ds4}} \quad (1)$$

where g_{m1} is the gate transconductance of transistor M1 and g_{ds2} and g_{ds4} are the output conductance of M2 and M4, respectively.

The slew rate (SR) is the maximum output-voltage rate, either positive or negative, given by:

$$SR = \frac{I_{ref}}{C_1} \quad (2)$$

Here, I_{ref} is the current source of circuit C_1 is the total output capacitance. This capacitance is estimated as the sum of the load capacitance CL and drain capacitance of M_2 and M_4 . Input common-mode range (ICMR) is the maximum and minimum input common-mode voltage, defined as:

$$ICMR^- = v_{DS5(sat)} + v_{GS1} + v_{SS} \quad (3)$$

$$ICMR^+ = v_{DD} - v_{GS3} + v_{TN1} \quad (4)$$

In this case, $v_{DS5(sat)}$ is the saturation voltage of transistor M_5 , v_{GS1} and v_{GS3} are gate-source voltages of M_1 and M_3 , respectively, v_{DD} and v_{SS} are voltage sources of circuit and v_{TN1} is the threshold voltage of M_1 . The v_{DD} and v_{SS} source voltages are defined in this example as -1.65V and 1.65V, respectively.

The gain-bandwidth product is given by:

$$GBW = \frac{g_{m1}}{C_1} \quad (5)$$

The cost function for the circuit in this case is related to the power dissipation, defined as:

$$f = \frac{(v_{DD} + |v_{SS}|) \cdot I_{ref}}{P_0} + R \quad (6)$$

where, R is a penalty constraint function, which will be a large value if the constraints are not met, and zero if all constraints are met. P_0 is the reference power dissipation for normalization purpose.

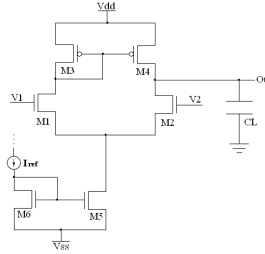


Fig. 3 – Schematics of a differential amplifier

The optimization procedure was implemented in Matlab, using the GAOT Toolbox as described before. We analyzed three scenarios with different population (10, 100 and 1000 individuals), in order to verify which one is more suitable for this type of problem. Tab.1 shows the values of constraints, optimization time and the achieved final cost function (optimized power dissipation). Analyzing this table it is possible to notice that the best value of individual numbers, between the analyzed, is 1000 individuals, because this population presented a major minimization in the power dissipation. Tab.2 shows the final values of variables and tab.3 shows a comparison between achieved and imposed restrictions. Fig.4 shows the variation of cost function and the slew rate (SR) along the iterations.

The proposed methodology was initialized with a random generated population. After more than 2000 iterations, final population satisfies all constraints and provided optimized power dissipation. This is a valuable characteristic of design automation using genetic algorithms, because a good guess for initial values is not mandatory.

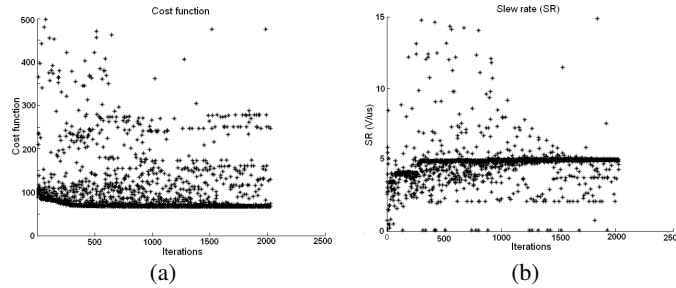


Fig. 4 – Evolution of algorithm response along with iterations: a) cost function; b) slew rate variation.

Tab.1 – Optimization results for different population sizes

Individuals	GBW	Slew Rate	Voltage Gain	ICMR-	ICMR+	Power dissipation	Time	Generations
10	3.91MHz	5.12V/μs	61.18 dB	-1.02V	1.07V	148.33μW	22min	2524
100	949kHz	5.00 V/μs	62.29dB	-0.94V	0.81V	148.80μW	19min	2354
1000	5.95MHz	5.00 V/μs	60dB	-0.70V	1.03V	139.46μW	25min	2026

Tab.2 – Circuit variables

Variable	Initial value	Optimized value
$W(M1 \text{ e } M2)$	random	99.98 μm
$W(M3 \text{ e } M4)$	random	24.17 μm
$W(M5 \text{ e } M6)$	random	67.49 μm
$L(M1 \text{ e } M2)$	random	1.85 μm
$L(M3 \text{ e } M4)$	random	2.56 μm
$L(M5 \text{ e } M6)$	random	23.96 μm
I_{ref}	random	51.22 μA

Tab.3 – Circuit constraints

Restriction	Required	Reached
Gain (A_v0)	60.00dB	60.00dB
GBW	1.00MHz	5.95MHz
SR	5.00V/us	5.00V/us
ICRM-	-0.70V	-0.70V
ICMR+	0.70V	1.03V
Gate area	-	3727.79 μm^2
Power diss.	minimize	139.46 μW

4. Conclusion

The proposed methodology for the synthesis of basic analog building blocks presented good results in a reasonable computing time. Genetic algorithms are very suitable for analog design automation because the convergence of the final solution is not directly dependent of initial solution, and it is not necessary a great knowledge by the human designer about the circuit characteristics. However, it is very important to determine the size of population (number of individuals) because it is directly related to the quality and the time of optimization. In our work, a population of 1000 individuals provides the best result.

Electrical simulator using the ACM model implemented in this methodology guarantee the search in all regions of operation of MOSFET transistors.

As future work, we intend to compare the solution obtained with GAs with other optimization heuristics. Also, we can explore the use of electrical simulators from different vendors, expand the methodology for other analog basics blocks and create an interface for the human designer.

5. Acknowledgments

The grant provided by FAPERGS research agency for supporting this work is gratefully acknowledged.

6. References

- [1] M. Degrauwe, O. Nys, E. Dukstra, J. Rijmenants, S. Bitz, B. L. A. G. Go@art, E. A. Vittoz, S. Cserveny, C. Meixenberger, G. V. der Stappen, and H. J. Oguey, "IDAC: An interactive design tool for analog CMOS Circuits", IEEE Journal of Solid-State Circuits, SC-22(6):1106{1116, December 1987
- [2] M. D. M. Hershenson, S. P. Boyd, and T. H. Lee, "Optimal design of a CMOS op-amp via geometric Programming", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 20(1):1{21, January 2001.
- [3] A. Girardi and S. Bampi, "LIT - An Automatic Layout Generation Tool for Trapezoidal Association of Transistors for Basic Analog Building Blocks", Design automation and test in Europe, 2003.
- [4] R. S. Zebulum, M.A.C. Pacheco, M.M.B.R., Vellasco, "Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms", USA: CRC, 2002.
- [5] Rego, T. T., Gimenez, S. P., Thomaz, C. E., "Analog Integrated Circuits Design by means of Genetic Algorithms". In: 8th Students Forum on Microelectronic SForum'08, 2008, Gramado - RS, Brazil.
- [6] Taherzadeh-Sani, M. Lotfi, R. Zare-Hoseini, H. Shoaie, O., "Design optimization of analog integrated circuits using simulation-based genetic algorithm", Signals, Circuits and Systems, 2003, International Symposium on, Iasi, Romania.
- [7] P. Venkataraman, "Applied Optimization with Matlab Programming", John Wiley e sons, New York, 2002.
- [8] Linden, Ricardo, "Algoritmos Genéticos – Uma importante ferramenta da inteligência artificial", Brasport, Rio de Janeiro, 2006.
- [9] A. I. A. Cunha, M. C. Schneider, and C. Galup-Montoro, "An MOS transistor model for analog circuit design". IEEE Journal of Solid-State Circuits, 33(10):1510{1519, October 1998.
- [10] Christopher R. Houck, Jeffery A. Joine and Michael G. Kay, "A Genetic Algorithm for Function Optimization: A Matlab Implementation", North Carolina State University, available at <http://www.ie.ncsu.edu/mirage/GAToolB>
- [11] P. E. Allen and D. R. Holberg, "CMOS Analog Circuit Design", Oxford University Press, Oxford, second Edition, 2002.

PicoBlaze C: a Compiler for PicoBlaze Microcontroller Core

Caroline Farias Salvador¹, André Raabe², Cesar Albenes Zeferino¹
{caroline, raabe, zeferino}@univali.br

¹Grupo de Sistemas Embarcados e Distribuídos

²Ciência da Computação

Universidade do Vale do Itajaí

Abstract

PicoBlaze microcontroller is a soft-core provided by Xilinx with a set of tools for applications development. Despite the range of tools available, developers still depend on assembly programming, which makes hard the development and extends the design time. This work presents the design of an integrated development environment (IDE) including a C compiler and a graphical simulator for the PicoBlaze microcontroller. It is being developed in order to provide a tool for programming PicoBlaze at high-level, making easier the development of new applications based on this microcontroller.

1. Introduction

The PicoBlaze™ microcontroller [1] is a compact 8-bit RISC microcontroller core developed by Xilinx to be used with devices of the Spartan®-3, Virtex®-II, and Virtex-II ProFPGA families. It is optimized for low deployment cost and occupies just 96 slices (i.e. 12.5%) of an XC3S50 FPGA and only 0.3% of an XC3S5000 FPGA [1].

PicoBlaze is available by Xilinx as a free VHDL core, supported by a set of developing tools from Xilinx and by a third-part company. The Xilinx KCPSM3 [1] – *Constant (K) Coded Programmable State Machine* – is a command-line assembler which generates the files for synthesis of a PicoBlaze application. The Mediatronix pBlazIDE software provides a graphical interface and includes an assembler and an instruction-set simulator. Besides these tools, an unofficial compiler, called PCCOMP (PicoBlaze C Compiler) [2] can be used, but it presents several limitations, as discussed later, which motivated the development of a new compiler.

This work intends to build an IDE (Integrated Development Environment) with a C compiler and a graphical simulator for the PicoBlaze microcontroller, providing developers with a tool for programming PicoBlaze at high-level. The proposed compiler will perform the steps of lexical, syntactic and semantic analysis, and will generate an assembly file to be assembled by using KCPSM3. The IDE will also include an instruction set-simulator and a graphical interface to make easier the testing of the developed applications.

This paper presents issues related with the design of the proposed compiler, which is currently under development. The text is organized in five sections. Section 2 presents the features of PicoBlaze microcontroller and its design flow. Section 3 presents the analysis of PCCOMP, the only C compiler available for the development of PicoBlaze applications. Section 4 presents issues related with the proposed IDE design and describes the methodology to be used to validate its implementation. Section 5 presents the conclusions.

2. PicoBlaze microcontroller

A microcontroller can be defined as a "small" electronic component, with an "intelligence" program, used to control peripherals such as LEDs, buttons and LCD (Liquid Crystal Display), and basically includes a processor, embedded memories (RAM – Random Access Memory and ROM – Read-Only Memory) and Input/Output peripherals integrated on a single chip [3].

The core of PicoBlaze is fully embedded in an FPGA and does not need external resources, making it extremely flexible. Its basic features can be easily expanded due to the large number of input and output pins.

The PicoBlaze microcontroller CPU is a RISC (Reduced Instruction Set Computer) 8-bit processor. It includes 16 8-bit registers, an 8-bit ALU (Arithmetic Logic Unit), a hardware stack CALL / RETURN and a PC (Program Counter). The microcontroller also includes a scratchpad RAM, a ROM and input and output pins. The diagram block of this microcontroller is shown in Fig. 1.

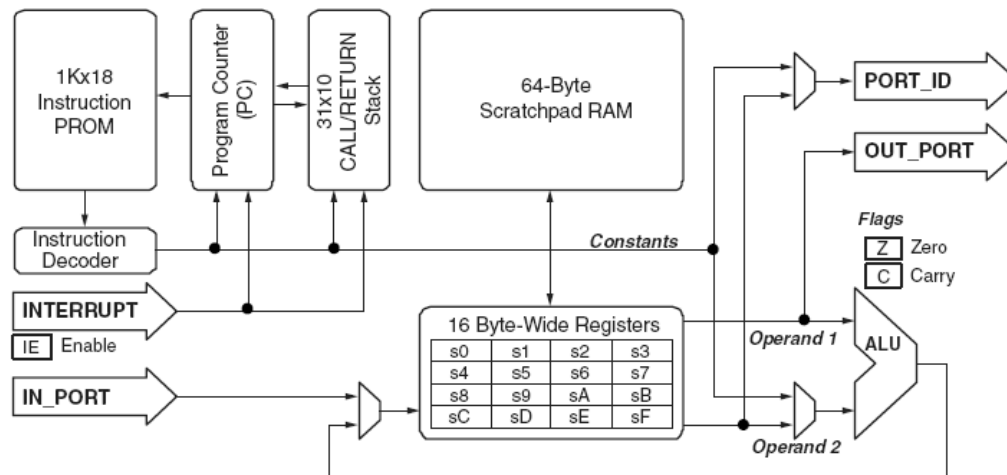


Fig. 1 – Functional blocks of the PicoBlaze microcontroller

As previously discussed the design flow for applications development based on PicoBlaze is supported by the Mediatronix pBlazIDE and by the Xilinx KCPSM3 assembler. The design flow based on this tool is depicted in Fig. 2.

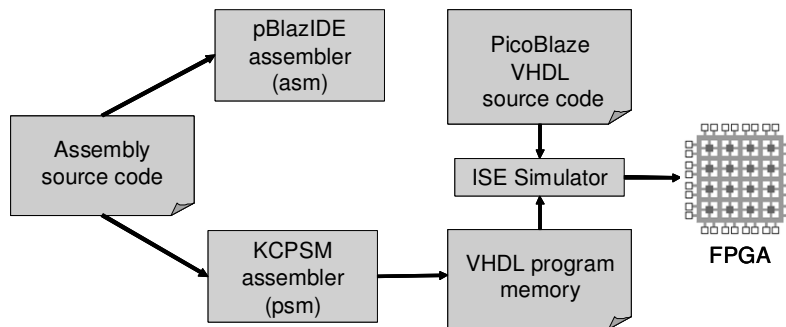


Fig. 2 – Design flow of PicoBlaze

In the design flow of PicoBlaze, a developer can use the KCPSM3 assembler or the assembler of the pBlazIDE. The advantage of the pBlazIDE relies in its support for debugging based on a graphical interface, which allows step-by-step debugging, showing current state of register bank, data memory, flags, interrupts and input and output pins, which are very useful in the code development phase. However, to generate the VHDL model for the program memory, the designer has to use the KCPSM3 tool.

After generate the VHDL model of the program memory for the target application, the designer has to use the Xilinx ISE tools to simulate the PicoBlaze running the application tool. Also, ISE must be used to synthesize the microcontroller into the FPGA.

3. Related works

The only compiler available to develop applications based on PicoBlaze is PCCOMP (PicoBlaze C Compiler) [2]. It is command-line C compiler which generates assembly code with the extension “.PSM”, which is used as input source file by the Xilinx KCPSM3 assembler. The PCCOMP was developed by Francesco Poderico and is compatible with the Windows XP operating system. This work was motivated by some troubles found in the use of PCCOMP as compiler in the development of PicoBlaze applications. A number of limitations was found, which are described below.

Although PCCOMP includes a user manual, this document does not cover all the features related with PCCOMP usage, and some of them are discovered only by using the compiler. Problems such as declaration of variables and syntax of the language were some of the features found only in the stage of analysis.

Another difficulty is the low quality of the compiler error messages, because the compilation is done through command line and it is not possible to check the error message and automatically locate the piece of code related with the error.

Limitations were also found in the debugging of the assembly code generated by PCCOMP. One can verify that the generated code works correctly by using the pBlazIDE tool. However, to do that, it is necessary to

import the code generated by PCCOMP and make several changes in the assembly code, following instructions presented in PicoBlaze user Guide [1].

3.1. Analysis of PCCOMP limitations by using testbench codes

In order to analyze PCCOMP more accurately and to create a parameter for comparison with the compiler under development there were used a testbench defined by the Dalton Project [4]. The Dalton Project specifies a set of programs described in C++ which have been used to validate the synthesizable models of 8051 microcontroller and other processors.

The testbench codes were first translated by hand to PicoBlaze assembly and validated by using the pBlazeIDE assembler and then were compared with assembly code generated by PCCOMP.

Some changes in the testbench codes were necessary in order to adapt them to PCCOMP, meeting the requirements of the supported syntax. Also, in some tests, the program did not need to be changed but the assembly code generated by PCCOMP did not run correctly. Problems related to the declaration of local variables, initialization of variables, data types, operators and arithmetic expressions were also found.

Moreover, when considering the number of assembly instructions generated by the compiler (i.e. the program size), it was verified that the code generated by PCCOMP is much bigger than the one generated by hand, as shown in Tab. 1.

Tab.1 – Comparison of the number of instructions in assembly codes generated by hand and by using PCCOMP

Program	Number of assembly instructions in the code generated by hand	Number of assembly instructions in the code generated by PCCOMP
fib.c	37	207
sqroot.c	73	198
divmul.c	54	231
negcnt.c	11	40
sort.c	83	283
xram.c	14	82
cast.c	25	175
gcd.c	23	79
int2bin.c	33	87

It's known that hand coded assembly is usually smaller than compiler generated ones, but it is believed that this difference presented in Tab. 1 can be significantly reduced by a new compiler.

4. Design of the proposed IDE

To develop the proposed IDE, a complete design was performed, including the analysis of requirements, the building of use cases diagrams and interface prototyping, and the description of the grammar to be used to implement the compiler. The major requirements specified for the IDE (compiler + debugger + interface) to be developed include:

- The system will allow the C language compilation and the step-by-step debugging of applications;
- The system will indicate the errors found in applications during the compilation;
- The system will allow the developer to configure processor flags by using the graphical interface;
- The system will allow the user to enable and disable processor interrupt by using the graphical interface;
- The system will allow the user to visualize the current status of the registers, of the hardware stack, of the data memory and of the input and output pins during debugging by using the graphical interface;
- The system will allow the inclusion of pieces of assembly code inside the C code;
- The system will allow the user to develop headers in assembly language;
- The system will allow the user to view the generated assembly code; and
- The system will support data arrays and procedure calls.

Fig. 3 depicts the prototype of the interface for the system to be implemented, according with the requirements shown before. It has shortcuts to deal with file operations (eg. open, new, save) and also to compile and debug. At the left side, there is a number of boxes to show the status of registers, I/O, stack and data memory. At the right side is the box for the code edition, and, at the bottom, is the message box.

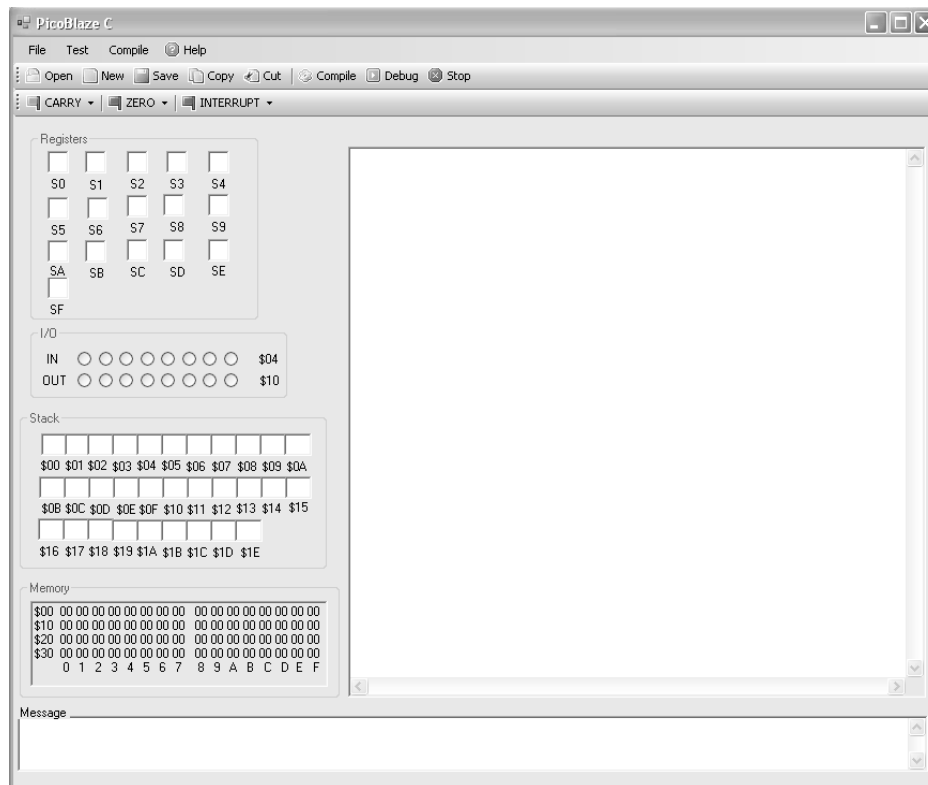


Fig. 3 – Main screen of the system.

The grammar defined for the compiler supports the include directive, headers, functions prototypes, procedures, conditional structures (if, elseif, else and switch), loop structures (for, while and do while), assignments, and arithmetic, relational and logical operations. The compiler development is being conducted using traditional compiler generator tools.

The IDE is currently under development. It will be validated by compiling the same testbench applications used to evaluate PCCOMP. The generated assembly files will be used as input files to the pBlazeIDE assembler, which will be used to check if the generated assembly is correct. After this, the number of generated instructions will be analyzed and compared with PCCOMP. The assembly code will also be used as file entry for the KCPSM3 assembler, which will generate the VHDL model for the memories. After that, PicoBlaze will be synthesized into an FPGA in order to perform the physical validation of the generated code.

5. Conclusion

In this paper it was presented the ongoing development of an IDE with a C compiler and a graphical simulator for the PicoBlaze microcontroller. At this first stage of the work, it was performed the analysis of similar works and the design of the proposed IDE, which is under implementation phase.

6. References

- [1] Xilinx, "PicoBlaze 8-bit Embedded Microcontroller: User Guide", June 2008.
- [2] F. Poderico, "Picoblaze C compiler: user's manual 1.1", July 2005.
- [3] SOUZA, David J. "Desbravando o PIC," 6nd ed., Ed. Érica: São Paulo, 2003.
- [4] The UCR Dalton Project, 2001. Available at: <http://www.cs.ucr.edu/~dalton/>



Digital Design

Equivalent Circuit for NBTI Evaluation in CMOS Logic Gates

Nivea Schuch, Vinicius Dal Bem, André Reis, Renato Ribas
{nschuch, vdbem, andreis, rpribas}@inf.ufrgs.br

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre,
Rio Grande do Sul, Brazil

Abstract

As technology scales, effects such as NBTI increase their importance. The performance degradation in CMOS circuits caused by NBTI is being studied for many years and several models are available. In this work, an equivalent circuit representing one of these models is presented and evaluated through electrical simulation. The proposed circuit is process independent once the chosen model does not present any relation to technology parameters. Experimental results indicate that the equivalent circuit is valid to evaluate the degradation of PMOS transistors due to aging, and each transistor can be simulated individually, allowing so the evaluation of complex CMOS gates in a linear cost in relation to the number of transistors.

1. Introduction

As semiconductor manufacturing migrates to more advanced deep-submicron technologies, accelerated aging effect becomes one major limiting factor in circuit design. This is a challenge for designers to effectively mitigate degradation and improve the system lifetime. One of the most important phenomenon that causes temporal reliability degradation in MOSFETs is the Negative Bias Temperature Instability (NBTI) [1].

NBTI is an effect that leads to the generation of interface Si/SiO₂ traps, causing consequently a shift in the PMOS transistor threshold voltage (V_{th}), being dependent on the operating temperature and exposure time. This effect induces a performance degradation, increasing the signal propagation delay in the circuit paths. Besides, it degrades the drive current and the noise margin. Although it is known since a long time [2], only in nano-scale designs it has been identified as a critical reliability issue [3] due to the increase of both gate electrical fields and chip operation temperature, combined with the power supply voltage reduction.

Recently, much effort has been expended to further the basic understanding of this mechanism [4]. In order to estimate the threshold voltage degradation caused by NBTI, different analytical models have been proposed in literature [3], [5], [6]-[10]. Moreover, some of these models are somewhat difficult to be used once specific and empirical process parameters are required.

In order to have an estimate about circuit failure probability, electrical simulation may be performed applying one of those models represented by an equivalent electrical circuit. In this paper, it is demonstrated the use of the R-D model, proposed in [5], which describes the dependence between physical and environmental factors. The main goal is to validate the equivalent circuit proposed to represent the NBTI R-D model. It is used to evaluate the threshold voltage degradation of each transistor independently, demonstrating its usefulness for CMOS complex gates evaluation in a linear cost in relation to the number of transistors.

2. Modeling

Although the first experiments on NBTI reports to late 60's, only in late 70's a comprehensive study of available experiments was performed. Jeppson [11] presents a generalized Reaction-Diffusion model, being the first one to discuss the role of relaxation and bulk traps.

As NBTI was not an issue on NMOS technology at that time, no much research on this field was performed until early 90's. Increased electrical field and temperature due to the technology scaling reintroduces NBTI concerns for both analog and digital circuits. Several works proposed different models based on the Reaction-Diffusion model presented by Jeppson [3], [5], [6]-[10]. Those models are usually empirical and related to specific technologies, therefore they are quite difficult to implement for circuit simulation and particularly hard to be compared among themselves and with others.

A general and accurate analytical model was presented by Vattikonda et al. [5]. It is based on the physical understanding and published experimental data for both DC and AC operations. This model can be appropriately represented by an equivalent electrical circuit, allowing us to perform Spice simulations analysis.

It is strongly believed that when a gate voltage is applied, it initiates a field-dependent reaction at the Si/SiO₂ interface, that generates interface traps (Nit) by breaking the passivated Si-H bonds [3],[6],[9],[10]. These traps appear because of the positive holes from the channel that cause the diffusion away of the H, provoking the increase of V_{th} . The dependence between V_{th} and Nit can be noted by:

$$\Delta V_{th} = (qN_{it})/C_{ox} \quad (1)$$

where C_{ox} is the oxide capacitance per unit area.

During the operation of PMOS device, there are two different phases of the NBTI effect, depending on the bias condition: stress and recovery. During the stress phase, when $V_{SG}=V_{DD}$, positive interface traps are accumulating and H is diffused away. In the recovery phase, when the $V_{GS}=0V$, H is diffused back and anneals the broken Si-H, recovering the NBTI degradation [5]. The stress phase is also known as static phase.

The NBTI effect can be modeled as ‘static’ or ‘dynamic’ operation. The static one presents only the stress phase because it is required that the gate stays only negative biased during all the time. But, in actual circuit operations, where gate voltage switches between 0V and V_{DD} , both stress and recovery phases occur, denoting the dynamic operation.

For the direct calculation of V_{th} variation under NBTI, the Vattikonda’s model has been adopted in this work [5]. The formulation for stress phase is:

$$\Delta V_{th} = \sqrt{K_v^2 \cdot (t - t_0)^{1/2} + \Delta V_{th0}^2} + \delta_v \quad (2)$$

being

$$K_v = A t_{ox} \sqrt{C_{ox} (V_{GS} - V_{th})} [1 - V_{DS} / \alpha (V_{GS} - V_{th})] \exp(E_{ox} / E_0) \exp(-E_a / kT) \quad (3)$$

For recovery phase, the following equation is used:

$$\Delta V_{th} = (\Delta V_{th0} - \delta_v) [1 - \sqrt{(\tau(t - t_0)) / t}] \quad (4)$$

These models are scalable with target process and design parameters, such as gate oxide thickness (t_{ox}), gate-source (V_{GS}) and drain-source (V_{DS}) transistor voltages, device threshold voltage (V_{th}), temperature (T) and time (t). In tab. 1, the default values of Vattikonda’s model coefficients are given for 45 nm PTM process [12].

Tab.1 - Default values of Vattikonda’s model coefficients [5], considering PTM process [12].

$A = 1.8 \text{ mV/nm/C}^{0.5}$	$\eta = 0.35$	$E_0 = 2.0 \text{ Mv/cm}$
$\alpha = 1.3$	$\delta_v = 5.0 \text{ mV}$	$E_a = 0.13 \text{ eV}$

In fact, the stress and recovery processes are somewhat more complex. They may involve oxide traps and other charged residues [9], [13]–[15]. These non-H based mechanisms may have faster response time than the diffusion process. Without losing generality, their impact can be included as a constant of δ_v [5].

Mathematical simulation results of V_{th} degradation using the formulas presented above are shown in fig. 1 for both (a) static and (b) dynamic NBTI.

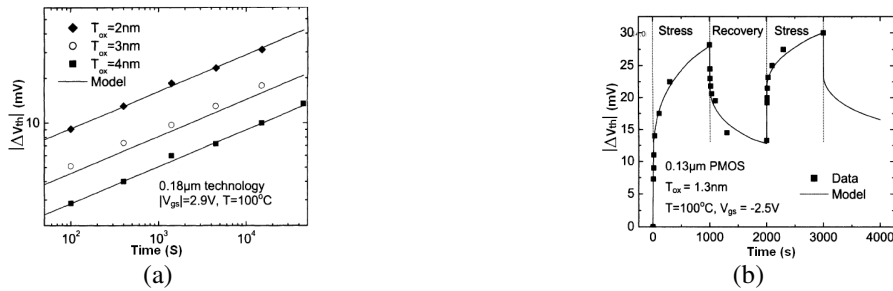


Fig. 1 – Results of the simulation of static (a) and dynamic (b) NBTI presented in [5].

3. Equivalent Circuit

In circuits real life usage, each transistor is excited in different moments, for a different amounts of time. Therefore, each transistor will suffer distinct aging speed. This work presents the development of an equivalent electrical circuit representation to the analytical model proposed by Vattikonda *et al.* [5], and its application in electrical simulation evaluating the aging effect of every PMOS transistor on the circuit. For each PMOS transistor, the NBTI effect is calculated and its degradation evaluated.

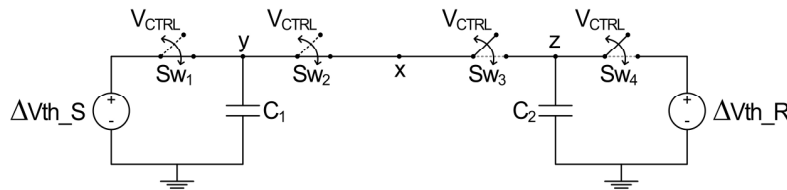


Fig. 2 – Equivalent electrical circuit for Vattikonda’s NBTI model.

In order to implement the threshold voltage variation (ΔV_{th}) presented in Equations (2) and (5), it is proposed the circuit shown in fig. 2. As every transistor is evaluated individually, it is not possible to apply the degradation calculation as a parameter to the used model card (global process parameters for all devices under

simulation). Thus, individual device threshold voltages variation are controlled by modifying individual source-bulk transistor voltage (V_{SB}), according to [16]:

$$V_{th} = V_{th0} + \gamma(\sqrt{\Phi_s + V_{SB}} - \sqrt{\Phi_s}) \quad (5)$$

The V_{BS} of every transistor is represented as a dependent source of the potential at node 'x' in fig. 2. The voltage sources, ΔV_{th_S} and ΔV_{th_R} , represent values dependent of the variables that are present on each NBTI phase. The values ΔV_{th_S} and ΔV_{th_R} are calculated according to the stress and the recovery Equations (2) and (4), respectively. The resultant equations implemented in these sources are:

$$\Delta V_{th_S} = \sqrt{K_v^2 \cdot (V_{int_S})^2 + V(z)^2} + \delta_v \quad (6)$$

being

$$K_v = A_{ox} \sqrt{C_{ox} (\Delta V_{BS} - V_{th})} [1 - V_{DS} / \alpha (\Delta V_{BS} - V_{th})] \exp(E_{ox} / E_0) \exp(-E_a / kT) \quad (7)$$

and

$$\Delta V_{th_R} = (V(y) - \delta_v) [1 - \sqrt{\gamma(V_{int_S}) / V_{int_S}}] \quad (8)$$

The variable V_{int_S} which represents the operating time and relaxation time of each transistor is obtained by integrator circuits, which were not represented in fig. 2, implemented by using ideal components (R, C and OpAmp). The voltage between gate and ground nodes of the transistor under analysis (V_{CTRL}) controls the activation of the integrator circuit, allowing the measurement of both operation and relaxation times.

Therefore, during the simulation time, these sources will provide the values corresponding to the threshold voltage variation. When a gate voltage is applied in a given transistor, the threshold voltage variation is calculated by the circuit and provided through the voltage at node 'x', i.e. $V(x)$. V_{CTRL} voltage defines which ideal switch (Sw_1 - Sw_4) is open and closed, allowing the calculation both of stress and recovery phases, independently.

Fig. 3 illustrates how the equivalent circuit controls the V_{BS} dependent source, therefore controlling the threshold voltage of the transistor under evaluation.

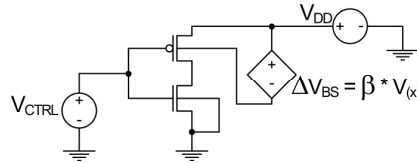


Fig. 3 – CMOS inverter under NBTI aging evaluation.

4. Experimental Results

In order to validate the proposed circuit, HSPICE electrical simulations were carried out in two different CMOS logic gates, an Inverter and a 2-input Nand, being calculated the threshold voltage degradation for each of them. Moreover, the rise transition delay degradation (in comparison with the operation time) was also obtained. For simulations, 45 nm PTM process parameters [12], operating temperature of 100°C, and supply voltage of 1.1V were taken into account.

For Inverter gate, an input signal with a duty cycle of 50% ($t_{stress} = t_{recovery}$) was applied, during a period enough to evaluate the degradation of PMOS V_{th} in long term regime. In the case of Nand simulation, it was applied two different input signals (different duty cycles - 50% and 25%), one on each input, in order to prove that, using this method, it is possible to calculate the degradation of each transistor individually.

In fig. 4, it is shown the degradation of both logic cells, by presenting ΔV_{th} behavior. Notice that, on the Nand gate the transistor that was stimulated with 25% duty cycle signal (dotted line in fig. 4b) suffered more degradation than the other input, excited with 50% duty cycle signal (filled line in fig. 4b).

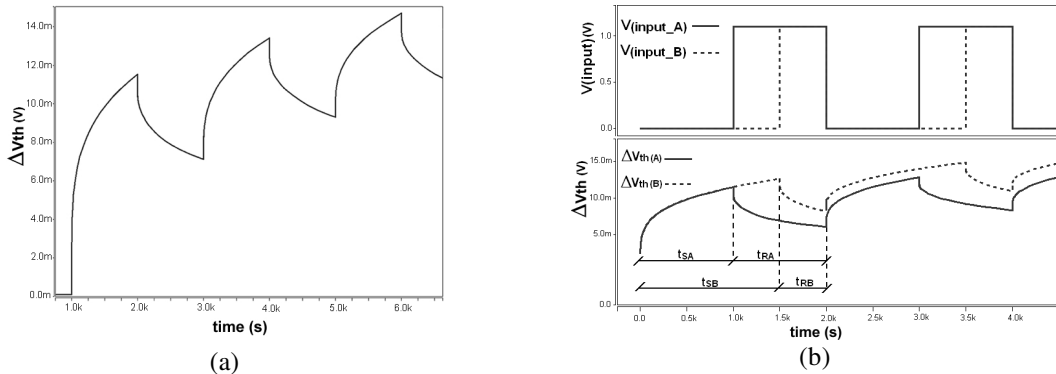


Fig. 4 – Results obtained by HSPICE simulation of the threshold voltage degradation in Inverter (a) and 2-input Nand (b). The labels t_{SA} , t_{SB} , t_{RA} and t_{RB} are the stress time and the recovery time for both inputs – A and B.

Threshold voltage degradation and rise transition delay variation are presented in tab. 2 and tab. 3, considering different operating times for Inverter and Nand gates. In order to better visualize the aging effect during a time interval, the results presented in tab. 2 and tab. 3 are plotted on the graph presented in fig. 5. The Y axis indicates the percentual threshold voltage degradation while the X axis indicates the elapsed time, from 0 to 5 years of operation.

Tab.2 - ΔV_{th} degradation.

Time Operation	Inverter	Nand	
		Input A	Input B
1 hour	2,3 %	2,2 %	2,5 %
1 day	4,4 %	4,4 %	5,4 %
1 month	9,4 %	9,2 %	11,7 %
1 year	16,2 %	15,9 %	20 %
5 years	22,7 %	22,3 %	27,8 %

Tab.3 - Rise delay propagation time degradation.

Time Operation	Inverter gate delay (t_{rise})
1 hour	1,3 %
1 day	2,8 %
1 month	6,4 %
1 year	11,3 %
5 years	15,8 %

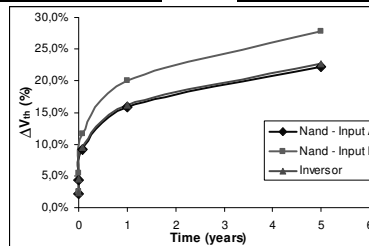


Fig. 5 – The aging effect on the transistors during 5 years of operation (for the Inverter and Nand).

5. Conclusions

An equivalent electrical circuit representing the NBTI R-D model, presented by Vattikonda *et al.* [5], has been proposed. Electrical simulations have demonstrated that it is suitable to predict CMOS cells degradation due to NBTI aging affect, evaluating individually each PMOS transistor at pull-up logic gate network. In future work, more complex CMOS gates, like And-Or-Inverter and Flip-Flops, will be considered.

6. References

- [1] D. Schroder, "Negative bias temperature instability: What do we understand?", Microelectronics Reliability, no. 47, 2007, pp. 841–852.
- [2] Y. Miura, Y. Matukura, "Investigation of silicon–silicon dioxide interface using MOS structure", Japan Journal of Applied Physics, no.5, 1966, pp. 180.
- [3] M. Alam, S. Mahapatra, "A comprehensive model of PMOS NBTI degradation." Microelectronics Reliability 45(1), 2005, pp. 71–81.
- [4] J. Massey, "NBTI: What We Know and What We Need to Know – A Tutorial Addressing the Current Understanding and Challenges for the Future", IRW Final Report, 2004.
- [5] R Vattikonda. et al., "Modeling and minimization of PMOS NBTI effect for robust nanometer design", Proc. Design Automation Conf., 2006.
- [6] S. Bhardwaj, et al., "Predictive modeling of the NBTI effect for reliable design", In Proc. IEEE Custom Integrated Circuits Conference, 2006, pp. 189–192.
- [7] S. Kumar et al., "NBTI-Aware Synthesis of Digital Circuits", DAC, 2007.
- [8] S. Chakravarthi, et al., "A Comprehensive Framework for Predictive Modeling of Negative Bias Temperature Instability," in Proc. IEEE Int. Reliability Physics Symposium, April 2004, pp. 273–282.
- [9] M. A. Alam, "A Critical Examination of the Mechanics of Dynamic NBTI for pMOSFETs," in IEEE International Electronic Devices Meeting, December 2003, pp. 14.4.1–14.4.4.
- [10] S. V. Kumar, C. H. Kim, S. S. Sapatnekar, "An analytical model for negative bias temperature instability," in Proc. ICCAD, 2006, pp. 493–496.
- [11] K. O. Jeppson, C. M. Svensson, "Negative bias stress of MOS devices at high electric fields and degradation of MNOS devices", Journal Applied Physics, 1977, 48(5), pp. 2004–14.
- [12] PTM web site, <http://www.eas.asu.edu/~ptm>.
- [13] S. Rangan, N. Mielke, E. C. C. Yeh, "Universal recovery behavior of negative bias temperature instability," IEDM, 2003, pp. 341–344.
- [14] G. Chen, et al., "Dynamic NBTI of PMOS transistors and its impact on device lifetime," IRPS, 2003, pp. 196–202.
- [15] V. Huard, M. Denais, "Hole trapping effect on methodology for DC and AC negative bias temperature instability measurements in pMOS transistors," IRPS, 2004, pp. 40–45.
- [16] N. Weste and David Harris, "CMOS VLSI Design: A Circuits and Systems Perspective", 3rd. ed., Addison Wesley, 2004.

Evaluation of XOR Circuits in 90nm CMOS Technology

Carlos E. de Campos, Jucemar Monteiro, Robson Ribas, José Luís Güntzel

{ cadu903,jucemar, guntzel}@inf.ufsc.br, rob_ribas@yahoo.com.br

System Design Automation Lab. (LAPS)
Federal University of Santa Catarina, Florianópolis, Brazil

Abstract

This paper evaluates six different CMOS realizations of XOR function: two with simple static CMOS gates only, one that uses complex static CMOS gate, one using transmission gate (TG), one using pass transistor and a traditional 6-transistor realization. These six versions are evaluated through electric-level simulations for the 90nm predictive technology model (PTM). The obtained data for power-delay product (PDP) shows that the 6-transistor XOR and the TG-based XOR present the best tradeoff between power and delay. When PDP is combined with transistor count to estimate realization costs, the 6-transistor XOR and the pass transistor XOR appear as the two best choices.

1. Introduction

The exclusive or (XOR) function is widely used in digital circuit design. It has application not only in arithmetic circuits, as adders and subtractors, but also in comparators, parity generators and error correction and detection circuits [1]. Therefore, the performance of XOR realization is of great interest since it can greatly influence on the performance of the whole circuit.

Although many different circuits exist for realizing the XOR function, in the last past years a number of new XOR circuits have been proposed [2][3]. Generally, these circuits adopt specific transistor topologies to produce both XOR and XNOR functions, resulting in more transistors than in the case only the XOR is realized. Such topologies are generally thought to be applied to full custom layout.

Another important aspect is that competition in the electronic industry continues to shorten the design cycle, hence increasing the pressure on designers. Therefore, in order to assure productivity, designers must rely on EDA tools to successfully accomplish their designs within the required schedule. Existing design flows with commercial EDA tools cover the whole steps of synthesis, including the physical synthesis step (also referred to as “backend”). The majority of commercially available physical synthesis tools rely on cell-based layout assembly. In such approach there is a limited number of layout choices for each logic gate, where this choices include layout topologies (different sized gates) and logically equivalent layout versions. In order to take full advantage on the physical synthesis tool without compromising the quality of the produced layout, it is important that designers be conscious of the limitations of the cell library. In the case of arithmetic circuits that demand high speed and low power, the way the XOR gates of the netlist are mapped to the cells of the library may have a significant impact on the generated circuit. In those cases it is also useful that the designer understands the tradeoffs between the most popular XOR gates concerning delay, power and transistor count.

This paper investigates delay, power and transistor count of six circuits that realize the XOR function. The original motivation behind the work was the evaluation of delay and power of XOR function mapped with simple and complex static CMOS gates [4]. However, due to the advent of compact portable consumer electronics, arithmetic circuits must consume the less possible energy. Therefore, we decided to include in our experiments three XOR circuits that are not exclusively based on static complementary CMOS gates.

2. Evaluated XOR circuits

Fig. 1 shows the XOR circuits that were investigated. Two versions of the circuit of fig. 1a were characterized: one with a 10fF capacitive load attached to each internal node (XOR2) and another one without internal loads (XOR1). The 10fF loads of version XOR2 are intended to model the parasitic capacitance associated to the routing, in case this circuit is used as the result of an automatic mapping done by a synthesis tool. Therefore, by comparing versions XOR1 and XOR2 one can observe the effect of a low quality placement on delay and power consumption of a (CMOS equivalent) sum-of-products realization of XOR function, when a typical physical design flow is used. Both XOR1 and XOR2 versions need 16 transistors.

Version XOR3 (fig. 1b) is realized by a single complex static CMOS gate plus two inverters. This is a cheaper alternative to realize the XOR function in a full restoring topology, since it needs 12 transistors.

Versions XOR4 (fig. 1c) and XOR5 (fig. 1d) are not able to restore the signal level at the gate output and therefore are more susceptible to noise. On the other hand, this is the reason why these XOR circuits tend to consume less power. However, since in XOR4 only one type of transistor is used to transmit the logic value from the input to the output, this XOR circuit may exhibit very poor signal levels at its output [4]. Thus, it

cannot be used in cascaded circuits that have many stages because the logic level can be degraded. This problem can be overcome by using transmission gates (TGs) in the place of pass transistors, as in XOR5. Apart from noise immunity and signal integrity problems, XOR4 and XOR5 require few transistors: only 4 in the case of XOR4 and 8 in the case of XOR5.

Fig. 1e shows the popular 6-transistor XOR circuit. This circuit exhibits a good compromise between transistor count (only 6) and electrical characteristics. Although it may present signal degradation problems, it can partially restore a degraded input signal (when applied to input Y). It also relies on TG to transmit input X to the output, thus maintaining logic level integrity.

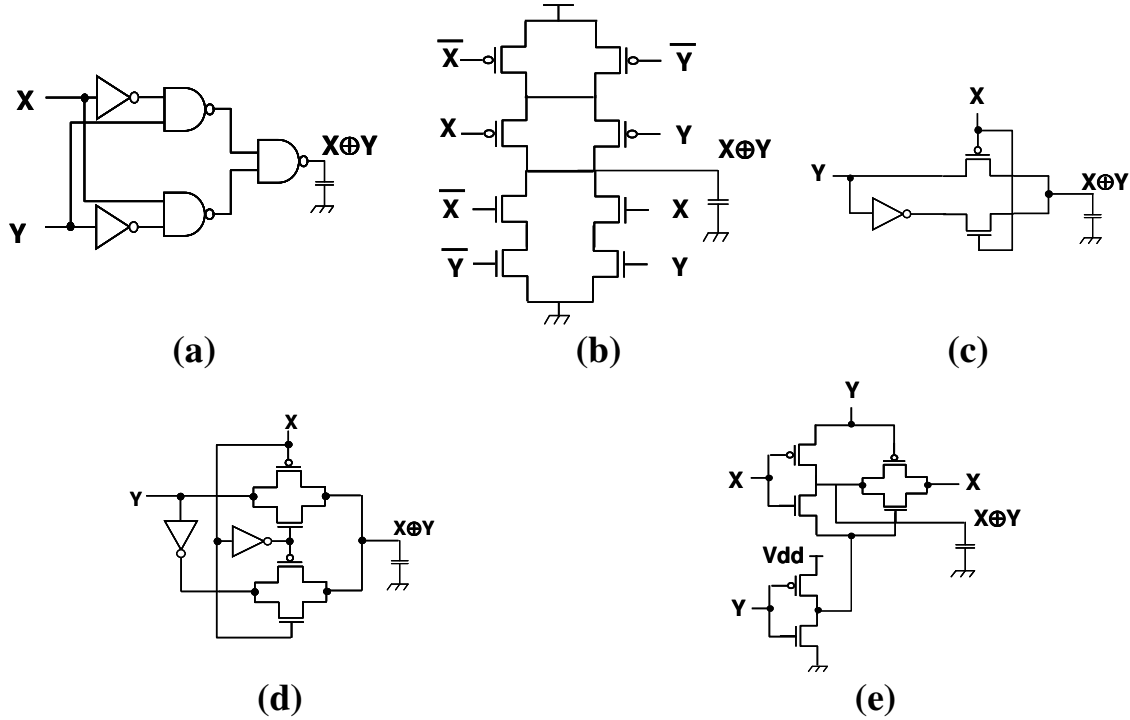


Fig. 1 – Investigated XOR circuits: XOR1 and XOR2 (a), XOR3 (b), XOR4 (c), XOR5 (d) and XOR6 (e).

3. Practical Experiments and Results

The six XOR versions were described in Spice format and simulated for the 90nm “typical case” parameters from the predictive technology model (PTM) [5] by using Synopsys HSpice simulator [6]. All NMOS transistors were sized with $W_n=360\text{nm}$ whereas all PMOS transistors were sized with $W_p=720\text{nm}$. In order to obtain the critical rising and falling delays of each XOR version, all possible single-input transitions were simulated. Besides this, each XOR version was simulated considering the output load capacitance varying from 20fF to 200fF, by a step of 20fF. Fig. 2a plots the evolution of maximum falling delay of the six XOR circuits with respect to the capacitive load, whereas fig. 2b shows the evolution of the maximum rising delay of the same circuits with respect to the same output load variation.

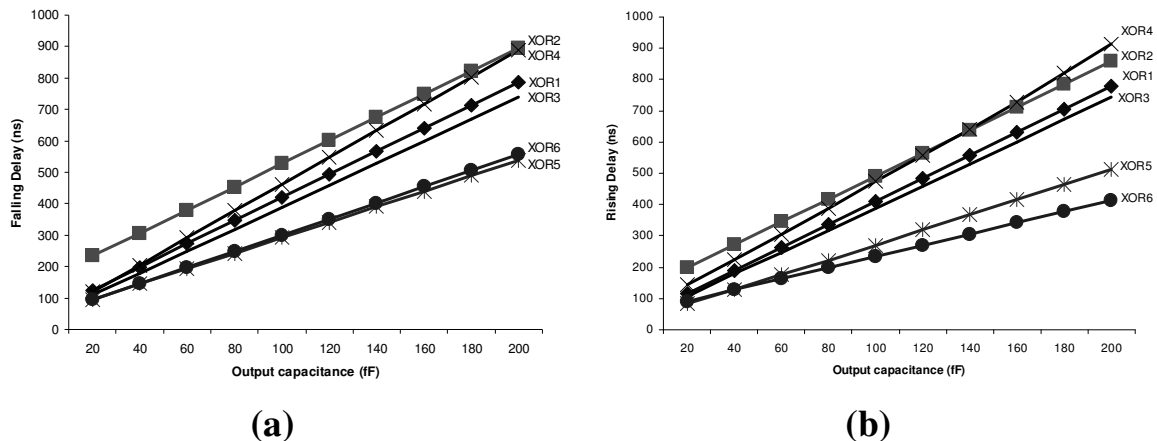


Fig. 2 – Critical delays of the six investigated versions of XOR circuits: falling (a) and rising delays (b)

The data of fig. 2 show that XOR5 and XOR6 are the fastest XOR circuits concerning both falling and rising transitions. XOR6 exhibits the smallest rising delays for output loads greater than 60fF whereas XOR5 exhibits the smallest falling delays for output loads greater than 120fF. For the other load cases these two XOR circuits have similar rising/falling delays. This way, concerning only delay, XOR5 and XOR6 may be declared as the best choices. Particularly for a 200fF load, XOR5 and XOR6 are about twice as fast as the fastest XOR circuit among the others. Concerning the falling delay, XOR2 is the slowest version for all output loads, followed by XOR4. Concerning the rising delay, XOR2 is the slowest XOR circuit for output loads up to 120fF. For output loads greater than 160fF XOR4 presents the greatest rising delay.

As mentioned in section 2, the XOR2 10fF internal capacitances are intended to model the inter-cell wires resulting from a poor quality placement for a CMOS equivalent sum-of-products realization of XOR function. Unlikely, XOR1 represents the same XOR mapping (CMOS equivalent sum-of-products), but considering that inter-cell routing can be disregarded, as the result of a good quality placement step. The intention of such experiment was to measure how severe can be the effect of the physical synthesis step for such tpe of mapping, when a 90nm technology is assumed. Not surprisingly, XOR2 presents the greatest rising delay for all output loads, and the greatest falling delays for output loads ranging from 20fF to 120fF. On the other hand, XOR1 has the third greatest falling and rising delays among all XOR versions. These results reveal that the performance of the sum of product version of XOR (XOR1 and XOR2) is highly sensible to the placement of the simple CMOS gates used. Therefore, when performance is a metric to be optimized in the automatic synthesis of arithmetic circuits with traditional (cell based) physical design flow, this XOR topology must be avoided.

Fig. 3 shows the average power of the six XOR circuits, as estimated from Hspice simulations. For each XOR circuit, the eight possible single input transitions were simulated within a total simulation time of 80 ns.

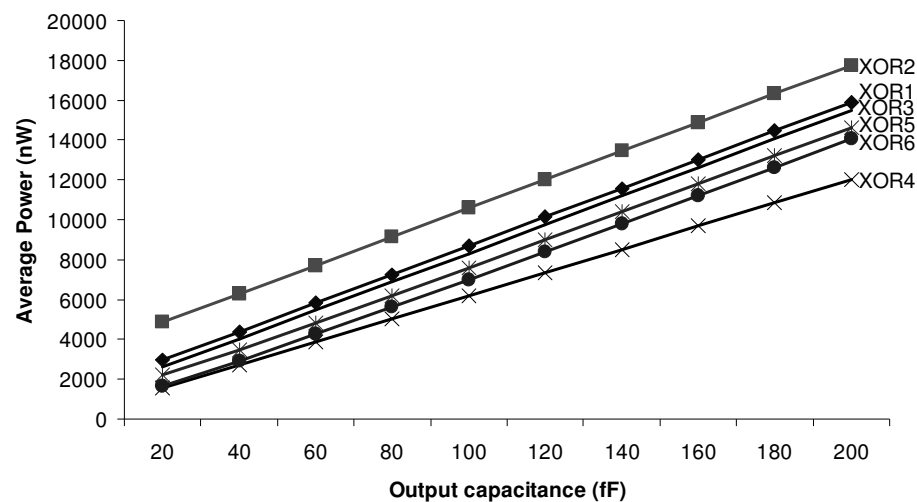


Fig. 3 – Average power consumption of the six XORs circuits

The data of fig 3 show that XOR4 is the XOR version that consumes less power among all investigated circuits. Moreover, the difference in power consumption between XOR4 and the other versions is remarkable mainly for high output capacitive loads (from 160fF to 200fF). On the other hand, XOR6 and XOR5 are the second and the third circuits requiring less power to operate within the defined output loads, respectively. Although they are not as economical as XOR4, they are not so power hungry as XOR2 and XOR1.

As it could be expected, XOR2 is the version that consumes more power for any output load. In addition to this fact, one may observe that XOR1 is the second XOR version consuming more power. These facts reinforce the affirmation that sum of products realizations of XOR is not appropriate when performance (critical delay and power) are to be optimized.

Tab. 1 presents comparative results using delay and power estimates obtained for 200fF output load. Besides falling and rising delays and average power, transistor count, power-delay product (PDP) and power-delay product versus transistor count (PDP x TC) values are also showed. XOR2, XOR3, XOR4, XOR5 and XOR6 values are normalized with respect to XOR1 values.

Delay and power normalized results confirm the tendencies already mentioned in the former paragraphs. In addition, the PDP is able to simultaneously take into account both delay and power. By analyzing normalized PDP values one can observe that XOR6 offers the best tradeoff between power and delay. In the same sense, XOR5 and XOR4 are the two next options. When combining the three metrics, power, delay and transistor count (PDP x TC) XOR6 is still the best version. However, XOR4 also shows to be very promising, since its PDP x TC value is very close to the one of XOR6.

Tab.1 – Normalized comparative results: delay and power considering a 200fF output load

	XOR1	XOR2	XOR3	XOR4	XOR5	XOR6
Max rising delay	-	1.10	0.95	1.17	0.66	0.53
Max falling delay	-	1.14	0.94	1.13	0.69	0.71
Average power	-	1.12	0.97	0.75	0.92	0.88
Transistor count (TC)	-	1.00	0.75	0.25	0.50	0.37
PDP	-	1.25	0.92	0.86	0.62	0.55
PDP x TC	-	1.25	0.69	0.22	0.31	0.20

4. Conclusions

This paper has investigated the critical delay and the power of six different XOR circuits. Electric-level simulation results have showed that CMOS equivalent sum-of-product XOR realizations (as XOR1 and XOR2) must be avoided when high performance low power arithmetic circuits are needed. The data also put in evidence the harm effect on delay and power of the XOR function originated from a poor quality placement for the sum-of-products mapping. This fact allows us to conclude that it may be very difficult to achieve high performance arithmetic circuits by using automatic physical design flows if sum-of-products are used to map the XOR function.

The experimental data also cleared that TG-base XOR (XOR5) and 6-transistor XOR (XOR6) present the best tradeoffs between delay and power. However, when power-delay product is combined with transistor count (to incorporate the implementation cost of a given XOR circuit), the two best choices are XOR6 and XOR4. These features make XOR6, XOR4 and XOR5 the most appropriate XOR circuits among the investigated ones, for building low-cost, high-performance low-power arithmetic circuits.

Future works include the evaluation of other XOR topologies as well as simulations with actual fabrication technologies.

5. References

- [1] J.-M. Wang, S-C Fang and W.S. Shiung, "New efficient designs for XOR and XNOR functions on the transistor level", *IEEE Journal of Solid-State Circuits.*, vol. 29, no. 7, pp. 780-786, July 1994.
- [2] H. T. Bui, Y. Wang and Y. Jiang, "Design and analysis of low-power 10-transistor full adders using XOR–XNOR gates," *IEEE Trans. Circuits Systems II: Analog Digital Signal Processing*, vol. 49, no. 1, pp. 25–30, Jan. 2002.
- [3] S. Goel, M. A. Elgamel and M. A. Bayoumi, "Design methodologies for high-performance noise-tolerant XOR-XNOR circuits", *IEEE Trans. Circuits Systems I: regular papers.*, vol. 53, no. 4, pp. 867-878, April 2006.
- [4] J.M. Rabaey, A. Chandrakasan, B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd. ed. [S.l.]: Prentice-Hall, 2003. 623-719 p.
- [5] Y. Cao et al., "New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Simulation." In: CICC'00, pp.201–204.S.
- [6] Synopsys Tools. Available at <http://www.synopsys.com/Tools/Pages/default.aspx> (access on March 16, 2009).

A Comparison of Carry Lookahead Adders Mapped with Static CMOS Gates

Jucemar Monteiro, Carlos E. de Campos, Robson Ribas, José Luís Güntzel
{jucemar, cadu903,guntzel}@inf.ufsc.br, rob_ribas@yahoo.com.br

System Design Automation Lab. (LAPS)
Federal University of Santa Catarina, Florianópolis, Brazil

Abstract

This paper presents an evaluation of carry lookahead adders (CLAs) mapped with different static CMOS gates. Three types of mappings were considered for the 4-bit CLA module: one with simple static CMOS gates (NAND, NOR and inverter) and two using complex static CMOS gates. Two topologies of 8-bit, 16-bit, 32-bit and 64-bit width CLAs were validated for the 90nm predictive technology model (PTM) by using Synopsys HSpice simulator. After that, their critical delay was estimated by using Synopsys Pathmill tool. The results show that the critical delay of CLAs mapped with complex static CMOS gates may be up to 12,4% smaller than that of their equivalents mapped with simple static CMOS gates. Moreover, the CLAs mapped with complex static CMOS gates require 26.8% to 29.4% less transistors. It was also evaluated the impact of hierarchy in the critical delay of CLAs.

1. Introduction

Addition is by far the most important arithmetic operation. Every integrated system has at least an adder. Moreover, many other arithmetic operations may be derived from addition. Therefore, efficient hardware realizations of addition are still subject of great interest for both industry and academia, since the performance of the whole integrated system may be affected by the performance of the arithmetic operations themselves [1].

Among the several existing addition architectures, the ripple-carry adder (RCA) [2] [3] is the most intuitive parallel adder and thus, may be the first architecture to be considered by designers, although its performance is quite poor. On the other hand, the carry lookahead adder (CLA) architecture [4] is able to operate at almost one order of magnitude faster than RCA architecture with a moderate increase in hardware cost. Therefore, CLA is preferable to RCA when high performance is desired.

While adopting a fully automated design flow various circuit solutions may exist, each of them leading to different characteristics in terms of performance, power and transistor count. Therefore, it is the responsibility of the designer to explore the design space in order to find the best possible solution. In this context, part of such space is provided by the choice of gates to be used, the so-called technology mapping (or technology binding) step, which is prior to the layout generation step.

In the design of integrated systems where performance requirements are not too stringent and/or the time-to-market is the most relevant restriction, designers rely on a fully automated design flow, where circuit layout is obtained by using backend physical design EDA tools. Even in such scenario, high performance arithmetic circuits may be needed. The unavailability of physical design tools devoted to arithmetic circuits lead designers to adopt conventional physical design flows that use standard cells. This way, the circuit to be realized must be mapped using only the logic functions available in the cell library, which may be quite limited. Even in the case automatic cell library generation is used, cells are generated according to certain predefined topologies (e.g., static complementary CMOS only). Hence, some fast adder architectures that achieve high speed thanks to specific mappings (e.g., the Manchester chain [5]) must be logically remapped, what can cause a significant loss of speed. This way, the CLA appears as the most promising fast adder architecture when an industrial physical design flow is to be adopted.

Having this context in mind, this paper presents a comparison of three mappings of carry-lookahead adders (CLAs) which are used to build two different architectures: ripple CLA and fully hierarchical CLA. Total transistor counts and critical delay estimates, obtained by using an industrial timing analysis tool, are used to evaluate the impact of adder architecture and technology mapping on the investigated CLAs.

2. Mapping 4-bit carry lookahead adders for automatic design flows

The speedup achieved by the CLA architecture comes from the way it computes the carries. Basically, all carries are computed in parallel [4] by providing individual circuits to compute each carry. To make this evident, the equations of the carries are unrolled, in such a way that each carry does not depend anymore on the previous one, but only on signals propagate (p_i) and generate (g_i), as described by the following equations:

$$C_0 = g_0 + p_0 \cdot C_{in} \quad (1)$$

$$C1 = g1 + p1 \cdot g0 + p1 \cdot p0 \cdot Cin \quad (2)$$

$$C2 = g2 + p2 \cdot g1 + p2 \cdot p1 \cdot g0 + p2 \cdot p1 \cdot p0 \cdot Cin \quad (3)$$

Generalizing:

$$Ci = gi + pi \cdot Ci-1 \quad (4)$$

with

$$gi = Ai \cdot Bi \quad (5)$$

$$pi = Ai \oplus Bi \quad (6)$$

Due to the cost associated with such parallelization of the carry, the width of the CLA basic block is generally limited to 4 bits.

In order to analyze the impact of the technology mapping on CLA performance and transistor count, three different versions of 4-bit CLA were described and simulated. Each version uses a different set of CMOS gates to implement the carry chain (equations 1 to 4), but all of them use the same circuit for the XOR function (the 6-transistor XOR gate) to implement signals pi (equation 6) and the sum outputs (not shown in equations). These versions are referred to as “S”, “C” and “Cg”. Version “S” uses only simple static CMOS gates (NAND, NOR and inverter) to implement the carry chain, whereas versions “C” and “Cg” use complex static CMOS gates. The complex gates topology is chosen looking to minimize transistor count. The difference between versions “C” and “Cg” relies on the fact that version “Cg” uses complex CMOS gates to map equations 1 to 5, whereas version “C” uses complex CMOS gates to map equations 1 to 4. As a result, version “Cg” combines signal gi with the carry chain. It is important to remark that the three mappings were accomplished by hand. Fig. 1a shows the logic schematic of the 4-bit CLA module of mapping “C”.

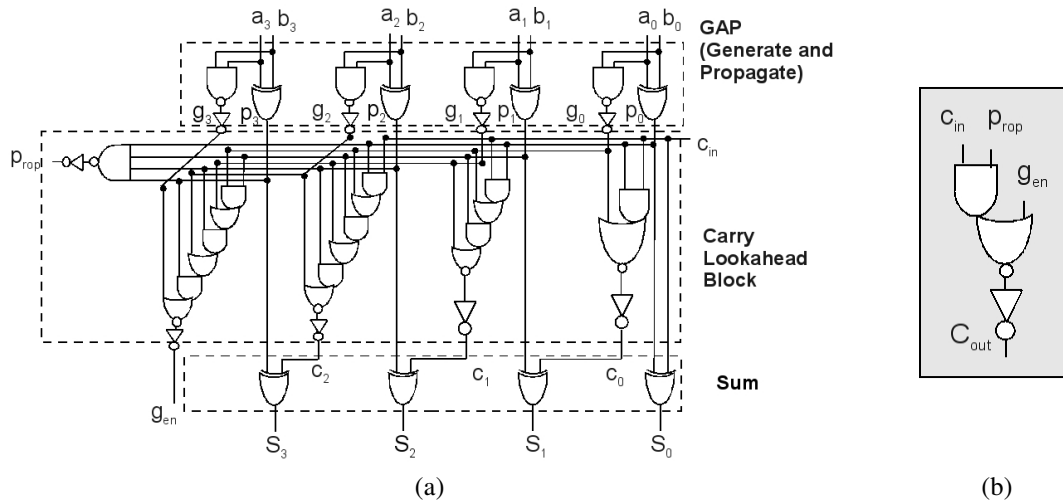


Fig. 1 – Schematic of version “C” of the 4-bit CLA module (a) and circuit to compute the carry out (b).

3. Building higher order CLAs

The three versions of 4-bit CLA module described in the latter section were used to build 8-bit, 16-bit, 32-bit and 64-bit width CLAs. Two different CLA architectures were considered: one that is built up by connecting 4-bit CLA modules in a ripple-carry fashion and another one that is built up by connecting 4-bit CLA modules in a fully hierarchical way [1]. Fig. 2 shows the block diagram of these two CLA architectures, which are referred to as “r” and “h” herein.

The combination of the three versions of 4-bit CLA module with the two architectures for higher order CLAs gives rise to six different versions of CLAs that are the subject of the comparison accomplished in this work. These six versions are referred to as: CLArS, CLArC, CLArCg, CLAhS, CLAhC and CLAhCg.

It is important to observe that in order to connect the 4-bit CLA modules in a ripple-carry way (CLA versions identified with “r”), each 4-bit CLA module is modified to compute its own carry out. In the case of version “C” of the 4-bit CLA module, the circuit used to compute the carry out is shown in fig. 1b. In the case of version “S”, the circuit to compute the carry out uses only simple static CMOS gates.

4. Experimental results and comparison

The six CLA versions were described in Spice format and validated for the 90nm “typical case” parameters from the predictive technology model (PTM) [6] by using Synopsys HSpice simulator [7]. After that, their critical delay was estimated by using Synopsys Pathmill tool [7] assuming 10fF as output charge.

The graphs of figs. 3a and 3b show the total transistor count for the ripple CLAs (“CLAr”) and for the fully hierarchical CLAs (“CLAh”), respectively. It is possible to observe that, for a given mapping (S, C or Cg) the CLAh versions require more transistors than the corresponding CLAr versions. This is due to the logic required to connect the 4-bit CLA modules in a hierarchical manner. Also, and more important, is the fact that the “C” and “Cg” CLAs require less transistors than “S” CLAs. This behavior is the same for both CLAr and CLAh. It is also remarkable that such difference increases with the increase in CLA bit width, meaning that the wider is the adder, the most advantageous is the use of complex static CMOS gates (instead of simple static ones).

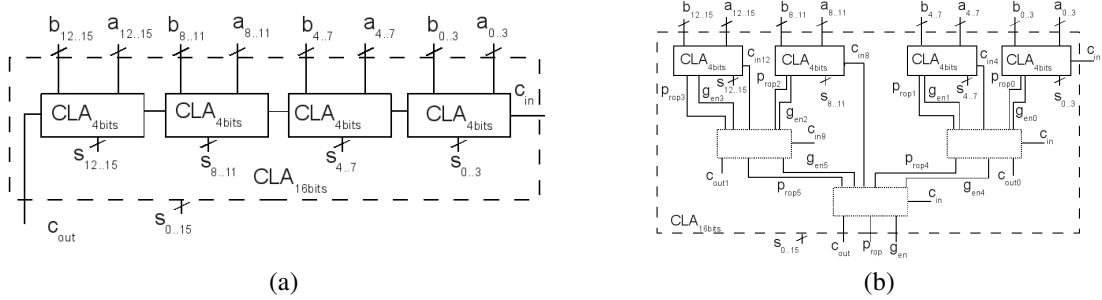


Fig. 2 – Diagram blocks for 16-bit “ripple” CLA (a) and 16-bit “fully hierarchical” CLA (b).

Comparing only the CLAs with carry chain mapped with complex static CMOS gates, one can notice that the “Cg” CLAs require less transistors than the “C” CLAs. However, this difference is not significant and therefore, it is not possible to claim that “Cg” CLAs are the best choice prior to analyze other relevant metrics, as for instance, critical delay.

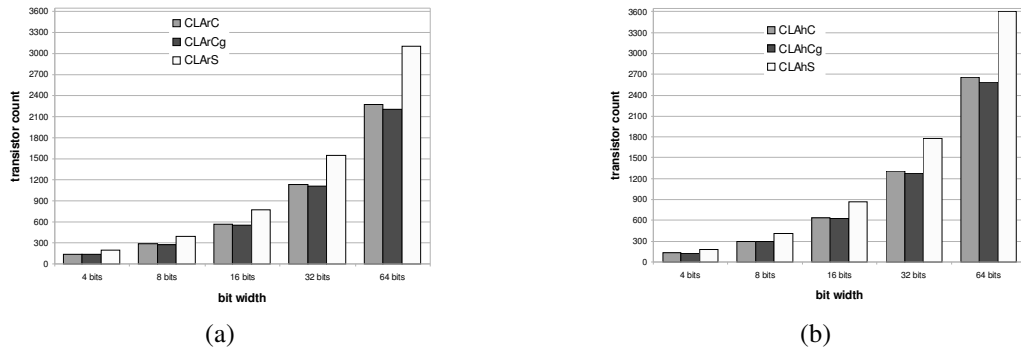


Fig. 3 – Total transistor count for CLAr versions (a) and for CLAh versions (b).

Figs. 4a and 4b show the critical delay for CLAr and CLAh versions, respectively, as reported by PathMill. As one could expect, the critical delay of the CLAh’s increases almost linearly with respect to the increase in bit width. This is not true for CLAr’s. Such behavior of the CLAh’s critical delay comes from the way 4-bit CLA modules are connected in such CLA architecture, that is, fully hierarchical. Such feature is responsible for a significant speedup in higher order CLAh’s: 32-bit CLAh’s are 15,9% (in average) faster than 32-bit CLAr’s, whereas 64-bit CLAh’s are 39,0% (in average) faster than 64-bit CLAr’s!

On the other hand, the impact of the technology mapping on the critical delay of CLAs is more prominent in higher order CLAs and mainly, on the CLAr’s. The graphs of fig. 4 also show that for all CLAs, but the 4-bit CLAr’s, mapping “C” resulted in faster CLAs than mapping “Cg”. This is because when signal generate (gi) is mapped together with the carry chain (this is the case of mapping “Cg”) the (average) number of stacked transistors of the required complex static CMOS gates becomes greater. Since the delay of complex static CMOS gates is proportional to the number of stacked transistors, the “Cg” mapping of carry chains results in slower circuits than those carry chains obtained from the “C” mapping.

By using the same data showed in Fig. 4, Tab. 1 highlights the speedup provided by the use of complex gates (“C” and “Cg” mappings). The critical delays of the CLAs mapped with simple gates (“S”) are used as reference. The data in tab. 1 reveals that the use of complex static CMOS gates results in 10,59% and 12,43% of speedup for 32-bit and 64-bit CLAr’s that use the “C” mapping and 5,47% and 9,42% of speedup for the same width of CLAr’s, when mapping “Cg” is used. On the other hand, the use of complex gates results in speedups of 9,31% and 10,82% for 32-bit and 64-bit CLAh’s that use the “C” mapping and 4,07% and 6,33% for the

same widths of CLAh's if mapping "Cg" is adopted. As highlighted by the data in tab. 1, the impact of using complex static CMOS gates is more relevant for the CLAr's. This is because the delay of CLAh's is already optimized by the hierarchy of the connections, in such a way that there is not much room to further improvements. On the other hand, mappings "C" resulted in greater speedup than mappings "Cg".

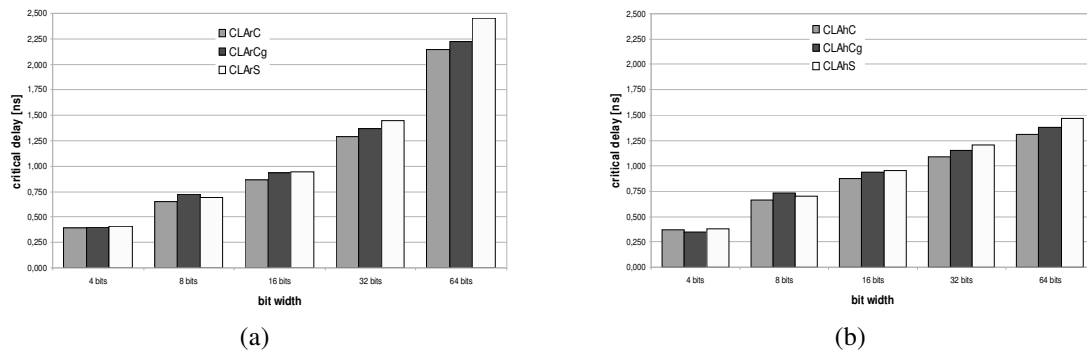


Fig. 4 – Critical delay for CLAr versions (a) and for CLAh versions (b).

Tab.1 – Speedup provided by the use of complex static CMOS gates (in %).

Bit width	100x (CLAhS-CLAhC) /CLAhS	100x (CLAhS-CLAhCg) /CLAhS	100x (CLArS-CLArC)/ CLArS	100x (CLArS-CLArCg) /CLArS
4 bits	2.65	8.20	4.18	2.21
8 bits	5.13	-4.13	5.66	-5.08
16 bits	8.10	1.37	8.18	0.32
32 bits	9.31	4.07	10.59	5.47
64 bits	10.82	6.33	12.43	9.42

5. Conclusions and perspective work

This paper has evaluated six carry lookahead adders (CLAs). These CLAs differ by the mapping of the 4-bit CLA basic module and also by the way these modules are connected together to form wider CLAs. Two of the mappings use complex static CMOS gates whereas another one use simple gates only. The way the 4-bit CLA modules are connected together gives rise to two CLA architectures: the "CLAr" (ripple CLA) and the "CLAh" (fully hierarchical CLA).

Experimental results showed that the use of complex static CMOS gates leads to CLAs that require less transistors (up to 29.4% less) and are faster (up to 12.4% of speedup) than their counterparts mapped with simple gates only. Results also highlighted that connecting CLA modules in a fully hierarchical fashion greatly contributes to reduce CLA critical delay.

Perspective works include repeating all the experiments by using backannotated data and complete power analysis of the investigated CLAs.

6. References

- [1] V. Oklobdzija, "High-Speed VLSI Arithmetic Units: Adders and Multipliers." In: *Design of High-Performance Microprocessor Circuits*, IEEE Press, 2001, pp. 181-204.
- [2] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, New York: Wiley, 1979.
- [3] J.M. Rabaey, A. Chandrakasan, B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd. ed. [S.l.]: Prentice-Hall, 2003. 623-719 p.
- [4] A. Weinberger and J.L. Smith, "A Logic for High-Speed Addition", In: National Bureau of Standards, Circulation 591, 1958, pp. 3-12.
- [5] T. Kilburn et al, "Parallel Addition in Digital Computers: A New Fast 'Carry' Circuit", *Proceedings of IEE*, Vol. 106, pt. B, 1959, pp. 464.
- [6] Y. Cao et al., "New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Simulation." In: *CICC'00*, pp.201-204.S.
- [7] Synopsys Tools. Available at <http://www.synopsys.com/Tools/Pages/default.aspx> (access on March 16, 2009).

Optimization of Adder Compressors Using Parallel-Prefix Adders

João S. Altermann, André M. C. da Silva, Sérgio A. Melo, Eduardo A. C. da Costa
jaltermann@hotmail.com, {andresilva, smelo, ecosta}@ucpel.tche.br

Laboratório de Microeletrônica e Processamento de Sinais - LAMIPS
Universidade Católica de Pelotas - UCPEL
Pelotas, Brasil

Abstract

Adder compressors are the type of circuits that add more than two operands simultaneously. The most basic structure is the 4:2 compressor that adds four operands simultaneously. The main aspect of this compressor is the reduced critical path composed by Exclusive-OR gate and multiplexer circuits. In this compressor it is needed one line of Ripple Carry adder (RCA) in order to perform the recombination of partial carry and sum results. However, this type of adder is not efficient because carry is propagated between all the addition blocks. In this work we propose the improvement of performance of the adder compressors by using more efficient parallel prefix adders in the carry and sum recombination line. We have also applied these efficient adders in the extended versions of 8-2 and 16:2 adder compressor circuits. The results show that although the compressors with RCA present less power consumption, these circuits can present 60% delay reduction when using parallel prefix adders, thus the adder compressors using parallel prefix adders are more efficient when the power delay product (PDP) metric is taken into account.

1. Introduction

Addition is one of the most basic arithmetic operation. This type of operation is present in many applications such as multiplication, division, filtering, etc. The most basic circuit to perform the addition is the Ripple Carry Adder (RCA) [1]. This adder is composed by a cascade of full adder blocks. Although this adder is composed by a simple structure, that enables an easy implementation, it presents a bad performance because the carry result is propagated between all the full adder blocks. Thus, when performance result has to taken into account, parallel prefix adders have been the best choice [2] Parallel-prefix adders are very suitable for VLSI implementation since they rely on the use of simple cells and keeps regular connection between them. The prefix structures allow several tradeoffs between: i) the number of cells used, ii) the number of required logic levels and iii) the cells fan-out [2].

Although the parallel prefix adders are more efficient than the RCA adders, they do not enable the addition of more than two operands simultaneously. For this type of operation the 4-2 adder compressors circuits [3] have been used as the way of perform the addition of four operand simultaneously, with no penalties in the critical path. In this work we have implemented 16-bit wide 4-2 adder compressor and extended versions of 8-2 and 16-2 adder compressors.

In the n-bit wide 4-2 adder compressor, the recombination of partial carry and sum results is performed by a line of RCA adder. However, as mentioned before, this type of adder presents a bad performance, because the carry propagation between the addition blocks. Thus, in this work we propose the use of more efficient parallel-prefix adders in the compressor architectures. These more efficient adders reduce the critical path in the recombination line of partial carry and sum results.

The results we present show that although the adder compressors present less power consumption when using RCA, they are significantly more efficient when using parallel-prefix adders in its structure. A range from 40% to 63% is achievable in terms of delay reduction. We have also presented that by using parallel-prefix adders the compressors can save up to 22% in the power delay product (PDP) metric.

2. Parallel-prefix adders

A parallel prefix circuits computes N outputs from N inputs using an associative operator. Most prefix computation pre-compute intermediate variables from the inputs. The prefix network combines these intermediate variables to form the prefixes. The outputs are post computed from the inputs and the prefixes. In the parallel-prefix adders the sum are divided in three main steps.

- Pre-computation: Generate the intermediate generate and propagate pairs (g, p) using AND and exclusive-OR gates as follows the equations (1) and (2).

$$gi = Ai \cdot Bi \quad (1)$$

$$pi = Ai \oplus Bi \quad (2)$$

- Prefix: All the signs calculated in the first step are processed by a prefix network, using a prefix cell operator shown in Fig.1 forming the carry signals [1].

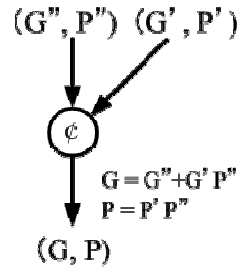


Fig. 1 – The prefix cell operator

- Post-computation: Using the intermediate signals propagate and the carry signals can find a sum result using other exclusive-OR gate as follows the equation (3).

$$Si = Pi \oplus Ci - 1 \quad (3)$$

In practical terms, an ideal prefix network would have: i) $\log_2(n)$ stage of logic, ii) fan-out never exceeding 2 at each stage and iii) no more than one horizontal track of wire at each stage.

Many parallel-prefix networks have been described in literature to calculate the carry signals by taken into account the mentioned characteristics. The classical networks include Brent-Kung [4], Sklansky [5], and Kogge-Stone [6] alternatives. Although Brent-Kung network presents a minimal number of operators, which implies in minimal area, this structure has a maximum logic depth that implies longer calculation time. In order to solve the problem of the logic depth, Kogge-Stone structure appears as a good alternative. This structure introduces a prefix network with a minimal fan-out of an unit at each node. However, the reduction of logic depth is obtained at cost of more area. Others prefix networks have appeared in literature to calculate carries with area and delay variables. The most known structures are Han-Carlson [7], Ladner-Fischer [8], Knowles [9], Beaumont-Smith [10], and Harris [11]. Some of the structures are based on Kogge-Stone approach. A modified parallel-prefix adder using the Ling Carry is proposed in [2]. The arrangement proposed in [2] can reduce the delay of any parallel-prefix adder. This technique uses the equations of ling carries to compute the prefix network. In this work we have used this technique in the Kogge-Stone network in order to find a minimal delay value

3. Adder compressor

Adder compressors appear in literature as efficient addition structures, since these circuits can add many inputs simultaneously with minimal carry propagation [3]. In this work we implement 4-2 compressors and two extended versions of 8-2 and 16-2 compressors as showing figures 4 and 5, respectively.

3.1. Adder compressor

In [12] it was presented a structure to add four inputs simultaneously. This structure was named 4-2 Carry Save Module and contains a combination of full adder cells in truncated connection in which it is possible a fast compression. This structure was after improved by [3]. The 4-2 compressor has five inputs and three outputs, where the five inputs and the output Sum have the same weight (j). On the other hand, the outputs Cout and Carry have a greater weight (j+1). The Sum, Cout and Carry terms are calculated according to equations (4), (5) and (6) [13]. On important point to be emphasized in the equations is the independence of the input carry (Cin) in the output carry Cout. This aspect enables higher performance addition implementation.

$$Sum = A \oplus B \oplus C \oplus D \oplus Cin \quad (4)$$

$$Cout = (A \cdot B) \cdot (A \cdot C) \cdot (B \cdot C) \quad (5)$$

$$Carry = [A \oplus B \oplus C] \cdot (D \cdot Cin) \cdot (D \cdot Cin) \quad (6)$$

The improvement in the structure of a 4-2 compressor circuit, using multiplexer is shown in the Fig. 2. This structure has a more reduced critical path, where the maximum delay is given by three exclusive-OR gates. This critical path is smaller than that given by equations (4), (5) and (6).

The final sum result of the 4-2 compressor is given by a combination of *Sum*, *Cout* and *Carry* terms, where the term *Sum* has weight (*j*) and *Cout* and *Carry* terms have weight (*j*+1), as can be seen in equation (7).

$$S = Sum + 2(Cout + Carry) \quad (7)$$

To implement an *n*-bit wide 4:2 adder compressor it is needed a recombination of partial *Carry* and *Sum* terms, as can be seen in the example of a 16-bit 4:2 compressor shown in Fig. 3. As can be seen in this figure, the recombination of the *Carry* and *Sum* is obtained by using a cascade of half adder (HA) and full adders (FA) circuits in a Ripple Carry form. This line represents a bottleneck in the compressor performance, since the partial carries produced by the actual block is propagated to the next blocks. In this work we intend to speed-up this structure by using more efficient parallel-prefix adders in these lines.

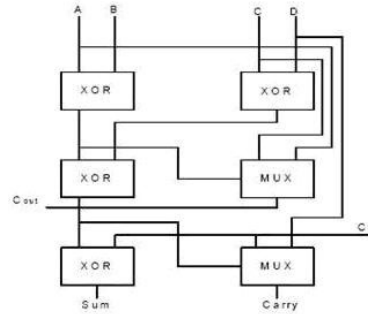


Fig. 2 – 4-2 compressor with XOR and multiplexer [3]

The Fig. 3 shows a 16-bit 4-2 adder compressor, where we can observe the line of ripple carry adder. The focus of this work is to replace that line by efficient adders such as parallel-prefix adders. The Figs. 4 and 5 showing 1-bit 8-2 and 16-2 adder compressor expansion, respectively.

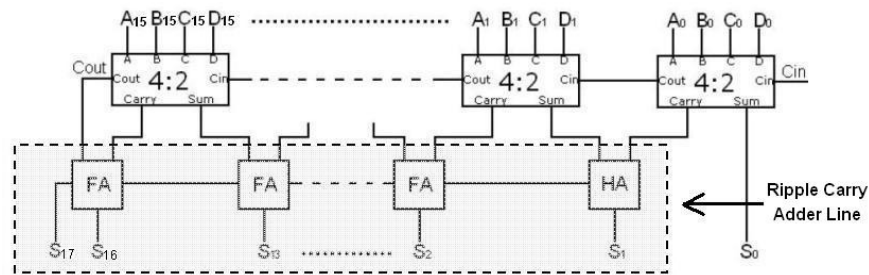


Fig. 3 – 16-bit 4-2 adder compressor structure

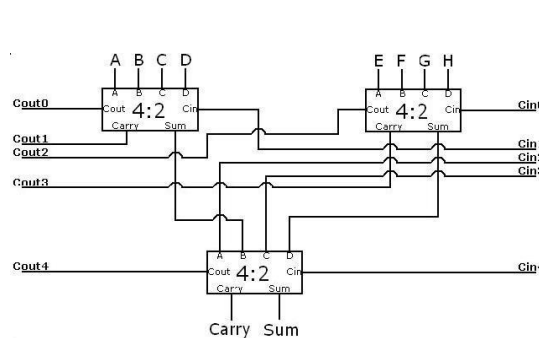


Fig. 4 – Block diagram of 8-2 adder compressor

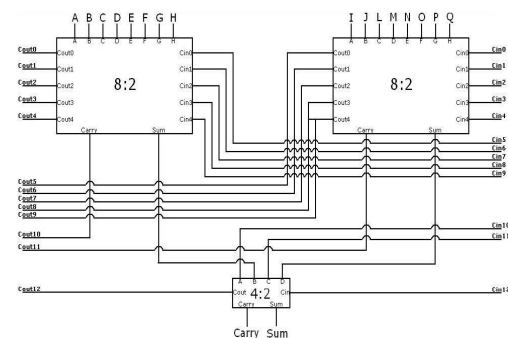


Fig. 5. Block diagram of 16-2 adder compressor

4. Experimental Results

In this section area, delay and power consumption results from 4-2, 8-2 and 16-2 16 bit wide adder compressors are presented in Tab. 1 with various efficient adders utilized in line adders. The parallel-prefix adders and adder compressors were all described in BLIF (Berkley Logic Interchange Format), which represents a textual description of the circuits at gate level. Area and delay values were obtained in SIS (Sequential Interactive Synthesis) environment [14]. Power consumption was obtained in SLS (Switch Level Simulator) tool [15]. Area is given in terms of number of literals, where each literal is approximately two transistors. Delay values are calculated from the model of general delay in SIS library. Power consumption values are obtained in

SLS tool after converting circuits from BLIF to SLS format. For the power estimation, 10.000 points of a random vector were used. Power Delay Product (PDP) metric is also presented for the compressors.

5. Conclusions

In this work area, delay and power results for a set of efficient parallel-prefix adders were studied. These adders were all applied to conventional 4-2 and extended versions of 8-2 and 16-2 compressors. The main results showed that the compressors are much more efficient when using parallel-prefix adders. In terms of power consumption the compressors with RCA are the best way. However, the PDP metric points to the efficiency of the compressors with parallel-prefix adder due to the significant delay reduction. As future work we intend to study other efficient adders presented in literature and implement these adders in transistor level in order to explore low power aspect of the compressors.

Tab. I - Area, delay, power estimates and PDP metric of 16-bit 4-2, 8-2 and 16-2 adder compressors

Adder Line	Area (Lits)			Delay (ns)			Power (mW)			PDP		
	4-2	8-2	16-2	4-2	8-2	16-2	4-2	8-2	16-2	4-2	8-2	16-2
Ripple Carry	600	1396	2932	43.88	54.19	57.89	10.90	45.76	128.62	478,50	2480.2	7446.1
Brent – Kung	659	1471	3007	25.66	30.98	34.76	14.50	57.23	162.86	372,21	1773.1	5661.2
Sklansky	679	1495	3031	22.22	27.60	31.38	15.02	57.29	156.05	333,92	1581.2	4897.0
Harris	702	1516	3052	19.87	25.13	29.32	14.55	58.61	159.14	289,18	1472.9	4666.2
Beaumont – Smith	729	1543	3079	19.82	27.69	32.08	15.10	61.06	165.33	299,33	1690.8	5303.8
Han - Carlson	702	1516	3052	18.15	25.06	29.11	14.52	57.85	155.70	263,62	1449.7	4532.5
Ladner - Fischer	704	1506	3042	17.60	25.48	29.74	14.50	58.22	157.86	255,36	1483.4	4694.7
Knowles (4, 2, 1, 1)	758	1632	3168	17.60	24.13	28.16	15.63	64.34	169.15	275,11	1552.6	4763.2
Knowles (2, 2, 2, 1)	800	1590	3126	17.46	23.00	27.73	16.42	62.36	166.83	339,16	1434.4	4626.3
Kogge – Stone	804	1636	3172	15.92	23.65	27.72	16.52	64.36	169.11	263,11	1522.2	4687.7
Ling Adders (KS)	836	1668	3204	15.93	22.86	27.72	17.35	69.17	187.53	276,40	1581.4	5198.4

6. References

- [1] B. Parhami, "Computer Arithmetic - Algorithms and Hardware Designs," Oxford University Press, 2000.
- [2] Dimitrakopoulos, Nikolos, "High-Speed Parallel-Prefix VLSI Ling Adders," IEEE 2005.
- [3] V. Oklobdzija, D. Villeger, S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers and Alghorimic Approach," IEEE Transaction on Computers, Vol.45, N_3, 1996.
- [4] R. P. Brent, H. T. Kung, "A regular Layout for Parallel Adders", IEEE, 1982.
- [5] J. Sklansky, "Conditional-Sum Addition Logic," IRE transactions on computers, 1960.
- [6] P. M. Kogge, H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence equations," IEEE, 1973.
- [7] T. Han, D. A. Carlson, "Fast Area-Efficient VLSI Adders", IEEE, 1987.
- [8] R. E. Ladner, M. J. Fischer, "Parallel Prefix Computation," ACM, 1980.
- [9] S. Knowles, "A Family of adders", IEEE, 2001
- [10] A. Beaumont-Smith, C. Lim, "Parallel Prefix Adder Design," IEEE, 2001.
- [11] D. Harris, "A Taxonomy of Parallel Prefix Networks," IEEE, 2003.
- [12] A. Weinberger, "4-2 Carry-Save Adder Module," IBM Technical Disclosure Bulletin, 1981.
- [13] A. M. Silva, "Técnicas para a Redução do Consumo de Potência e Aumento de Desempenho de Arquiteturas Dedicadas aos Módulos do Padrão H.264/AVC de Codificação de Vídeo," Technical Report, University Catholic of Pelotas, 2008.
- [14] E. Sentovich. K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton and A.L. Sangiovanni-Vincentelli., "SIS: a system for a sequential circuits synthesis," Berkeley University of California 1992.
- [15] A. Genderem, "SLS an efficient switch timing simulator using min – max voltage waveform," International Conference of VLSI 1989 New York.

Techniques for Optimization of Dedicated FIR Filter Architectures

Mônica L. Matzenauer, Jônatas M. Roschild, Leandro Z. Pieper, Diego P. Jaccottet,
Eduardo A. C. da Costa, Sergio J. M. de Almeida.

monicalorea@gmail.com, jonatasroschild@bol.com.br, leandrozaf@pop.com.br,
diego_porto_j@yahoo.com, {ecosta, smelo}@ucpel.tche.br

Universidade Católica de Pelotas – UCPEL

Abstract

This work proposes the optimization of FIR (Finite Impulse Response) filters by using efficient radix-2^m array multiplier and coefficient ordering technique. The efficient array multiplier, used in the Fully-Sequential FIR filter architecture, reduces the number of partial product lines by multiplying groups of m-bit simultaneously. In this work we have used m=4 array multiplier that performs radix-16 multiplication in 2's complement. We have also used an algorithm that choice the best ordering of the coefficients in the FIR filter with the purpose of reducing the switching activities in the buses by minimizing the Hamming distance between the consecutive coefficients. As will be shown, the use of the efficient array multiplier with an appropriate ordering of the coefficients can contribute for a significant reduction of power consumption of the FIR architectures.

1. Introduction

For a given transfer function, there are two main aspects to be considered when designing a hardwired parallel filter which are the number of discrete coefficients and the number of bits. The first one determines the amount of multipliers and adders to be implemented and the later determines the word length of the entire datapath. In this case, the most expensive operator in terms of area, delay, and power in a FIR filter operation is the multiplier.

In this work, FIR filtering is addressed by the implementation of Fully-Sequential dedicated hardware, where the main goal is the power reduction by applying the efficient array multiplier of [1], whose structure is composed by less complex multiplication blocks and resort to efficient Carry Save adders (CSA) [2], and coefficient ordering technique [3], [4], [5].

In [5] the use of the coefficient ordering technique was limited to an 8-tap 16-bit wide Fully-Sequential FIR architecture. In this work, we have extended-up the obtained results by proposing a new algorithm for the use of the coefficient ordering algorithm in FIR filters with any number of coefficients and any number of bits. For this work, this new algorithm is applied to 16-tap and 32-bit FIR architectures. Moreover, we have applied to the FIR architectures the array multiplier of [1] which is more efficient than that used in [5]. The main results show the great impact on reducing power consumption in higher FIR filter architectures when using coefficient ordering technique. As will be shown, it is possible to reduce almost 70% power consumption in 32-bit 16-tap FIR architecture.

2. FIR Filter Realization

The operation of FIR filtering is performed by convolving the input data samples with the desired unit impulse response of the filter. The output $y(n)$ of an N -tap FIR filter is given by the weighted sum of the latest N input data samples $x(n)$ as shown in Equation 1.

$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i) \quad (1)$$

In this work we address the problem of reducing the glitching activity in FIR filters by implementing an alternative Fully-Sequential architectures as shown in fig. 1 for an 8th order example [5]. As the same form as in [5], we have implemented a pipelined version of the Fully-Sequential FIR filter. The pipelined form is implemented by inserting a register between the outputs of the multiplier and the inputs of the adder. This register can be seen in the dotted lines of fig. 1. In fact, this register is responsible for reducing the great amount of glitching activity that is produced by the multiplier circuit. We have omitted the control block for the simplification of fig. 1. We will present results for 16 and 32-bit wide versions of 8th and 16th order of Fully-Sequential FIR filter architectures.

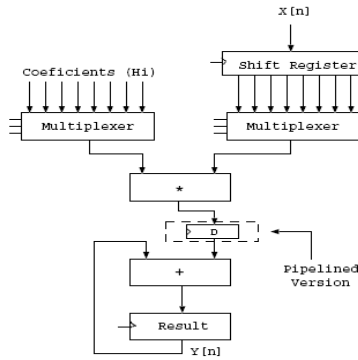
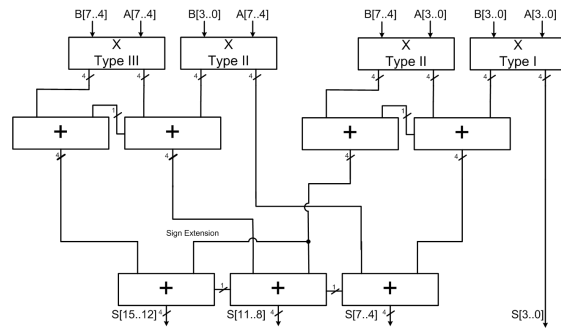


Fig. 1 – Example of a Fully-Sequential FIR filter

Fig. 2 – 8 bit wide binary array multiplier architecture ($m=4$).

3. Related Work on FIR Filter Optimization

When power reduction in multipliers has taken into account, Booth multiplier has been the primary choice [6], [7]. However, in [1] it was shown that the radix-2^m array multiplier is more efficient than the Booth multiplier. Thus, we are using this array multiplier in the implemented FIR architectures.

As the output of a FIR operation is performed by a summation of a data-coefficient product, some techniques have addressed the use of coefficient manipulation in order to reduce the switching activity in the multipliers input. Coefficient Ordering, Selective Coefficient Negation and Coefficient Scaling [3], [4] are some of these techniques, whose the main goal is to minimize the *Hamming* distance between consecutive coefficients to reduce power consumption in the multiplier input and data bus. In [5] an extension of the Coefficient Ordering technique is applied to FIR architectures. However, the results are limited to 8th order and 16-bit architectures. In this work we extend-up the use of this technique by applying it to 16th order and 32-bit architectures.

4. Efficient Multiplier and Coefficient Ordering

In this section we summarize the main aspects of the array multiplier of [1]. The limitations of the algorithm of [5] and the new algorithm for the coefficient ordering are also presented in this section.

4.1. Radix-2^m Array Multiplier

For the operation of a radix-2^m multiplication W -bit values, the operands are split into groups of m bits. Each of these groups can be seen as representing a digit in a radix-2^m. Hence, the radix-2^m multiplier architecture follows the basic multiplication operation of numbers represented in radix-2^m. The radix-2^m operation in 2's complement representation is given by Equation 2 [8].

$$A \times B = A' \times B' - A' b_{W-1} b_{\frac{W}{m}-1} 2^{W-m} - a_{W-1} a_{\frac{W}{m}-1} \sum_{j=0}^{\frac{W}{m}-1} b_j 2^{W-m+j} \quad (2)$$

For the architecture of the Equation 2, three types of modules are needed. Type I are the unsigned modules. Type II modules handle the m -bit partial product of an unsigned value with a 2's complement value. Finally, we have Type III modules that operate on two signed values. Only one Type III module is required for any type of multiplier, whereas $(2^{\frac{W}{m}} - 2)$ Type II modules and $(\frac{W}{m} - 1)^2$ Type I modules are needed. In the work of [1] it is proposed new Type I, Type II and Type III dedicated blocks for radix-2^m multiplication. These dedicated blocks are applied to W -bit wide array multipliers. The number of partial product lines for the architectures is given by: $(\frac{W}{m} - 1)$ [8].

4.2. Coefficient Ordering Algorithm

The technique of coefficient ordering proposed in [4] assumes that the coefficients in the architecture of a sequential FIR filter can be combined to reduce the switching activity at the input of the multiplier circuit. This is due to the fact that the addition operation in this type of filter algorithm is commutative and associative and thus, the output of the filter is independent of the order of the coefficient processing product. For a 4th order FIR filter, for example, the output can be computed according to Equation 3 or Equation 4.

$$Y_n = A_0 X_n + A_1 X_{n-1} + A_2 X_{n-2} + A_3 X_{n-3} \quad (3)$$

$$Y_n = A_1 X_{n-1} + A_3 X_{n-3} + A_0 X_n + A_2 X_{n-2} \quad (4)$$

In the algorithm of [5] the cost function is calculated for all the combinations for the coefficients. For the 8th order FIR filter used as example in [5], this exhaustive algorithm is still reasonable. However, for a higher number of coefficients this algorithm is less attractive due to the time necessary to process the large number of combinations. Thus, in this paper we propose a new algorithm that makes possible calculate the cost function for N coefficients and W -bits in sequential FIR filter architectures.

The difference between this new algorithm and the algorithm of [5] is that while the previous algorithm tests all possible combinations, the new one uses heuristic to find the best ordering of the coefficients. This aspect enables the calculation of the cost function for any coefficients with any bit-width in a faster way. We have observed that the new algorithm is around 6 to 8 times faster than the one presented in [5]. As an example, while the new proposed algorithm takes less than 5 minutes for the ordering of 16 taps, the previous one takes almost 30 minutes for the same task.

5. FIR Filter Results

This section presents area, delay and power results for 16 and 32 bit FIR filters with 8 and 16 coefficients. The coefficients were obtained in Matlab tool by using Remez algorithm. In the tests were used bandpass FIR filters, where 0.15 is the passband and 0.25 is the stopband frequencies.

The circuits were all implemented in BLIF (*Berkeley Language Interchange Format*), which represents a textual description of the circuit at the logic level. Area and delay values are obtained in SIS (*Sequential Interactive Synthesis*) environment [9]. Power consumption is obtained in SLS (*Switch Level Simulator*) [10] tool. Area is given in terms of number of literals, where each literal is approximately two transistors. Delay values were obtained from the model of general delay in SIS library. For the power consumption estimation it is done a conversion from BLIF to the transistor level format of SLS tool. The power results were obtained after using random type vectors at the filters input.

5.1. FIR Filter Results Using Efficient Multipliers

As can be observed in tab. 1, although the FIR filter with Booth multiplier presents less area value, this architecture is more efficient when using the array multiplier of [1]. In particular, the filter with radix-16 array multiplier presents significant reduction in terms of power consumption. In fact, these results are justified for the less complexity presented by the dedicated multiplication blocks of the array multiplier of [1]. The radix-16 ($m=4$) for example, uses more simple radix-4 ($m=2$) dedicated multiplication blocks in its structure. This justifies the less power consumption presented by the filter with this multiplier. On the other hand, the radix-256 ($m=8$) dedicated multiplication block is composed by two radix-16 structures and reduces more the number of partial product lines enabling significant delay reduction in FIR architecture, when compared against the structure with Booth multiplier, as can be observed in tab. 1.

Tab. 1- Results of the sequential FIR filter architecture with optimized array multipliers.

	Booth	$m=2$	$m=4$	$m=8$
Area (literal)	6394	7288	7788	8458
Delay (ns)	105.74	96.55	89.99	88.47
Power (mW)	573.90	331.24	270.74	317.74

5.2. FIR Filter Results Using Coefficient Ordering

The manipulation coefficient technique that has been applied to the Fully-Sequential architecture shows that the correlation between the coefficients can reduce the switching activity in the multipliers input. For a higher number of coefficients we can have a higher opportunity for saving power by the manipulation of coefficients. As can be observed in tab. 2, almost 70% of power reduction can be achievable for a 16th order and 32-bit architecture. In fact, as could be observed in [1] there is a relationship between the higher complexity of the dedicated multiplication blocks and the less number of partial product lines presented in the array structures of the multiplier. Thus, the techniques presented by [1] are more efficient when applied to multipliers with a higher number of bits. Thus, the combination of the coefficient ordering algorithm with the use of the efficient 32-bit array multiplier enables a significant power reduction in the FIR filter architecture as shown in tab. 2.

Another important aspect to be observed in the results of tab. 2 is the power reduction presented by the filters with the radix-16 multiplier of [1], when compared against the filters with the multipliers of [8]. In fact, this is explained by the fact that the multiplier of [1] is an improvement of the multiplier of [8].

Tab.2 – Results of the FIR filters with radix-16 ($m = 4$) array multipliers

	8 taps – 16 bits		16 taps – 16 bits		16 taps – 32 bits	
	Original coefficients	Ordered coefficients	Original coefficients	Ordered coefficients	Original coefficients	Ordered coefficients
POWER(mW) Optimized multiplier of [1]	127.31	95.39	153.29	127.37	1129.39	354.00
POWER REDUCTION	25.07%		16.91%		68.65%	
POWER(mW) original multiplier of [8]	144.49	114.19	195.89	158.32	1233.26	451.50
POWER REDUCTION	26.53%		19.17%		63.38%	

6. Conclusions

This work presented low power techniques for the optimization of FIR filter architectures. Efficient radix- 2^m array multiplier was used in the FIR architectures and it was possible to observe that the filter architectures that use the radix-16 array multiplier are more efficient when compared against state of the art Booth multiplier. In this paper it was also presented an algorithm for the calculation of the best ordering of the coefficients in order to reduce the switching activity in the data buses. The results showed that a combination of the use of coefficient ordering algorithm in a higher number of taps and the use of the efficient array multiplier in a higher number of bits has a great impact on power reduction in FIR architectures. As future work we intend to extend-up our algorithm for the calculation of the cost function for Semi-Parallel FIR architectures.

7. References

- [1] Pieper, L.; Costa, E.; Almeida, S.; Bampi, S.; Monteiro, J. "Efficient Dedicated Multiplication Blocks for 2's Complement Radix-16 and Radix-256 Array Multipliers". In: 2nd International Conference on Signals, Circuits and Systems, 2008, Hammamet. Proceedings of the 2nd SCS 2008, 2008.
- [2] Sohn, A. "Computer Architecture – Introduction and Integer Addition". Computer Science Department – New Jersey Institute of Technology, 2004.
- [3] Mehendale, M.; Sherlekar, S.; Venkatesh, G. "Algorithmic and Architectural Transformations for Low-Power Realization of FIR Filters". In Eleventh International Conference on VLSI Design, pages 12-17, 1998.
- [4] Mehendale, M.; Sherlekar, S.; Venkatesh, G. "Techniques for Low Power Realization of FIR Filters". In Design Automation Conference, DAC, 3(3), pages 404-416, September, 1995.
- [5] Costa, E.; Monteiro, J.; Bampi, S. "Gray Encoded Arithmetic Operators Applied to FFT and FIR Dedicated Datapaths". VLSI-SoC, 2003.
- [6] Booth, A. "A Signed Binary Multiplication Technique. J. Mech. And Applied Mathematics", (4): 236-240, 1951
- [7] Gallagher, W.; Swartzlander, E. "High Radix Booth Multipliers Using Reduced Area Adder Trees". In Twenty-Eighth Asilomar Conference on Signals, Systems and Computers, volume 1, pages 545-549, 1994.
- [8] Costa, E.; Monteiro, J.; Bampi, S. "A New Architecture for Signed Radix-2m Pure Array Multiplier". IEEE International Conference on Circuit Design, ICCD, September 2002.
- [9] Sentovich, E. et. al. "SIS: a System for a Sequential Circuits Synthesis". Berkeley University of California 1992.
- [10] Genderen, A. "SLS an Efficient Switch Timing Simulator Using min – max Voltage Waveform". International Conference of VLSI 1989 New York.

Reconfigurable Digital Filter Design

¹Anderson da Silva Zilke, ²Júlio C. B. Mattos
aszielke@gmail.com, julius@ufpel.edu.br

¹Universidade Luterana do Brasil
Curso de Ciência da Computação - Campus Gravataí
²Universidade Federal de Pelotas
Grupo de Arquitetura e Circuitos Integrados

Abstract

Digital Signal Processing is present in several areas. This processing can be done by general purpose processors, DSP processors or application-specific integrated circuits. The reconfigurable circuit can combine the high performance of DSP architectures and the reconfiguration flexibility. This paper presents different reconfigurable architectures for FIR and IIR digital filters. The proposed architectures are described in VHDL hardware description language and prototyped in FPGA. This paper shows the implementation results for the different proposed architectures.

1. Introduction

Nowadays, the electronic systems market does not stop growing, and products with digital signal processing capabilities are more necessary. These systems are everywhere, for example, mobile phones, video recorders, CD players, modems, TVs, medical equipments and so on.

Signal is a representation of analog physical quantities during some period of time. These signals are in the every day life. During the past, most part of signal processing in the equipments was done by analog circuits. After 80's, the digital signal processing technology increased mainly by the introduction of digital circuits with more processing capacity [1].

Digital Signal Processing (DSP) is concerned with the representation of the signals by a sequence of numbers or symbols and the processing of these signals using mathematics, algorithms, and techniques to manipulate these signals after they have been converted into a digital form [1]. In most cases, these signals originate as sensory data from the real world: seismic vibrations, visual images, sound waves, etc. The continuous real-world signals are analog. Then the first step is usually to convert the signal from an analog to a digital form, by using an analog to digital converter (A/D converter). Often, the required output signal is another analog signal, which requires a digital to analog converter (D/A converter). Digital filter is widely used in Digital Signal Processing. This digital filter is a system that performs mathematical operations to reduce or to enhance certain aspects of that signal.

The Digital Signal Processing can be done using standard processors, application-specific integrated circuit (ASICs) or on specialized processors called digital signal processors (DSPs) [2]. The DSP processors have specific features to increase the performance when running DSP applications. However, the ASICs have the best results, when compared with the processors, in terms of performance and power dissipation. These systems can be designed using some hardware description language.

The reconfigurable systems can be defined as hardware circuits that can change their function during the execution. These circuits combine the flexibility of software with the high performance of hardware by processing with very flexible high speed computing.

This work proposes three new reconfigurable architectures for digital filters. These different architectures were proposed to implement: FIR filters, IIR filters and finally one architecture that can implement both FIR and IIR filters. These architectures aim to combine the flexibility of reconfigurable systems with the performance of dedicated hardware.

This paper is organized as follows. Section 2 explains the Digital Filters. Section 3 presents the details of proposed reconfigurable digital filter architectures. Section 4 presents the results obtained from the different architectures. Finally, section 5 presents the conclusions and discusses future work.

2. Digital Filters

Digital filters are a very important part of Digital Signal Processing. In fact, their good performance is one of the key reasons that DSP has become so popular [1]. Digital filter is a system that performs mathematical operations on a sampled, discrete-time signal to signal separation or signal restoration. Signal separation is when a signal has been contaminated with interference, noise, or other signals. On the other hand, signal restoration is used when a signal has been distorted in some way. Analog filters can be used for these same

tasks; however, digital filters can achieve far superior results [1]. However, digital filters have lot of limitations like: latency and more limited in bandwidth. The digital filters can be used in several types of applications.

In DSP systems, filter's input and output signals are in the time domain. This signal is sampled at a fixed period of time. To convert the signal from an analog to a digital form, an A/D converter is applied. This signal is digitized and represented as a sequence of numbers, then manipulated mathematically, usually based on the Fast Fourier Transform (FFT), a mathematical algorithm that quickly extracts the frequency spectrum of a signal. Finally, the signal is reconstructed as a new analog signal by a D/A converter.

The digital filter design process is a complex topic [3]. The design task is based on the choice of filter and the coefficient definition according the filter. Digital filters are classified into two basic forms:

- Finite Impulse Response (FIR) filters: implements a digital filter by convolving the input signal with the digital filter's impulse response. In this filter, the outputs depend on the present input and a number of past inputs;
- Infinite Impulse Response (IIR) filters: these filters are called recursive filters and use the previously calculated values from the output, besides points from the input. The impulse responses of this recursive filters are considered infinite.

3. Proposed Reconfigurable Digital Filter Architectures

The aim of this work is to propose a digital filter architecture that can be reconfigurable according to the application requirements (different types of filters and its configuration). Thus, one can use the same hardware structure to different applications, combining hardware reuse and good performance. For example, one can run different applications in the same equipment requiring different digital filter configurations (different number of taps and coefficient values).

Three different reconfigurable architectures were proposed to implement: FIR filters, IIR filters and both FIR and IIR filters. The implemented architectures were described in VHDL and prototyped in FPGA.

The first architecture (to implement FIR filter) can be seen at fig. 1. This architecture can implement a FIR filter until 7 taps. It was introduced one multiplexer (to select the output) and some registers (to load the coefficient values of the different filters that can be implemented).

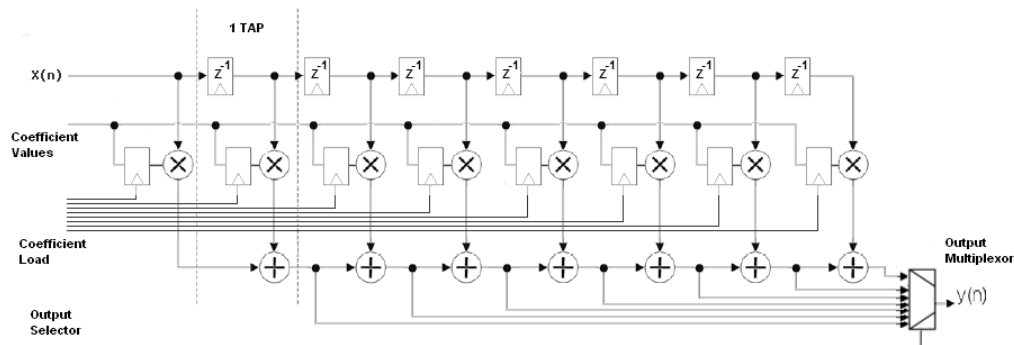


Fig. 1 – Reconfigurable FIR Filter Architecture.

Fig. 2 shows the datapath of reconfigurable FIR filter architecture. This architecture has two memories: the coefficient memory stores the coefficient values of the different filters that can be implement at this architecture and the reconfigurable memory stores the reconfiguration options. The value stored into reconfiguration memory provides the address to access the coefficient values (stored into coefficient values) and the number of taps. Moreover, there is one interrupt input to select which filter will be executed.

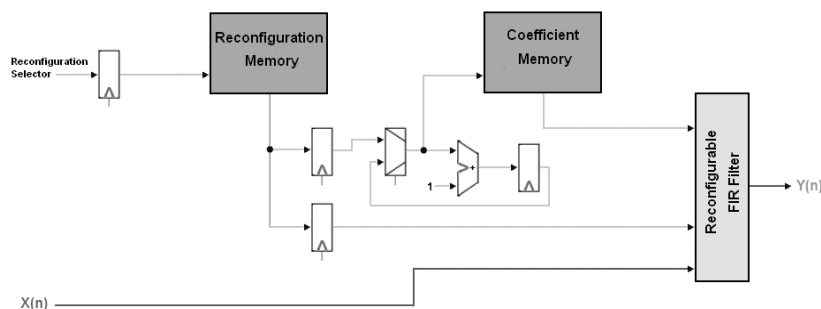


Fig. 2 – Reconfigurable FIR Architecture Datapath.

The design and execution process of the reconfigurable filter is based on the following steps:

- Determine how many filters are necessary and design this filters (design time);
- Based on the maximum number of taps, generate the reconfigurable (design time);
- Load the values at the reconfiguration memory (design time);
- Load the coefficient values at the coefficient memory (design time);
- The application selects the right filter using the interrupt input (run time);
- The reconfigurable filter loads the coefficient values to the registers and starts to filter (run time);
- The architecture processes normally (filtering) waiting the interrupt request.

Fig. 3 illustrates the reconfigurable FIR-IIR filter architecture. This architecture can implement either FIR or IIR filters. This architecture has two structures (A and B) that are necessary when implementing IIR filters (it is recursive). The reconfigurable IIR filter architecture is not shown at this paper because it is very similar to the FIR-IIR architecture. In the mixed architecture (depicted in the fig. 3) is included one more multiplexer (rightmost multiplexer at the figure). Moreover, the reconfigurable memory was changed to store the type of the filter (FIR or IIR) and there are some changes at the control part and datapath. The control part and datapath of the FIR-IIR architecture is showed in the fig. 4.

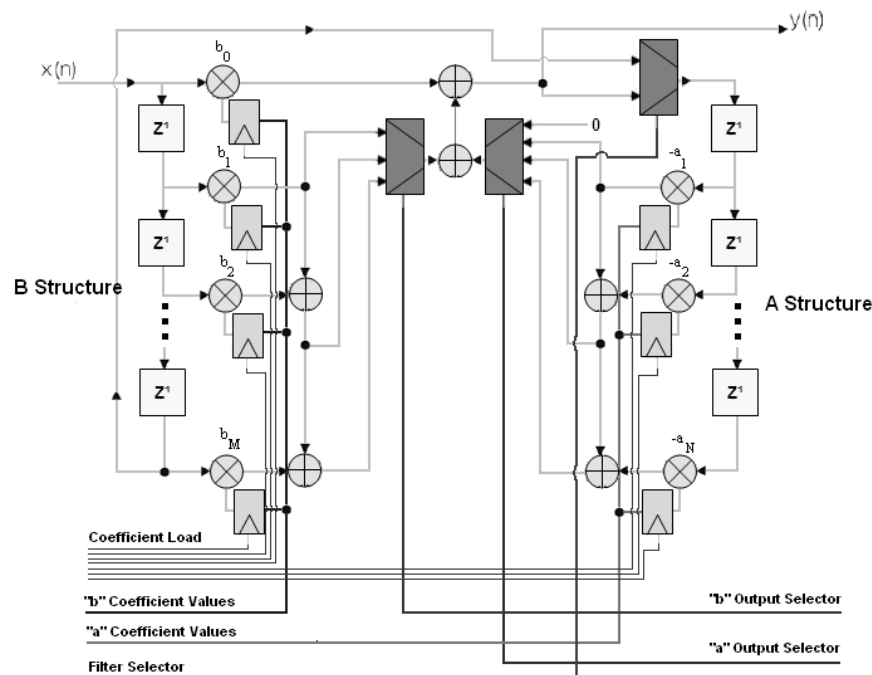


Fig. 3 – Reconfigurable FIR-IIR Architecture.

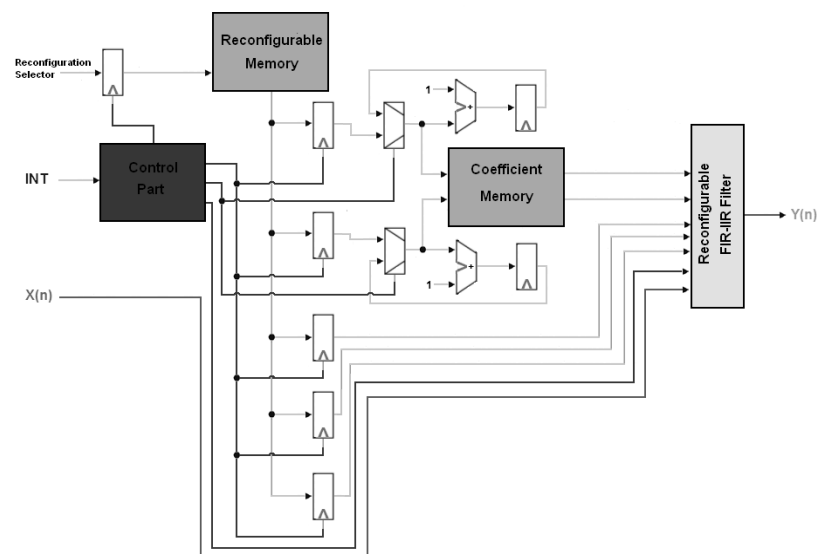


Fig. 4 – Reconfigurable FIR-IIR Architecture (with Control Part and Datapath).

4. Results

To synthesize and to simulate the implemented architecture, we used the software ISE 9.2i [4] as well as the SPARTAN-3 FPGAs [5], both from Xilinx. This kind of FPGAs presents the characteristic of having internal memory to the device. In order to use these memories, the Core Generator Tool [6] was used. The filter design was done using Matlab Tool [7].

In this section, the results obtained from the synthesis of the reconfigurable architectures are presented. These results are based on the FPGA used area (number of slices, flip-flops and *Look up Tables*) and the maximum clock frequency provided by the ISE 9.2i software. The architecture synthesis was done for the chip EPF XC3S200-FT256, family SPARTAN-3 from Xilinx.

Tab. 1 presents the synthesis results using 10 taps filter to the FIR and IIR architectures. The FIR-IIR architecture can implement a 10 taps IIR filter and 20 taps FIR filter. All filters are 8 bit. The IIR architecture uses about double FPGA resources than FIR architecture because its recursive architecture. However, the mixed architecture (that can implement FIR or IIR architectures) uses a few resources than IIR architecture it is the IIR architecture with some modifications. The maximum clock provided by the software was 44.280 MHz in the mixed architecture.

Tab. 1 – Synthesis Results of Reconfigurable Architectures.

FPGA Resources	FIR	IIR	FIR-IIR
Slices	574	1177	1301
Flip-Flops	1011	2248	2493
LUTs	245	436	473
Maximum Frequency	152.792 MHz	45.491 MHz	44.280 MHz

5. Conclusions

This paper presents the implementation of three reconfigurable digital filter architectures: finite impulse response (FIR), infinite impulse response (IIR) and a reconfigurable architecture that can implement both FIR and IIR filters. The third architecture combines a good performance with the flexibility, making possible the implementation of different digital filter types in the same architecture.

Regarding future work, there is the need to study alternatives to increase performance of the architectures. We also have to compare this architecture with other architectures proposed in the research community.

6. References

- [1] Steven W Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Pub, 1997. Available at: <<http://www.dspguide.com>>.
- [2] Phil Lapsley, Jeff Bier, Amit Shoham, Edward A. Lee. *DSP Processor Fundamentals: Architectures and Feature*, Wiley-IEEE Press, 1997.
- [3] Paulo S. R. Diniz, Eduardo A. B. da Silva and Sergio L. Netto. *Digital Signal Processing - System Analysis and Design*, Cambridge, 2002.
- [4] Xilinx. Xilinx ISE 9.2i Software Manuals and Help. Available at: <http://www.xilinx.com/support/sw_manuals/xilinx7/index.htm>.
- [5] Xilinx. SPARTAN-3 FPGA Family: Complete Data Sheet. 2006. Available at: <<http://direct.xilinx.com/bvdocs/publications/ds099.pdf>>.
- [6] Xilinx. CORE Generator System. Available at: <http://www.xilinx.com/products/design_tools/logic_design/design_entry/coregenerator.htm>.
- [7] The Mathworks. MATLAB - The Language of Technical Computing. Available at: <<http://www.mathworks.com/products/matlab/>>.

ATmega64 IP Core with DSP features

¹Eliano Rodrigo de O. Almança, ²Júlio C. B. Mattos
ealmanca@gmail.com, julius@ufpel.edu.br

¹Universidade Luterana do Brasil
Curso de Ciência da Computação - Campus Gravataí
²Universidade Federal de Pelotas
Grupo de Arquitetura e Circuitos Integrados

Abstract

Nowadays, the Intellectual Property market and development do not stop growing, mainly in digital circuits. There are software and hardware IP components designed to be used as a module in larger projects. This paper presents the implementation of an ATmega64 microcontroller core suited to digital signal processing applications. This core was developed in VHDL hardware description language and prototyped in FPGA. In this paper, implementation results are presented, emphasizing on the area occupied and the frequency.

1. Introduction

In the past decades, the development and market of IP (Intellectual Property) has grown significantly [1][2]. The IPs components are software or hardware that can be designed to be sold as a module that can be used in several larger projects. The IP design is an incentive for innovation and creativity, where small businesses can develop products to compete in the global market.

There are several kinds of IP blocks, from simple blocks, such as a serial controller to complex processor cores. These processor cores can be implemented in different architectures: CISC (Complex Instruction Set Computer) or RISC (Reduced Instruction Set Computer). The RISC architecture, also known as load / store, is based on a reduced instruction set and usually has better performance than the CISC ones.

Nowadays, the microcontrollers are widely used in several applications, mainly in industries and embedded systems, due to its advantages such as low cost and low energy consumption. They are used in dedicated systems in general like cars, appliances, industrial automation and others. There are several microcontrollers available in the market with different architectures, number of bits, and so on. The Atmel ATmega64 [3] is a good performance and cost microcontroller based on the RISC architecture.

On the other hand, there are a lot of applications using DSP (digital signal processing), because these signals are present in our daily life: music, video, voice, etc. The digital signal processing aims to extract some signal information. This retrieval of information is performed by mathematical operations. There are processors designed to make efficient the digital signal processing: the DSP processors. These processors have mechanisms to improve the performance of these applications.

At the present time, one can prototype a circuit in an easy way. The FPGAs (Field Programmable Gate Arrays) offer the possibility of fast prototyping to the final user level with a reduced design cycle, making them an interesting option in system designs [4]. For the digital circuit prototyping in FPGA, the VHDL hardware description language is widely used. The aim of this work is design an ATmega64 IP core with DSP features. In this work, the VHDL and FPGA are used to describe and prototype the IP core.

This paper is organized as follows. Section 2 explains the main features of the ATmega64 microcontroller. Section 3 presents the details of ATmega64 IP core with DSP features implementation. Section 4 presents the results obtained from the ATmega 64 architecture. Finally, section 5 concludes this work and discusses future work.

2. ATmega64 Microcontroller

The Atmel ATmega64 [3] microcontroller is a RISC processor (Reduced Instruction Set Computer) with limited set of instructions and fixed instruction format, allowing a quick and easy decoding. The instructions in a RISC machine are directly executed on hardware without an interpreter (microcode). This fact introduces better performance in RISC machines [5].

The ATmega64 microcontroller has advantages as a low cost, low energy consumption as well as good performance. This microcontroller has the following main features [3]:

- 8-bit RISC architecture;
- set of 130 instructions, and the most run in a single clock cycle (2 or 4 bytes instruction size);

- set of 32 general purpose working registers;
- the microcontroller uses a Harvard architecture – with separated memories and buses for program and data;
- 8-bit bi-directional I/O port named A to G.

In a typical ALU (arithmetic and logic unit) operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle. The ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate value are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction. Program memory space is divided in two sections, the Boot program section and the Application program section.

An interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a distinct Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower Interrupt Vector address, the higher the priority.

3. ATmega64 Implementation

The implemented ATmega64 IP core was described in VHDL, having as its target its prototyping in FPGA. A VHDL data-flow and structural description was developed. Basically, The ATmega64 architecture VHDL description is divided into two main parts using the concept of RT (register-transfer) design [6]: control part and datapath. The architecture diagram is present at fig. 1:

- Control – Part that implements the Finite State Machine (FSM) of the Architecture, being responsible for the architecture rhythm;
- Datapath – It executes the architecture actions: memory access, PC incrementation, access to the arithmetical units, addressing mode and others.

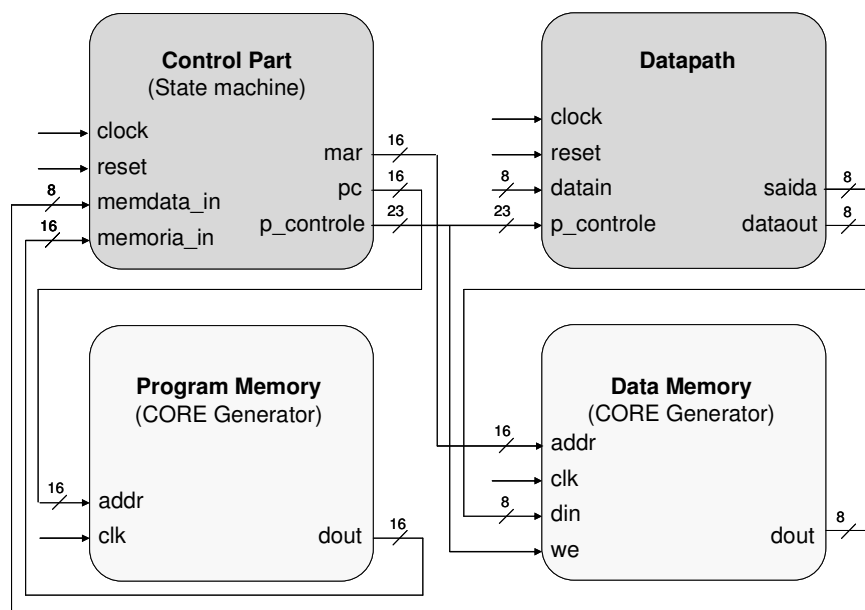


Fig. 1 – ATmega64 Architecture Diagram.

To synthesize and simulate the implemented architecture, we used the software ISE 9.2i [7] as well as the SPARTAN-3 FPGAs [8], both from Xilinx. This kind of FPGAs presents the characteristic of having internal memory to the device. In order to use these memories, the Core Generator Tool [9] was used.

In this work, just a subset of the ATmega64 instruction set was implemented. To select a useful instruction subset, a FIR (finite impulse response) filter was implemented in C language. The FIR program was compiled and a subset of 23 instructions was selected to be implemented based on Assembly code. These instructions include arithmetic, logic, data transfer, flow control, input and output instructions and correspond to 17% of the total microcontroller instructions. Tab. 1 shows the selected instructions that were implemented. The Load Indirect (LD) instructions are implemented in nine different types (the table shows just one).

Tab. 1 – Implemented Instructions of ATmega64 microcontroller.

Mnemonics	Operands	Description
ADD	Rd, Rr	Add two Registers
ADC	Rd, Rr	Add with Carry two Registers
EOR	Rd, Rr	Exclusive OR Registers
MUL	Rd, Rr	Multiply Unsigned
RJMP	K	Relative Jump
CPC	Rd,Rr	Compare with Carry
CPI	Rd,K	Compare Register with Immediate
BRNE	K	Branch if Not Equal
MOV	Rd, Rr	Move Between Registers
LDI	Rd, K	Load Immediate
LD	Rd, X	Load Indirect (several types)**
LDS	Rd, k	Load Direct from RAM
STS	k, Rr	Store Direct to RAM
IN	Rd, P	In Port
OUT	P, Rr	Out Port

The format of the MUL instruction has changed to simplify its development because the output of the implemented ULA has just 8 bits. In the original format, the result is stored in two fixed registers (R1: R0), where the operation is performed as follow: $R1: R0 \leftarrow Rd \times Rr$. In the simplified format the result is stored in the Rd register, where the operation is performed as follow: $Rd \leftarrow Rd \times Rr$.

The datapath has the following main parts: the register file, the ALU (arithmetic and logic unit) and some multiplexers to select inputs and outputs operations from the input and output ports. The register file has 32 (8-bit) general purpose registers. The ALU implements the following operations: AND, OR, NOT, XOR, multiply, Adder and Subtractor. The ALU was also modified to execute multiply-accumulate instruction (MAC). One more opcode was included in the microcontroller and the MAC instruction can execute the multiply-accumulate in just one cycle, making the execution of DSP applications more efficient.

The datapath receives the control word generated by the control part. These control words are generated by the instruction decoding based on a hardwired process. The implemented architecture is multicycle. The main part of the control is a state machine, which is responsible for the sequence of data processing in the datapath.

4. Results

In this section, the results obtained from the synthesis of the ATmega64 IP core are presented. These results are based on the FPGA area occupied (number of slices, flip-flops and *Look up Tables*) and the maximum clock frequency provided by the ISE 9.2i software. The architecture synthesis was done for the chip EPF XC3S200-FT256, family Spartan-3 from Xilinx.

The VHDL description was simulated with several testbenchs running from small to large programs. The small programs was generated by hand and translated to COE format (for the memory). The FIR filter program was translated and adapted to COE format from the code produced by the compiler.

Tab. 1 presents the synthesis results of the ATmega64 IP Core with MAC instruction. These synthesis results are generated from a 1450 VHDL code lines with 13 different entities. The maximum clock provided by the software was 9.14 MHz. The architecture has a maximum frequency of 65.730MHz (Clock period of 15.214ns). Tab. 1 shows the area results in terms of the number of slices, the number of flip-flops and LUTs. In the implementation, the number of slices, flip-flops and LUTs used did not exceed 20%, allowing an increase in the number of instructions or other usage for the remaining device area (e.g., implement other DSP structures).

Tab. 1 – Synthesis Results of the ATmega64 IP Core.

Table 1: Synthesis Results of the P4-Mega-V4.1 Core.			
Clock Period		Maximum Frequency	
15.214ns		65.730MHz	
FPGA Area			
	Slices	Flip-Flops	LUTs
Used Components	393	360	621
Available Components	1920	3840	3840
Utilization (percentage)	20%	9%	16%

The most part of the used components (number of slices, flip-flops and LUTs) are used by the datapath (and inside the datapath, the most part of the components are used by multiplier). The multiplier was responsible by the increase of the clock period thus reducing the maximum frequency.

5. Conclusions

This paper presents the implementation of an ATmega64 microcontroller core suited to DSP (digital signal processing) applications. The microcontrollers are widely used today, has several advantages such as low cost and low energy consumption. However, these microcontrollers are not suitable to DSP applications. Thus, this work intends to provide an IP core that matches with low cost and energy plus a good performance on some DSP applications (the usage of a microcontroller with DSP features intends to solve just part of DSP applications - simple ones). Moreover, the engineers can use well known microcontroller architecture with DSP features.

Results in terms of FPGA area and frequency are presented in the paper. The IP area usage is too small making it suitable to be used in a larger project. For this architecture implementation was selected a few number of microcontroller instructions. Moreover, the implemented architecture is a multicycle one.

Regarding future work, there is the need to extend the implementation to support the full microcontroller instruction set. We also have to implement the pipeline version of the microcontroller as well the interrupt system. Analyzing the DSP features, it is necessary to increase the number of DSP structures (to extend the achieved DSP applications) and another important future work is the modifications that are necessary in the compiler (e.g. to generate the MAC instruction).

6. References

- [1] Rick Mosher. "Choosing Hardware IP", Available at: <http://www.embedded.com/columns/technicalinsights/178600378?_requestid=1116105>.
- [2] Brazil IP. Brazil IP Network. Available at: <<http://www.brazilip.org.br/>>.
- [3] Atmel. *Atmel 8-BIT AVR Microcontroller*, 2006. 395p.
- [4] K. Skahill, *VHDL for Programmable Logic*, Addison Wesley Longman, Menlo Park, 1996.
- [5] David A. Patterson; John L. Hennessy. *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 1996. 760p.
- [6] M. Mano; C. R Kine. *Logic and Computer Design Fundamentals*, Prentice Hall, 1997.
- [7] Xilinx. Xilinx ISE 9.2i Software Manuals and Help. Available at: <http://www.xilinx.com/support/sw_manuals/xilinx7/index.htm>.
- [8] Xilinx XILINX. SPARTAN-3 FPGA Family: Complete Data Sheet. 2006. Available at: <<http://direct.xilinx.com/bvdocs/publications/ds099.pdf>>.
- [9] Xilinx XILINX. CORE Generator System. Available at: <http://www.xilinx.com/products/design_tools/logic_design/design_entry/coregenerator.htm>.

Using an ASIC Flow for the Integration of a Wallace Tree Multiplier in the DataPath of a RISC Processor

¹Helen Franck, ¹Daniel S. Guimarães Jr, ²José L. Güntzel, ¹Sergio Bampi, ¹Ricardo Augusto da Luz Reis,

{hsfrank, dsgjunior, bampi, reis}@inf.ufrgs.br, guntzel@inf.ufsc.br

¹Grupo de Microeletrônica (GME) – INF – UFRGS

²Laboratório de Automação do Projeto de Sistemas (LAPS) – INE – UFSC

Abstract

This article describes the impact in terms of area, performance and power, resulting from the integration of a Wallace Tree multiplier in the datapath pipeline of a RISC processor. The project was carried out using a conventional flow based on EDA (Electronic Design Automation) tools from a behavioral description of the processor and a RT (Register Transfer) level description of the multiplier, both written in VHDL. The validation results showed that the inclusion of the multiplication in the processor datapath resulted in an increase of only 8.7% in terms of power dissipation and a reduction of 26% in terms of frequency of operation, which are reasonable prices for the implementation of the multiplication operation directly via hardware.

1. Introduction

Processors with RISC characteristics are present in most embedded systems, which are usually responsible for the tasks of supervision and/or implementation of non-critical parts of an embedded application. The low number of transistors and the energy efficiency of RISC processors are fruit of the close relationship between the set of instructions and organization of the components and make it the preferred architecture for integrated systems rather than the CISC model. In particular, the relative regularity shown by the datapath pipeline of RISC processors facilitates the development of descriptions at the RT (Register Transfer) level, so that today, it is possible to find descriptions of RISC processors in Verilog and VHDL language, many of these belonging to OpenSource projects.

Currently, the design of high performance processors and / or low power consumption, like those which compose portable electronic devices, like cell phones and notebooks, also requires high interference of the designers, because the most critical blocks need to be designed manually, using a full custom approach.

Furthermore, the design of processors for applications whose performance restrictions and / or energy consumption are not so severe can make use of EDA (Electronic Design Automation) tools in a conventional design flow, even if one or more blocks need to be separately synthesized and afterwards integrated.

Even taking into account the applications mentioned above and considering that the performance of arithmetic operations is a critical factor in achieving the project, the Wallace Tree multiplier was chosen to be integrated into the RISC processor to present a balance between overhead of area and critical delay [1] [2].

In this project, the set of Cadence tools was used with the 0.18 μ m technology of the TSMC Design-Kit [3].

The section 2 and 3 detail, respectively, the RISC processor and the Wallace Tree Multiplier designed. The flow of the standard cell project is presented in section 4. The design decisions related to the integration of the processor with the Wallace Tree Multiplier are discussed in section 5. Finally, section 6 presents the results and conclusions of this work.

2. RISC Processor

The RISC processor used as an example in this work was proposed by West and Eshraghian [1]. This is a 16-bit processor that has a pipeline of instructions and modules that allow the communication of the processor with external circuits, such as a co-processor. The Fig. 1 shows the organization of the modules that compose this processor. There is the ALU_DP module, which is composed of sub-modules IO_REGS, ALU and EXT_BUS_DP. The IO_REGS is responsible for providing the appropriate operands for the ALU. The block called ALU also includes registers to store data from the external bus and to keep the pipeline temporal barrier, in addition to the registers required to store the inputs A and B and the resulting output of ALU [1]. The ALU itself is divided into 3 sub-modules: the adder, the Boolean unit and the shifter.

In order to allow greater performance of multiplication operations by running them directly in hardware, this module was modified by the inclusion of a Wallace Tree Multiplier. The design of this multiplier is explained in the section 3 below.

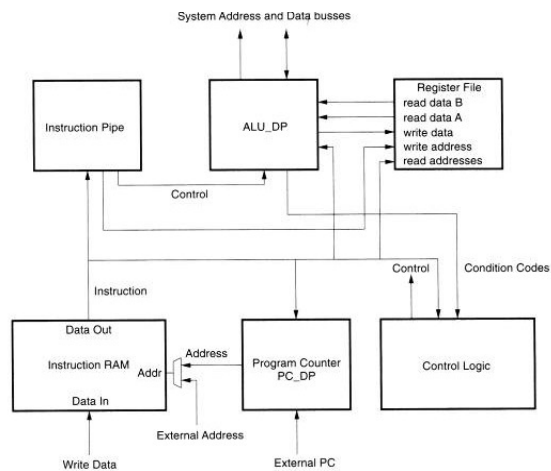


Fig. 1 – RISC processor [1]

The registers bank is composed by 128 words of 16 bits each. This module has a port for writing addressed by WA and two ports for reading the operands (A and B) addressed by RA0 and RA1 [1].

To understand better the proposed organization it is necessary to examine the ISA (Instruction Set Architecture) of processor, which is composed of two groups: Instructions of ALU and Instructions for Transfer of Control.

The instruction classes for Control Transfer includes conditional and unconditional jumps, and calls to subroutines. ALU-type instructions deal with the execution of arithmetic and logic operations between operands in the registers bank [1]. The instruction word has a maximum size of 32 bits, and in some cases only 28 bits are used. There are four types of structures for the instruction word, of which three belong to the instruction class for ALU operations and one belongs to the class of Control Transfer, as seen in Fig. 2.

#bits=2 <i>IT=1</i>	6 <i>OP</i>	8 <i>WR</i>	8 <i>RA</i>	8 <i>RB</i>
#bits=2 <i>IT=2</i>	6 <i>OP</i>	8 <i>WR</i>	8 <i>RA</i>	8 <i>LITERAL</i>
#bits=2 <i>IT=3</i>	6 <i>OP</i>	8 <i>WR</i>	12 <i>LITERAL</i>	
#bits=2 <i>IT=4</i>	6 <i>OP</i>	8 <i>COND</i>	12 <i>JA</i>	

Fig. 2 – Format of Instructions.

3. Wallace Tree Multiplier

Examining an adder more carefully, it is possible to notice that it is basically a counter that counts the number of 1's in A, B and C inputs, coding the SUM and CARRY outputs [1]. The Tab. 1 illustrates this concept.

Tab.1 - Adder as a counter of 1's

ABC	CARRY SUM	Number of 1's
000	00	0
001	01	1
010	01	1
011	10	2
100	01	1
101	10	2
110	10	2
111	11	3

A one bit adder (full Adder) provides a 3:2 compression, with relation to the number of bits. The addition of the partial product in the column of a multiplier array can be thought of as the total numbers ones in each column, with the carry being passed to the next column to the left [1]. Considering a 6x6 multiplication, the product P5 is composed by the sum of six partial products and a possible carry from the sum of P4.

The Fig. 3 shows the adders necessary in a multiplier based on the style of adding Wallace Tree [1]. The adders are arranged vertically in rows that indicate the time at which the output of adder becomes available.

While this small example shows the general technique of adding Wallace Tree, it does not show the real advantage of the speed of a Wallace Tree.

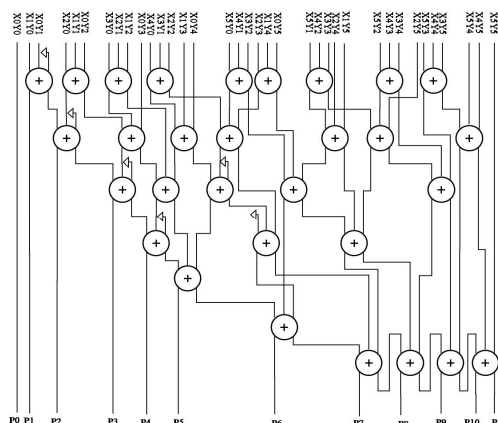


Fig. 3 – Adder Wallace Tree Multiplier for 6x6

In the figure above there are two parts identifiable as array and CPA (Carry Propagate Adder), which is seen in the lower right corner of the figure. Although the CPA has been shown in this work and implemented as a ripple carry, any technique of acceleration of carry can be used.

The delay through the array of addition (not including the CPA) is proportional to $\log(\text{base } 3/2)n$, where n is the width of the Wallace Tree [1]. In a simple array multiplier it is proportional to n . Then, in a multiplier of 32 bits, where the maximum number of partial products is 32, the compression (3:2 compressor) are: $32 \rightarrow 22 \rightarrow 16 \rightarrow 12 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 2$. Thus, there are delays of 9 adders in the array. In a multiplier array there are 16. To obtain the total time of addition, the CPA final time has to be added to the propagation time of the array [1]. This work developed a Wallace Tree multiplier of 16 bits, resulting in a considerable compression, thereby taking the advantages of technical Wallace Tree and also allowing the integration with the RISC processor, since this was also designed with a size of 16 bits.

4. Project Flow

This work was developed under a conventional design flow of ASICS (Application-Specific Integrated Circuit). The tools used in this flow are: Compilation (Xilinx ISE 9.1), Simulation (Xilinx ISE Simulator 9.1), Logic Synthesis (RTL Compiler) [3], Formal Verification (LEC) and Physical Synthesis (SoC Encounter) [3].

The Fig. 4 details the flow used. The steps shown in the figure are detailed below. The first step of the project was the RTL coding. In this step the multiplier was described in structural description, respecting the proposed topology, while the processor was described in behavior. Both descriptions were made in VHDL language and both were synthesized using the Xilinx ISE software, version 9.1 [5].

After completion of the RTL description, the multiplier and processor were simulated with the aim of a functional verification system.

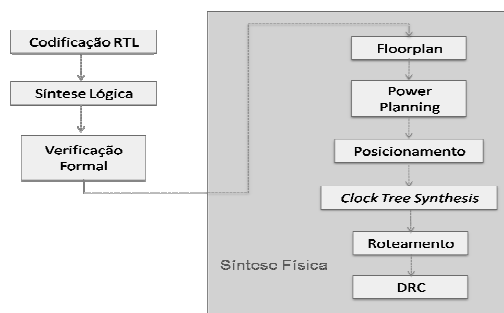


Fig. 4 – Project Flow

The next step was the synthesis of logic design. This phase includes several steps, including mapping technology, which is driven by the constraints file of the project (Synopsys Design Constraints). Moreover, at this stage were also performed optimizations of delay, power and area. The logic synthesis resulted in a netlist file where the mapped cells, they belong to the standard cells library used are described in a structural Verilog file. This file serves as input to the stages of physical synthesis and formal verification [4].

After logic synthesis, the RTL Compiler tool allows an analysis of power, area and delay of the circuit, from data provided by the library of used logic cells. Also, it is examined the critical path of the system. In this

part of the analysis, it is possible to observe changes in the technological mapping conducted by the delay constraints imposed by the constraints file written.

The next step after obtaining the netlist is the validation of it. One of these methods is a formal verification using the LEC software, in which is performed a verification of equivalence between the netlist generated and the RTL code used in the logic synthesis. With the validated netlist, the next step is the physical implementation of the project, which includes several phases, as detailed in Fig. 4.

5. Insertion of the Wallace Tree Multiplier at DataPath

To enable the integration of the RISC processor with the Wallace Tree multiplier was necessary to take some decisions before the start of the project.

Both processor and multiplier were defined with operand width of 16 bits, to interfere as little as possible in the project proposed by [1]. Another important point to keep the consistency of integration was the use of opcodes with undefined value, or without instructions related. Then one of these opcodes was used for defining the instruction of multiplication, resulting in no change in ISA for other instructions.

Whereas the critical delay of the processor is determined by the delay of the multiplier in the logic synthesis (RTL Compiler), the constraints file (SDC) was set to the maximum optimization of the circuit by the tool, thus reaching a maximum delay of 5ns for the multiplier.

In terms of physical synthesis, developed using the SoC Encounter tool, the aspect ratio, set in the Floorplan stage, was determined by the processor, i.e., a value that optimize the datapath of the processor was used. This value was set to 0.3, i.e., the height taking 30% the width of the chip.

As mentioned above, the operands of the multiplier were determined with a width of 16 bits. Therefore, the result will have a width of 32 bits, thus requiring special treatment to be stored in the memory of the processor because it has the data of 16 bits. To resolve this difference in numbers of bits two consecutive positions of the register bank were used for storing the result of the multiplication.

6. Results and Conclusions

After the physical synthesis is complete, there was an assessment of the impact of hardware relative to the processor. The data shown in Tab. 2 were extracted from the SoC Encounter tool.

Tab. 2 – Results

	Mult.	Proc.	Proc.+Mult.
#Cells	699	10393	11092
Area	25161 μm^2	227442 μm^2	252603 μm^2
Power	10.147 mW	116.514 mW	126.661 mW
Delay	5 ns	3.7 ns	5 ns

We can see that on the number of cells used, there was an increase of 6.72% with the inclusion of the multiplier, which reflects an area in 11,06% higher. This difference is explained by the use of cells with larger area, used by the multiplier circuit that due to the greater complexity involved in the operations conducted by it. On the power used by the new system, the increase was 8,70%. Moreover, the critical delay increased by about 35%, i.e. the processor, previously operated at a frequency of 270 MHz, will operate at 200 MHz

In this context, it can be taken to increase power and decrease in frequency of operation for the small gain on energy, because the transaction would be emulated by software, the computing time would be much greater, resulting in a higher cost of energy. With the multiplication performed directly by hardware, has a cost. However, this cost is offset by the completion of complex operations of multiplication in a few cycles.

7. References

- [1] N. H. E. WESTE, K. ESHRAGHIAN, "Principles of CMOS VLSI design: a systems perspective", Addison - Wesley Longman Publishing Co., Inc., Boston, MA, 1985. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
- [2] J. M. RABAEY, A. CHANDRAKASAN, NIKOLIC B., Digital Integrated Circuits: A Design Perspective, Prentice-Hall, Inc., Englewood Cliffs, NJ 2003.
- [3] Cadence Design Systems.; <http://www.cadence.com>
- [4] Tutorial 5: Automatic Placement and Routing using Cadence Encounter.; <http://csg.csail.mit.edu/6.375/handouts/tutorials/tut5-enc.pdf>
- [5] XILINX Corporation: ISE Quick Start.; <http://www.xilinx.com/itp/xilinx7/books/docs/qst/qst.pdf>.

Implementation flow of a Full Duplex Internet Protocol Hardware Core according to the Brazil-IP Program methodology

Cristian Müller, Lucas Teixeira, Paulo César Aguirre, Leandro Zafalon Pieper, Josué Paulo de Freitas, Gustavo F. Dessbesel, João Baptista Martins

{cristianmuller.50, sr.lucasteixeira, paulocomassetto, leandrozaf, josue.freitas }@gmail.com,
gfd@mail.ufsm.br, batista@inf.ufsm.br

Federal University of Santa Maria – UFSM / GMICRO

Abstract

This paper is a report of the work that is been developed at UFSM in the context of the Brazil Intellectual Property Program and the used methodology. The core that is been implemented is a full duplex partial implementation of Internet Protocol version four. It includes most of the requirements needed to work as gateway between two networks. The methodology used in the project development follows the Brazil-IP methodology and it is briefly described as well as the protocol and specifications.

1. Introduction

The development of a network protocol in hardware brings some benefits such as raising the performance of the computer network by decreasing its latency and increasing the data flow through routers and switches. Besides the performance in the live transmissions, as voice over Internet Protocol, the hardware implementation of Internet Protocol (IP) allows the protocol to work in full duplex mode, sending and receiving data at the same time, because different entities are responsible to receive and send data. Using a general purpose processor in a task such as gigabit network routing would require further features in the system as extra memory and specific software. Packing all these features in a single chip and dedicated architecture means that the clock frequency requiring can be lower. These reasons motivated us to implement the IP version four (IPv4) in hardware. This project is part of Brazil Intellectual Property Program (Brazil-IP) [1] and the main goal is to implement a full duplex IPv4 in FPGA and subsequently as an Application Specific Integrated Circuit (ASIC).

2. Similar Applications

Many companies like Intel and Cisco have increased efforts and investments in network communications. Recently these companies introduced in the market some communication protocols implemented in hardware as solution in networks. Intel has solutions that integrate Gigabit Ethernet and PHY (PHYsical layer) in the same integrated circuit [6] beyond many solutions in Network Processors [7].

Cisco has introduced the G8000 Packet Processor family that provides Gigabit Ethernet and 10 Gigabit Ethernet connection ports, has support to IP version 4 and version 6 with several other features.

3. IPv4 Development

The IP is essential when dealing with Internet and most computer networks. Its purpose is to receive and send datagrams over Internet. It is described in RFC 791 [4] and it is not a reliable datagram service. Both the sending and receiving of datagrams are not assured. Often data can get cluttered, duplicated, or simply not reaching its destination, however it is a simple and quick data exchange mechanism. Safe transmission protocols work in a higher layer in the communication stack and can work through this gateway too.

Currently the version 4 (IPv4) is the most used IP protocol. It has 32 bits of address and the IP address is used for identification of the destiny in the datagram and this address is recognized worldwide. The IP protocol implements two basic functions addressing and fragmentation.

In addition, our IPv4 IP-core partially implements the ICMP (Internet Control Messages Protocol) and the ARP (Address Resolution Protocol) protocols thus operating as a network gateway in full-duplex mode and performing tasks of addressing, routing, fragmentation and reassembly.

The ICMP protocol is the responsible for control-related tasks and supply error messages. These messages, in turn, are used to handle acknowledgment errors which may occur when an IP datagram is sent or used to check the communication status. However, due to the large number of ICMP message types, just the essential ones have been taken into account at this moment.

The ARP protocol is responsible for tracking and storing the MAC address of each machine on the network in a cache (memory). But in the core's specification was chosen to store a table containing the static IP addresses and their respective MAC addresses.

The implementation of the IP core is composed by eleven modules. Each module performs a specific feature. fig. 1 shows the block diagram of the proposed core.

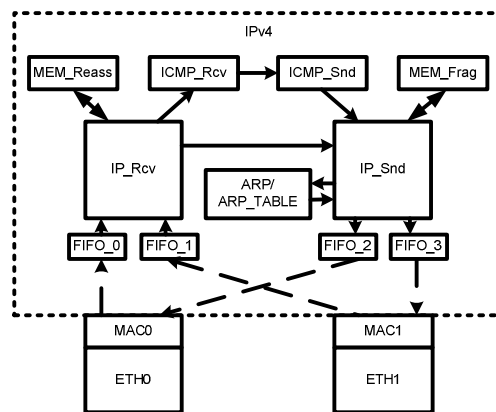


Fig. 1 – Block diagram of IPv4 IP core.

4. Project phases

The project is divided in five phases: specification, verification, codification, prototyping in FPGA and digital ASIC production and test. In industrial IC development usually codification and verification happens concurrently, in this project this phases are separated to allow every member of the group to go through all stages. The actual phase is the verification, during this phase automated testbenches are been built, the language used is SystemVerilog directed to Open Verification Methodology.

4.1. Specification

The first step of the project was the study of all IP features and stack of layers in network communication. All the system requirements of integrated circuit to be design were defined as input ports and outputs ports, modules (entities), logical features to be implemented, ports of each entry and performance goals.

The full-duplex mode operation was set because there are two entities, one responsible for receiving the data and the other responsible for sending the datagram as can be seen in fig. 1 (IP_Rcv and IP_Snd). The expected frequency of operation was defined as 125 MHz because it is the operation frequency of the Gigabit Ethernet MAC that provides a byte of data in each clock cycle.

4.2. Verification

The verification phase is critical in an ASIC design flow because if a functional error is not detected by the verification step, more cost and time will be necessary to solve this problem. If this error is not recognized in the next steps of the design flow the ASIC will not work as expected. Because of this the choose [5] of the methodology is so important and reflects in the project waste time and cost.

All project methodology is based in the Brazil-IP Verification Methodology [3] which has influences of VeriSC and OVM [4]. The methodology used for the verification is: first are build the testbenches following the right documentation made in the specification phase, after this the design is described in a hardware description language. With all verification environment built the codification engineers can work and they can run tests quickly (having the right computational resources).

The verification strategy used is to build a model that has the same functionalities as the core. Than in the same testbench the reference model and the Register Transfer Level (RTL) are placed and stimulus directed to explore each functionality of the codified design. The output of the model and the design are compared and than if there are any difference between them an error is generated warning that the design still have bugs (according to the logic implemented by the verification engineer).

To do the verification process a reference model is created in high level programming abstraction, the language used is SystemVerilog with libraries of Open Verification Methodology 2.0.1. Differently from other verification methodologies [2], this one purposes the use of the same language to code the model and the RTL design. The design of IPcore is sliced in eleven modules, each one is treated as one different design, in the last step of verification all modules will be connected together and verified with a full reference model made of all models together.

Most of testbenches use randomized stimulus instead of manually created or real stimulus, with this is possible to cover almost all possible registers values and operation cases with minimum waste time on developing stimulus. Each block needs a different kind of stimulus to be totally verified, to avoid wasting extra

time obtaining real stimulus are used the random ones. The randomization is orientated to build likely real stimulus, avoiding illegal forms and directed to the current test. The resource of coverage is possible to be used to guide the randomization process, observing and storing data about values putted in each variable or port the verification engineer can redirect the randomization so that the design passes through all operation modes. Another resource used is to run the test generating random stimulus and set up coverage targets, when reaches it the verification ends automatically. In this current project functional coverage is used in the verification, each module has all its operation states tested because the base for choosing the coverage targets are in the use cases. Part of the verification process is the design of the testbenches that is shown in the next section.

4.2.1. Testbenches

The verification environment that is been build consist in create isolated testbenches to each module and to the full design, each testbench tests different features of the design to ensure its correctness and full operation. The testbenches uses object oriented programming construction form, each internal block is a class and the design under test is driven by tasks of this classes. A transaction is a piece of data that is let into the design to run a full work cycle (as an IP datagram that will be processed). The transaction is passed to the reference model (transaction level transfer) and it is driven to the RTL design in signal form by a specific block.

The main elements of the testbench (See fig. 2) used in each test are:

Source: this entity is the transaction's generator classes when requested, it puts the transaction in two channels, they can be connected to the reference models or to the DUV's driver. One variation is the pre-source block that is used in the first step with just one output connected to a reference model.

Checker: this block is responsible to receive the transactions from the reference models and from the design monitor and compares them, if happens any difference between them an error message is generated to warn the tester engineer. FIFOS are used when transactions are been sent to the checker so it does not consider the time in its comparisons, just the sequence and values.

Transaction Driver: is responsible to drive the transactions to signals and insert it in the design under test when the final testbench is implemented. It uses the control signals and works in time domain.

Design Monitor: is responsible to do the opposite work that transaction driver does, it translates the signals received from the design under test to transaction form and sends it to the checker.

Actors: optionally some elements can be used to actuate in the inputs or outputs of the design to simulate the true working environment.

Sink: class used just to substitute the checker when only one reference model is used, it just receive and ignore the transactions.

Reference model: has the same functions as the design, but acts in transactions and is timeless.

Testbench: the module that has the channels connecting all the classes and where the test is executed.

With the methodology used in the verification first is built a single reference model testbench, where a pre-source is used just to test the functionalities of the reference model, connections and testbench. See fig. 2.

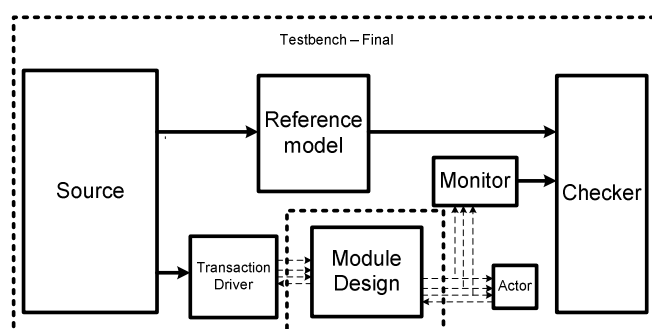


Fig. 2 – Final testbench.

The next step is building a double reference model with a complete source and checker, but using two reference models between them. In this step the source and checker are verified and will be maintained to the next steps.

The last step is the design under verification emulation, in this testbench all components are put together. The design is replaced by three entities inside the design place, the input signals are translated backwards to transactions by the first entity and driven to a reference model, in its output the transaction is driven again to signals and are exported from the design encapsulation. This test's objective is check the transaction driver's and monitor's features.

The final testbench is the design under test emulation testbench just replacing the design place entities by the RTL design. To create the code of the testbenches a tool called eTBc helps the verification engineer in this task, as is shown in the next section.

4.2.2. eTBc

To generate the test environment a tool called Easy Testbench Creator (eTBc) [8] is used. This software is the tool to support the use of the VeriSC Functional Verification Methodology [9] and makes quickly the construction of the environment for verification. Its aims to generate semi-automatic codes to each testbench element, as well as the connection between them. The software generates just the encapsulation to the reference models, design and transaction changes, all logic inside to generate oriented stimulus and to drive the stimulus inside the design needs to be build by the verification engineer.

The eTBc uses two characteristic languages types. The eTBc Design Language (eDL) and the eTBc Template Language (eTL). The eDL is used to describe the IP-core in transaction level in a file .tln that is describe in high-level approach to modeling digital systems. This file contains the description of the transaction which will be introduced in the model and the description of the signals that are connected from transaction driver to the design under verification.

The eTL language is the internal language of eTBc. It is used to declare the files containing the templates of testbenches blocks that will be generated, such as the source, driver and reference model. With these two files together it is possible to generate the elements needed for the testbenches. The eTBc generates all necessary files for the testbenches using BVM methodology such as Testbench Conception, Hierarchical Refmod Decomposition and Hierarchical Testbench.

4.3. Codification

The code build in this step is called Register Transfer Level Design, in this phase the IP-core will be described in hardware description language (HDL). During the encoding will be respected all the settings made during the specification. The HDL language used is the SystemVerilog [5], it is an emerging language and has synthesizable and non-synthesizable constructs. It is ready after achieving all the functional coverage criteria of data defined at the verification and be tested without the occurrence of errors. With the final code the logical synthesis is possible and a prototype can be programmed in a Field Programmable Gate Array (FPGA).

5. Conclusion

This paper showed some steps of the IPv4 project. Although the project is in a premature stage it reports a new methodology to the design of integrated circuits called BVM and the use of experimental tool (eTBc) that contribute to semi-automatic generation process of the testbenches. In a near future the IPv4 will be prototype in FPGA. The development of this project inside Brazil-IP program has the objective of constitute human resources capable to actuate in integrated circuit design all over Brazil.

6. References

- [1] Brazil-IP: Brazil Intellectual Property is an effort of Brazilian Universities linked to microelectronics to form workforce and relationships within design centers in all national ground. March 2009 in <http://www.brazilip.org.br/>.
- [2] RAMASWAMY, Ramaswamy, "The Integration of SystemC and Hardware-assisted Verification". March 2009 in <http://www-unix.ecs.umass.edu/~rramaswa/systemc-fpl02.pdf>
- [3] Brazil-IP Verification Methodology - BVM, Laboratory of Dedicated Architectures. March 2009 in <http://lad.dsc.ufcg.edu.br/>.
- [4] IEEE, Defense Advanced Research Projects Agency Request for Comment, Virginia, E.U.A, 1981. March 2009 in <http://www.ietf.org/rfc/rfc0791.txt?number=791>, <http://www.ietf.org/rfc/rfc0791.txt?number=826> and <http://www.ietf.org/rfc/rfc0791.txt?number=792>.
- [5] PURISAI, Rangarajan: "How to choose a verification methodology". April 2009 in <http://www.eetimes.com/news/design/features/showArticle.jhtml?articleID=22104709>
- [6] Intel. 82544EI Gigabit Ethernet Controller Datasheet. March 2009 in <http://download.intel.com/design/network/datashts/82544ei.pdf>.
- [7] Intel, Intel Network processors. March 2009 in http://www.intel.com/design/network/products/npfamily/index.htm?iid=necdnav2+proc_netproc.
- [8] PESSOA, Isaac Maia, "Geração semi-automática de Testbenches para Circuitos Integrados Digitais." March 2009 in http://lad.dsc.ufcg.edu.br/lad_files/Lad/dissertacao_isaac.pdf
- [9] SILVA, Karina R. G., "Uma Metodologia de Verificação Funcional para Circuitos Digitais." March 2009 in http://lad.dsc.ufcg.edu.br/lad_files/Lad/tese_karina.pdf.

Wavelets to improve DPA: a new threat to hardware security?

^{1,2}Douglas C. Foster, ²Daniel G. Mesquita, ¹Leonardo G. L. Martins, ¹Alice Kozakevicius

{dcfoster,mesquita}@smdh.com.br, {leonardo,alice}@smail.ufsm.br

¹Universidade Federal de Santa Maria – UFSM

²Santa Maria Design House – SMDH – Santa Maria, RS

Abstract

Nowadays in the world the frontiers are more and more tenuous, and economical relations depend on electronic transactions, of which security is essential. This security is provided by politics and protocols, where the main tool is the cryptography. This technology considers the application of algorithms combining mathematical functions and keys to transform information intelligible only to those who knows the secret. But as well as there is protection of information, also there are attacks to the cryptographic systems. Regarding the electronic circuits that process cryptographic algorithms by observing some information leaked by the circuit during data processing, it is possible to compromise its security. One of the most efficient attacks of this kind is the power consumption analysis, called DPA (Differential Power Analysis). The development of countermeasures against DPA attacks depends straightly on the knowledge of the attack. It is also important to improve the attack in order to anticipate attackers techniques. So, this work aims the implementation of a DPA attack platform and also has the objective to propose the improvement of these attacks using wavelets.

1. Introduction

Cryptographic functions play a main role in many electronic systems where security is needed, e.g. in electronic telecommunications or data protection. In particular, the use of smartcards as personal information devices for different applications (like banking and telecommunication) has increased significantly. The secure implementation of the cryptographic functions is important for the use of these security devices. In the 1990s side channel attacks [1] have appeared as a new class of attacks on security functions implemented in hardware.

Integrated circuits are built out of many transistors, which act as voltage-controlled switches. Current flows across the transistor substrate when charge is applied to (or removed from) the gate. This current then delivers charge to the gates of others transistors, interconnect wires, and other circuit loads. The motion of electric charge consumes power and produces electromagnetic radiation, both externally detectable. Therefore, individual transistor produces externally observable electrical behavior. Because microprocessor logic units exhibit regular transistor switching patterns, it is possible to easily identify macro-characteristics by the simple monitoring of power consumption. A side channel attack, like a DPA (Differential Power Analysis) attack, performs more sophisticated interpretations of this data [2].

The success of the attack is due to the number of power consumption samples and the quality of these measurements. By observing these power traces through statistical analysis it is possible to discover the most important information in the cryptographic system: the cryptographic key.

In the study of attacks to cryptographic devices, one of the most important contributions is the identification of the possible countermeasures that can be adopted to create new devices with more effective security. In [3] and [4] is shown some kind of countermeasures developed to increase difficulty and complexity in the DPA attack. But in order to investigate countermeasures, the attack technique must be mastered. So, the study of hardware attacks is also a way to prevent them. Maybe through the research of other mathematical tools to improve the analysis of the power consumption information may lead us to anticipate the attackers, allowing the development of innovative countermeasures.

This work presents a DPA attack platform and considers the possibility of using Wavelet Transform in order to improve the statistical analysis. This paper presents the DPA attack process and its main characteristics in section 2, where a DPA attack against the DES algorithm is also presented. Then in section 3, the Wavelet Transform is applied to DPA data. So, in the last section the purpose of this work is summarized.

2. DPA – Differential Power Analysis

DPA is a much more powerful attack than SPA (Simple Power Analysis), and is much more difficult to prevent [2], because SPA attacks use primarily visual inspection to identify relevant power fluctuations, while DPA uses statistical analysis and error correction techniques to extract information correlated to the secret keys. The first phase in a DPA attack concerns data collection. It may be performed by sampling the power

consumption of a cryptographic device during cryptographic operations. An oscilloscope is used to monitoring this signal by the insertion of a resistor between the VCC pins and the actual ground.

While the effects of a single transistor switching would be normally impossible to identify from direct observations of the power consumption, the statistical operations used in DPA are able to reliably identify extraordinary small differences in power consumption.

The cryptographic algorithm DES (Data Encryption Standard) had its first version developed at the beginning of 1970 by Horst Feistel. DES operation is based on 64-bit block messages, which will be encrypted with "keys" also 64-bit. Each 64-bit block message has an initial permutation in bits positions, and is divided in two halves called L and R. The eighth bit of each key-byte is called parity bit, which is responsible for the key integrity, and should be discarded. Then, a 64-bit message will have a 56-bit key.

The 56-bit key has exchanged its data and has divided equally between the first and last 28 bits, which will call C_n and D_n , respectively. After this step, DES uses circular shift left operation and "creates" more 16 values for C_n and D_n . From the concatenation of C_n and D_n , and a permutation in the bits positions, it has K_n . So, it's created a new set of keys to use in the data encryption.

In the following step, DES uses a function f , for $1 \leq n \leq 16$:

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

The f function will generate for $f(R_{n-1}, K_n)$, 8 groups of 6 bits. Each one of these groups will be inserted in the *S-Boxes*, which are tables that will transform the 6-bit input into 4-bit output.

In the last process of the encryption, to obtain the encrypted data, a final permutation of the L_{16} and R_{16} is done. In the fig. 1, it is possible to see in the power consumption trace of a cryptographic device, all the 16 rounds of the DES encryption process.

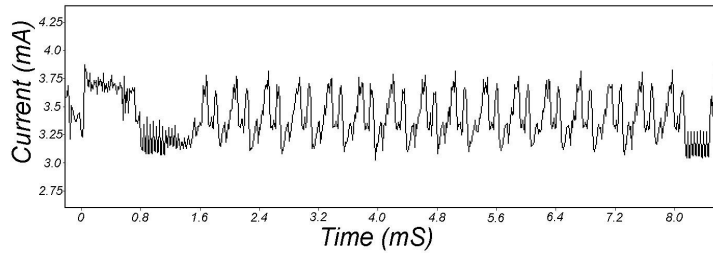


Fig. 1 – The 16 rounds of the DES process [8].

The DPA attack is described in [7] against a DES implementation and has the following process: the attack is performed targeting only one *S-Box*. First, it is necessary to make some measures (1000 samples, for instance) from the first (or the last) round of DES computation. After that the 1000 curves are stored and an average curve (AC) is calculated. Secondly, the first output bit of the attacked *S-box* is observed. This b bit depends only of the 6 bits from the secret key. Then an attacker can make a hypothesis on the involved bits. He computes the expected values for b ; this enables to separate the 1000 inputs into two categories: those giving $b=0$ and those giving $b=1$.

Thirdly, the attacker computes the average curve AC' corresponding to inputs of the first category. If AC' and AC have a difference much greater than the standard deviation of the measured noise, it means that the chosen values for the 6 key bits are correct. But, if AC' and AC do not show any visible difference, the second step must be repeated with another hypothesis for the 6 key bits. Afterwards the second and third steps must be repeated with a target bit b in the second *S-box*, as well as in the third one and so on, until the eighth *S-Box* is achieved. As a result, the attacker can obtain the 48 bits of the secret key. Finally, the remaining 8 bits can be retrieved by exhaustive search [7].

A classic DPA attack uses statistical analysis of power consumption. In the next section we introduce a new way to perform DPA, based on wavelets rather than averages.

3. DPA and Wavelets

The Wavelet functions are scaled and translated copies of a finite-length and fast-decaying oscillating waveform $\phi(x)$, which generate the space of all finite energy functions. In this work we are considering the Daubechies family of orthonormal wavelet functions, where the dilation relation between the scaling functions

$\phi(x)$ in two adjacent resolution levels is given by: $\phi(x) = \sqrt{2} \sum_{k=0}^{2N-1} a_k \phi(2x-k)$. The same kind of scale equation relating

the wavelet function to the scaling function is also valid and given by: $\psi(x) = \sqrt{2} \sum_{k=0}^{2N-1} b_k \phi(2x-k)$. These two

dilation equations completely define the scaling function, the wavelet and consequently, the spaces V_j and W_j through their filters a_k and b_k , $k = 0, 1, \dots, 2N-1$, where $b_k = (-1)^k a_{2N-1-k}$ and a_k is chosen according to the amount of null moments of the wavelet.

The Wavelet transform is a mathematical tool used to decompose a function f or continuous-time signal into different scale components and its corresponding complementary information (details $d_{j,i}$). The fast and accurate algorithm for this transformation is due to Mallat [9] and is referred as pyramid algorithm or fast wavelet transform (FWT). So, considering $c_{j,i}$, 2^j discrete values of the function f at the scale j , the fast wavelet transform, involving the filters a_k and b_k is given by the following relations: $c_{j+1,i} = \sum_{k=0}^{2N-1} a_k c_{j,2i+k}$

and $d_{j+1,i} = \sum_{k=0}^{2N-1} b_k c_{j,2i+k}$. Here the subindex $(j+1)$ is for data in a coarser resolution level. This Wavelet transform is exact, in the sense that from data at a coarser level, it is possible to recover information at the adjacent finer level j .

After computing the fast wavelet transform of the initial signal f , we obtain a factorization of f in different resolution levels: $f = \sum_{i=-\infty}^{+\infty} c_{J,i} \phi_{J,i} + \sum_{j=J}^1 \sum_{i=-\infty}^{+\infty} d_{j,i} \psi_{j,i}$, where $c_{J,i}$ represents the information of the signal on the coarsest level, and the family of coefficients $d_{j,i}$ is the details (wavelet coefficients) in different scales necessary to reconstruct the function in the fine scale 0.

Due to noise or other fluctuations, the wavelet coefficients $d_{j,i}$ are affected in different levels according to its scale j . One way to remove non-significant information from the signal is discarding wavelets coefficients from the factorization, before applying the inverse wavelet transform. In order to select the most significant details associated to the signal, we utilize a soft thresholding strategy:

$$\delta_{\lambda}^S(d_{j,i}) = \begin{cases} d_{j,i} - \lambda, & \text{if } d_{j,i} > \lambda \\ 0, & \text{if } |d_{j,i}| \leq \lambda \\ d_{j,i} + \lambda, & \text{if } d_{j,i} < -\lambda \end{cases}$$

The choice of a thresholding value λ is a fundamental issue, once the thresholded factorization $\hat{f}$ must stay “close” to f . Here as a first approach we are considering the standard universal threshold value given by Donoho [10].

As explained at the previous section, the DPA needs to compute one mean on a large number of power consumption samples to establish one correlation between the data being manipulated (and depending on few key bits) and the information leakage. This kind of attacks supposes there is an observable difference in the power consumption when a bit is set or clear. So analyzing $f = (AC' - AC)$ though the wavelet transform and considering also a thresholding strategy, it is possible to select details that are bigger than a reference value (the standard deviation of the measured noise), and consequently identifying any difference between the curves in a much more efficient manner.

Many countermeasures have been proposed in order to make those attacks difficult. Countermeasures proposed for DPA may be classified in two groups: algorithmic countermeasures on one hand and physical countermeasures on the other hand. The use of the Wavelet Transform in the DPA is still a very new approach. In this work we investigate the behavior of the power consumption through the application of several wavelet transforms, considering different Wavelet families. One of the main goals of this study is analyzing the data behavior especially in the presence of different hardware countermeasures in the cryptographic device, e.g., reduction of the signal/noise ratio (SNR), the reduction of the correlation between input data and power consumption by randomly disarrange the moment of time at result is computed.

Another popular physical countermeasure consists currently in varying the internal clock frequency in order to make DPA inappropriate. For this countermeasure application, the use of Wavelet Transform was proposed in [5], using Symlet (Symmetrical Wavelet) Transform. Here we will investigate the possibility of finding power consumption signatures for the same countermeasure, but considering Daubechies wavelets [6], exploring the orthogonal characteristic of these functions. Finally, because of the high quality of data acquisition, we also have the possibility to apply several continuous wavelets transforms, analyzing their features.

4. Conclusions and Further Work

This article presents the basis of a scientific work that is being developed during a Master Course in Computer Science at the Federal University of Santa Maria (UFSM).

The main goal of this research is to provide mechanisms to verify and ensure the security of cryptographic hardware. As a first step, a classic DPA platform is being implemented. After that we intend to use Wavelet Transform, and different wavelet families, to increase performance in the analysis of the power consumption information with the presence of different hardware countermeasures in the cryptographic device. This is an innovative approach that may lead to a new hardware attack, and consequently will allow the development of novel countermeasures, contributing to the state-of-art in hardware security.

5. References

- [1] Jens Ruendiger. "The complexity of DPA type side channel attacks and their dependency on the algorithm design". Information Security Technical Report, volume 11, Issue 3, p. 154-158, 2006.
- [2] Paul Kocher, and Joshua Jaffe, and Benjamin Jun. "Introduction to Differential Power Analysis and Related Attacks". Technical Report, Cryptography Research Inc., 1998. Available in <http://www.cryptography.com/resources/whitepapers/DPA.html> . Accessed on 2009 February 14th.
- [3] Fernando Ghellar and Marcelo Lubaszewski. "A novel AES cryptographic core highly resistant to differential power analysis attacks." In Proceedings of the 21st Annual Symposium on integrated Circuits and System Design. SBCCI '08. ACM, New York, NY, 140-145.
- [4] R. Soares, N. Calazans, V. Lomné, P. Maurine, L. Torres, and M. Robert, "Evaluating the robustness of secure triple track logic through prototyping." In Proceedings of the 21st Annual Symposium on integrated Circuits and System Design. SBCCI '08. ACM, New York, NY, 193-198.
- [5] Xavier Charvet, and Herve Pelletier. "Improving the DPA attack using Wavelet Transform". Available in <http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-3/physec/physecdoc.html>. Accessed on 2009 February 19th.
- [6] Ingrid Daubechies. "Ten Lectures onWavelets". CBMS-NSF Lecture Notes nr. 61. SIAM: Society for Industrial and Applied Mathematics, 1992. Philadelphia, Pensylvania.
- [7] Daniel Mesquita, Jean-Denis Techer, Lionel Torres, Michel Robert, Guy Cathebras, Gilles Sassateli and Fernando Moraes. "Current Mask Generation: an analog circuit to thwart DPA attacks". IFIP TC 10, WG 10.5, 13th International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC 2005), October 17-19, 2005, Perth, Australia.
- [8] Paul Kocher, Joshua Jaffe and Benjamin Jun. "Differential Power Analysis". In Proc. 19th International Advances in Cryptology Conference – CRYPTO '99, pages 388–397, 1999.
- [9] S. G. Mallat. "A theory for multiresolution signal decomposition, the wavelet representation". IEEE Transactions on Pattern Analysis and Machine Intelligence 11, 1989, 674-693.
- [10] D. Donoho, I. Johnstone. "Adapting to unknown smoothness via wavelet shrinkage". Journal of the American Statistical Association 90, 1995, 1200-1224.

Design of Electronic Interfaces to Control Robotic Systems Using FPGA

¹André Luís R. Rosa, ¹Gabriel Tadeo, ¹Vitor I. Gervini, ²Sebastião C. P. Gomes,
¹Vagner S. da Rosa

andrerodeghiero@gmail.com, debage_ecomp@yahoo.com.br, gervini@gmail.com,
scpgomes@terra.com.br, vsrosa@inf.ufrgs.br

¹Centro de Ciências Computacionais – FURG

²Instituto de Matemática e Física – FURG

Abstract

This article discusses aspects related to the control of robotic systems using a real time system implemented in FPGA. In order to control an automated environment it is necessary to capture the signals from various sensors of the plant to feed to a control law that is responsible for transmitting the control signals to the actuators. The aim of this work is to describe the operation of electronic circuits responsible for matching the signals that come in the controller, provided by the sensors, as well as the output signals from this to be directed to the frequency inverters that control AC motors.

1. Introduction

A robotic system is composed of sensors and actuators. The control of these systems is generally performed in closed loop. For that the control system need to capture the signals from the sensors and compare them to a signal used as a reference by means of a control law. The resulting signal is applied to an output connected to the actuator to minimize the positioning error according to the controller specifications [1].

According to the involved demands, it has been developed a real time control system of hardware and software based on FPGA (field programmable gate array) XC2VP30. This device has 2 PowerPC processors and 30 thousand programmable logic elements for the control system's implementation. The control system has supply limitations of voltage and current to external devices connected to its posts, what discards the possibility to drive actuators directly connected in this interface. The signal entering the controller must be limited to have low voltage and current values, consequently, it is indicated the use of an electrical insulation system.

The goal of this work was to design and build electronic interfaces to capture signals from various sensors and send them to a real time system, as well as send signals of control of this system to actuators, more precisely, frequency inverters responsible for the control of AC motors. In this article will be addressed the electro-electronic design part, simulations and experimental results of FPGA interfaces which were used.

2. Interfaces

The interfaces were divided in three modules, according to different uses and the type of connector I/O that FPGA offered. These modules are:

- Output module
- Incremental encoders module
- Digital inputs, RS-485 interfaces, and security systems module

These modules are described in details in the following sections.

2.1. Output Module

The output module has two similar circuits which increase the voltage (PWM signal) from the FPGA (0-2,6V) for a range of 0 to 12V approximately, what is enough to enable inputs of frequency inverters.

The first circuit, presented in Fig 1 (a), is suitable for driving digital circuits, such as digital inputs of frequency inverters responsible for the activation of the direction of rotation of motors. This circuit is composed of an operational amplifier working in the non-inverter amplifier configuration [2]. It receives a PWM signal (pulse width modulation) in its inputs and makes a comparison with a reference voltage. If the FPGA output voltage is greater than this value the operational amplifier places in its output a voltage of 12V, corresponding to high logic level. Otherwise, output remains low (0V).

The second circuit, presented in Fig 1 (b), is suitable for activation of analog devices, such as analog inputs of frequency inverters responsible for proportional control of rotation of motors. This circuit is composed of 2 RC networks, 2 operational amplifiers and resistors, which form a two stage low-pass filter. This circuit operates in the following way. First, a PWM signal (digital signal where the duty cycle time can be adjusted,

being held the same wave period) of 8 bits (256 different levels of voltage available) reach the RC circuit. This circuit is a low pass filter designed to extract the average voltage (proportional to the duty cycle voltage). The resulting signal is transferred to the non-inverter input of the operational amplifier. In the inverter input there is a reference value that once overcome, multiplies the effective voltage in the RC circuit by the configured gain in the inverter input. The second RC circuit filters the signal further and is followed by a unity gain amplifier to boost the output current [3]. On the developed board there are 7 analog outputs and 14 digital outputs (Fig 1).

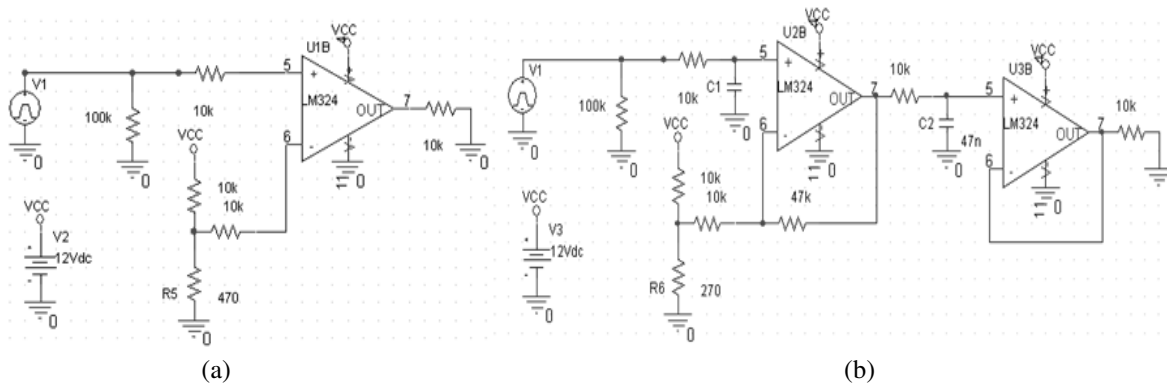


Fig. 1 – (a) Digital FPGA interface and (b) analog (PWM based) FPGA interface.

2.2. Incremental Encoders Module

This module is responsible to match the voltage levels (5V TTL, differential), usually used in the angular position sensors (incremental encoders), into acceptable levels of voltage by FPGA (0-3V). The signal pulses from these sensors are square waves that provide the reference in the motor shaft position to the control system. Such signals are transmitted on two channels, A and B, delayed 90° in phase to indicate the direction of rotation. The decoding of these signals is done by FPGA. Normally, such sensors send two differential signals: A, A $\neg$ (complement of A) and B, B $\neg$, which if used, reduce the possibility of errors in counting (decoding) due to electric interference. The project has a stage of pre-decoding performed by two operational amplifiers (one amplifier for one channel and its respective complement), which send their results to FPGA to perform the decoding of the direction of rotation, as well as the change position by counting the pulses. On the developed board there are 8 inputs available. Fig 2 shows the schematic of this circuit. The output has a voltage limiting network formed by a zener diode and a resistor to protect FPGA from overvoltage in its inputs.

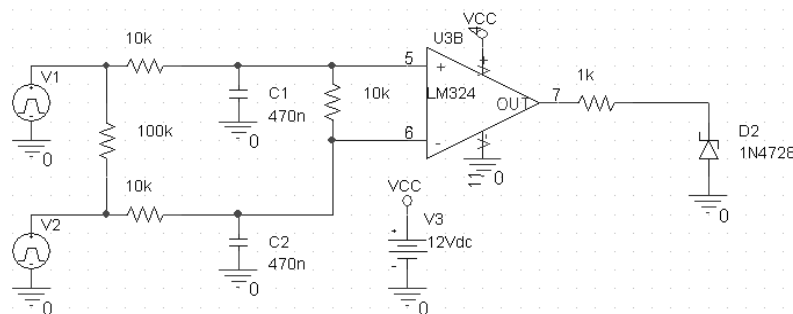


Fig. 2 – Interface circuit for an encoder channel.

2.3. Digital Inputs, RS-485 Communication, and Security System Module

This module, presented in Fig 3 (a), has 24 digital inputs isolated by the optocouplers that support a range of 0 to 30V in their inputs, converting them to the acceptable range for the FPGA (0 - 3,3V). Such electrical isolation is essential to preserve the system's electric integrity. In the event of a voltage peak higher than supported by the optocouplers, they will be damaged instead of FPGA. Such inputs are responsible for the sensing system, like detection of activation of limit switch sensors, confirmation of the operation of the frequency inverters or any other digital signal that needs to be monitored.

Additionally, this module includes 2 RS-485 communication channels. This standard was developed to meet the need of multi-point communication and its format allows connecting up to 32 devices [4]. Differently of the RS-232 standard (where there is a wire transmission, another wire for reception and a ground wire which is used as a reference for the other two), the RS-485 uses the method of differential transmission, where only

two wires are used (forming a differential pair). The logic level is decoded according to the voltage difference between them. In the event of any electromagnetic interference in pair, the induced noise will affect both signals in the same way, not damaging the decoding. Thus, the distance between the devices can be increased. This interface is implemented in a straightforward way through the use of a MAX485 integrated circuit.

The last circuit in this module is a security system, presented in Fig 3 (b), based on RC circuits and operational amplifiers, which are responsible to monitor the operation of the system controller (real time system). The FPGA generates a clock in one of its outputs which is analyzed by the circuit of protection. If any failure occurs with FPGA, the clock signal will be compromised, causing the activation of the safety circuit. Once active, it switches a relay that disconnects the power from the system, providing security to the robotic system and its users.

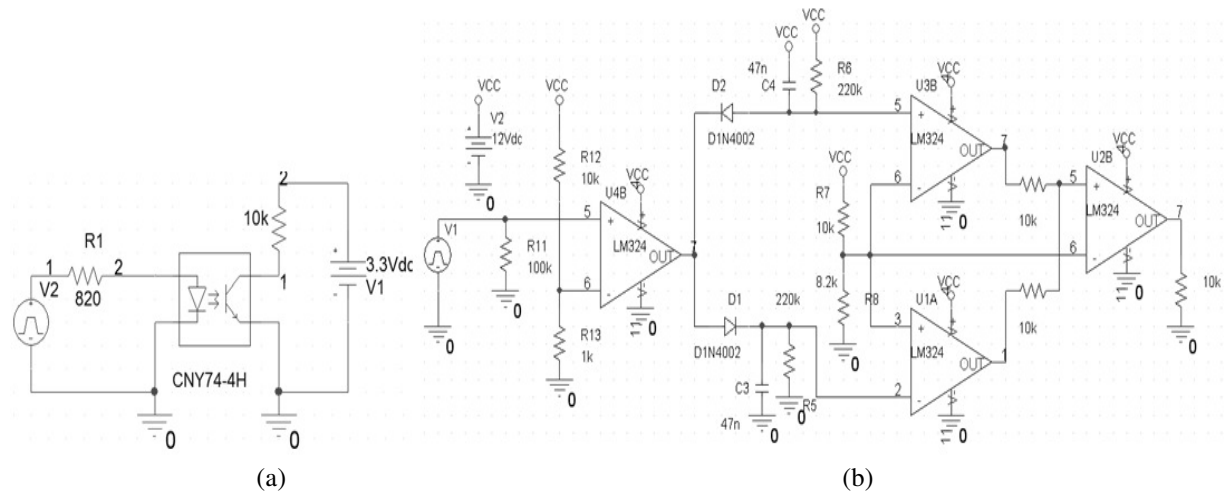


Fig. 3 – (a) Digital inputs circuit and (b) security system circuit.

3. Results

The design of interfaces went through several validation steps before being concluded. After the elaboration of the schematic and the analysis of the components involved, the circuits were tested in simulation.

Fig 4 shows a comparison between simulation and experimental result for the analog output circuit (Fig 1b) Fig 4a show the level of voltage in the output of the circuit of analog outputs by applying a PWM signal with a duty cycle of 75% in its input. After a set of simulations, printed circuit boards were developed and the circuits were built. Fig 4b shows experimental result for the analog output circuit (Fig 1b) when the PWM signal is being generated by the FPGA.

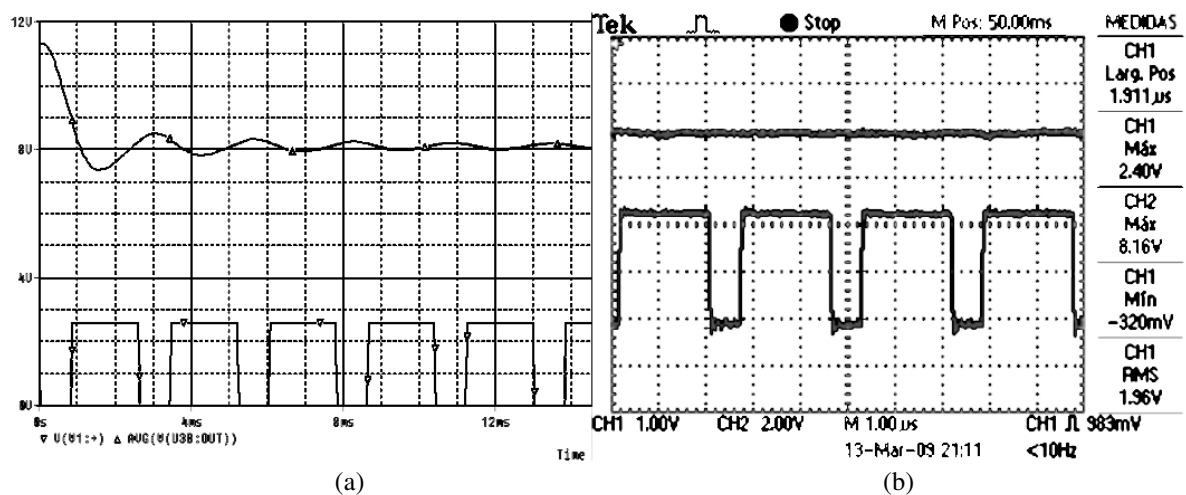


Fig. 4 – Comparison between (a) simulation and (b) experimental results for the analog output.

Fig 5 show the printed circuit boards developed for the circuits presented in this paper.

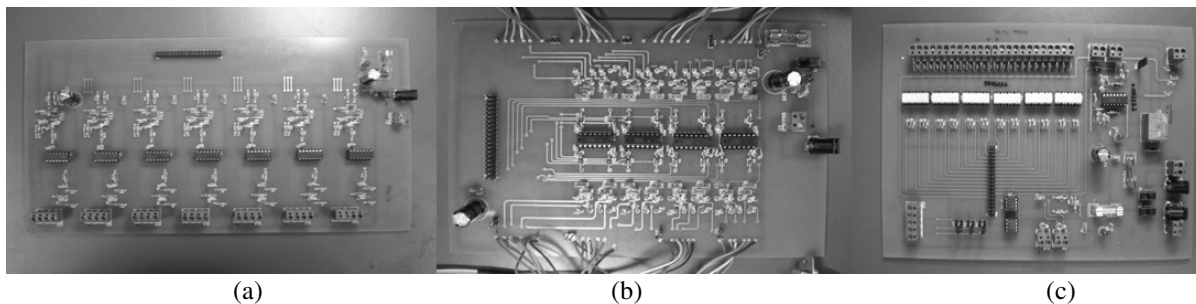


Fig. 5 – Developed interfaces: (a) Digital/analog output; (b) Incremental encoder interface; (c) Digital Input/RS-485/Security interface.

4. Conclusions

The development of electronic interfaces to match the I/O signals between the external devices and the real time system were successfully developed and the experimental results show that the used circuits are robust and safe. This I/O electric system was connected to a control system and was able to control in closed loop up to 7 AC motors simultaneously (using the 7 outputs available of the Output Module), communicate with up to 62 (30 devices in slave configuration + real time system in master configuration per channel) devices that support the standard RS-485 and receive signals from 24 digital devices (according to the 24 digital inputs of the interface). As shown throughout this paper, the main motivation at the development of these circuits was to control AC motors. However the developed interfaces can control any robotic system, through some minor changes in the input and output signal levels making the system composed by the developed interfaces and a FPGA board very versatile.

Future work will improve these interfaces, making a better characterization of the electric signals generated and improving the performance of the analog interfaces using modulation schemes other than PWM.

5. References

- [1] K. Ogata, “Engenharia de Controle Moderno”, 4th ed., São Paulo: Prentice-Hall, 2003, pp. 1-7.
- [2] R. Boylestad, L. Nashelsky, “Dispositivos Eletrônicos e Teoria de Circuitos”, 6th ed., Rio de Janeiro: LTC, 1999, pp 428-477.
- [3] R. Boylestad, “Introductory Circuit Analysis”, 10th ed., Prentice-Hall, 2002, pp. 1094-1123.
- [4] T. G. Moreira, “Desenvolvimento das Interfaces Eletrônicas Para Plataforma de Ensaio de Manobras de Embarcações”, graduation thesis, Engenharia de Computação, FURG, Rio Grande, 2007

Design Experience using Altera Cyclone II DE2-70 Board

Mateus Grellert da Silva, Gabriel S. Siedler, Luciano V. Agostini, Júlio C. B. Mattos
{mateusg.ifm, gsiedler.ifm, agostini, julius}@ufpel.edu.br

Universidade Federal de Pelotas
Grupo de Arquiteturas e Circuitos Integrados
Pelotas – Brasil

Abstract

Nowadays, electronic systems perform a wide variety of tasks in daily life, and the growing sophistication of applications continually pushes the digital design to more complex levels. Thus, it is necessary to use several tools to assist the design process, e.g. FPGAs and CAD tools. This paper presents an entire design flow process using Quartus II Design Software and Cyclone II DE2-70 board from Altera. As case study, a Neander processor computer was designed and implemented in FPGA.

1. Introduction

The integrated circuit (IC) technology has evolved drastically through the past decades. This technology is enabling a variety of new electronic devices that are everywhere for example, mobile telephones, cars, videogames and so on. As a result, digital design deals with a large amount of components and complexity. On the other hand, tools and techniques were developed intending to support and making the design process easier [1].

During the design process, in order to make sure that the project is being developed properly, it must be validated and tested. Moreover, the system should be prototyped. Prototyping consists of synthesizing a project in a tool or device to evaluate the results and check if the output is as expected.

The FPGAs (Field Programmable Gate Arrays) [2] offer the possibility of fast prototyping to the final user level with a reduced design cycle, making them an interesting option in system projects. FPGAs are high density programmable logic devices that can be used to integrate large amounts of logic in a single IC [2]. These devices present a useful combination of logic and memory structure, speeding the project prototyping up. Furthermore, FPGAs are reprogrammable and offer a relatively low cost. An alternative to FPGAs would be an Application Specific Integrated Circuits (ASICs). However, ASICs are expensive to develop and take a long time to be designed, increasing the time to market.

Over the past years, designers have embraced some hardware description languages like VHDL and Verilog, because these languages can provide a good description level and high portability. VHDL [2][3] is an acronym for VHSIC Hardware Description Language. This language presents a straightforward syntax, making it possible to program devices quickly. Furthermore, it is portable and flexible.

The aim of this paper is to present the design experience and prototyping a project using Altera DE2-70 FPGA of the Cyclone II family [4]. The project was designed using VHDL and synthesized with Quartus II Design Software [5] from Altera. As case study, the didactic processor Neander was designed.

This paper is organized as follows. Section 2 explains the Neander computer and some changes that were made. Section 3 presents the synthesis results obtained from the experiment. Finally, section 4 concludes this work and discusses future work.

2. Case Study

In this work, a Neander computer was described using VHDL. This processor has a small instructions set, and its implementation is quite easy. It is composed by a set of 8-bit registers, a memory structure and a Control Unit (CU). It has the following characteristics: address and data bus width of 8 bits, two's complement data representation, one Program Counter (PC) with a bus width of 8 bits, one status register with 2 conditions (negative and zero) [6]. The ALU component used in this work is capable of doing 2 arithmetic (ADD and SUB) and 3 logical operations (AND, OR and NOT) and it also has a function to simply send operand Y straight to the accumulator. The block diagram for this architecture is shown at fig. 1.

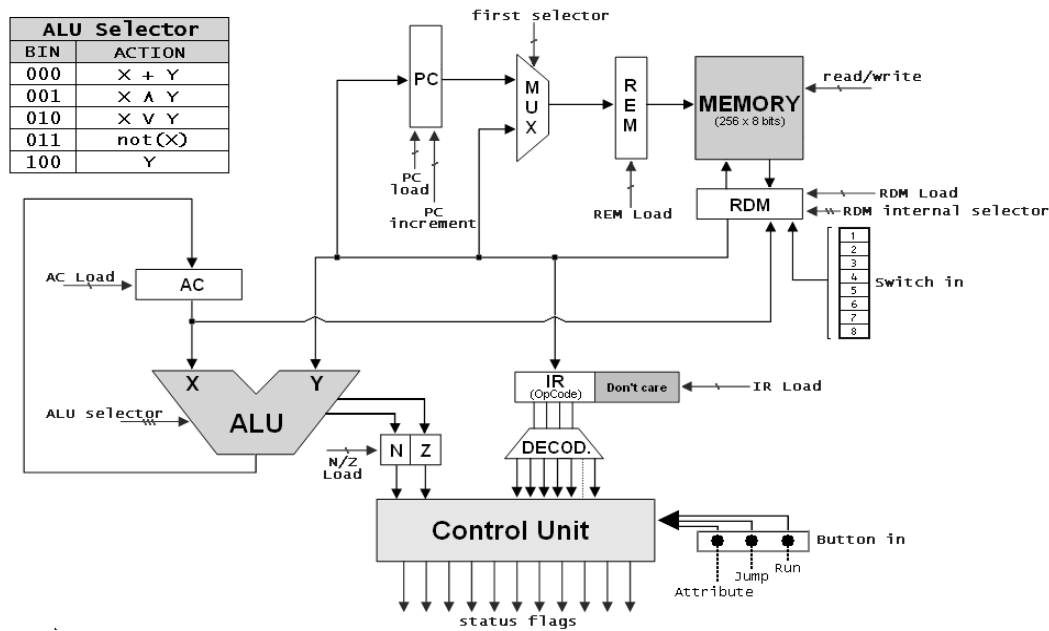


Fig. 1 – Neander Architecture.

The most complex component of the architecture is the control unit and it is responsible for controlling the flow of the architecture by communicating with the registers and other components. The control unit is based on a Finite State Machine (FSM). Due to the limited number of operations available for this processor, the Opcode (operation code) has a 4 bits width. The instruction set and operation code is presented at tab. 1.

Tab.1 –Neander Instruction Set and operation code.

Instruction	Code	Execution
NOP	0000	No operation
STA addr	0001	$\text{MEM(addr)} \leftarrow \text{AC}$
LDA addr	0010	$\text{AC} \leftarrow \text{MEM(addr)}$
ADD addr	0011	$\text{AC} \leftarrow \text{MEM(addr)} + \text{AC}$
OR addr	0100	$\text{AC} \leftarrow \text{MEM(addr)} \text{ or } \text{AC}$
AND addr	0101	$\text{AC} \leftarrow \text{MEM(addr)} \text{ and } \text{AC}$
NOT	0110	$\text{AC} \leftarrow \text{NOT AC}$
JMP addr	1000	$\text{PC} \leftarrow \text{addr}$
JN addr	1001	IF $\text{N}=1$ THEN $\text{PC} \leftarrow \text{addr}$
JZ addr	1010	IF $\text{Z}=1$ THEN $\text{PC} \leftarrow \text{addr}$
HLT	1111	End (halt)

For this work, an architecture that allows the user to manage memory data directly through the prototyped board was designed. In other words, memory data input occurs by simply setting a value using the switches (indicated as *Switch in* at fig. 1) and then pressing a button to attribute it where the PC is pointing (the PC value is indicated in one of the displays available in the board). To make that possible, it was necessary to implement another FSM to the design. One FSM is responsible for the user interface with the architecture's memory and another FSM is responsible for executing the instructions previously established. The architecture chooses which FSM will be executed based on a flag indicator that is set to '1' when the execution button is pushed. The state diagram presented at fig. 2 shows the flow of the memory management process, and fig. 3 shows the simplified state diagram for instructions execution.

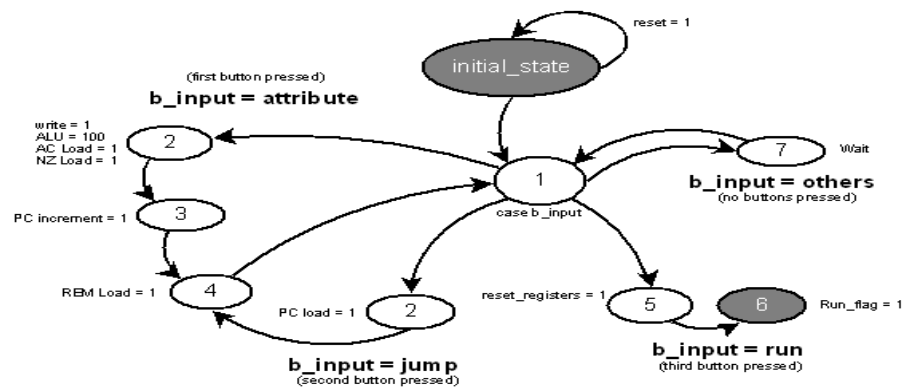


Fig. 2 – Memory interface FSM.

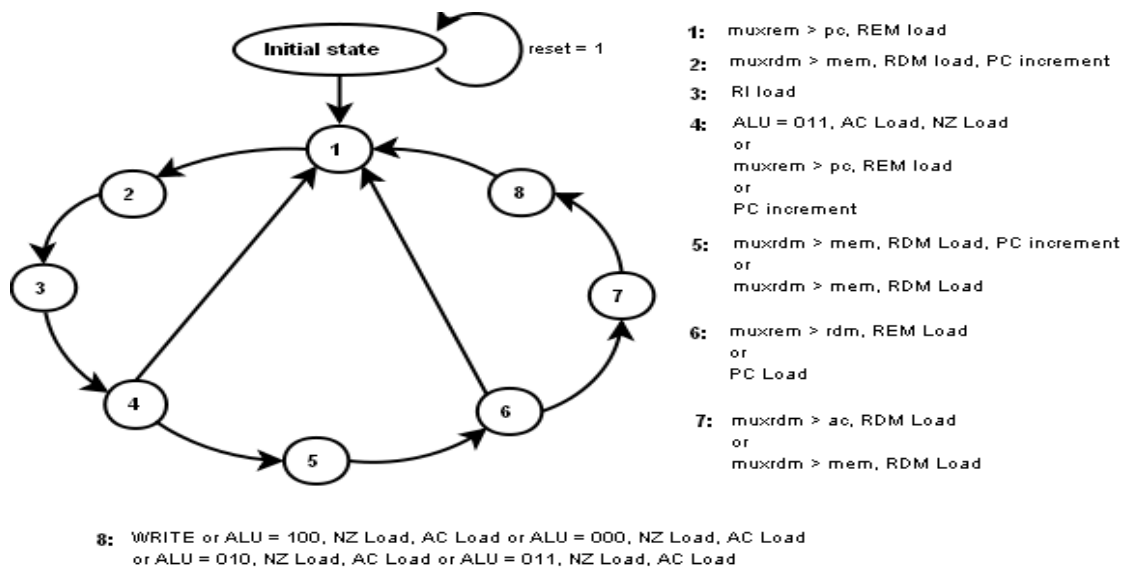


Fig. 3 – Neander's execution FSM.

For the user memory management interface, two more inputs were added in the original architecture: one for data values (8 switches are available for that) and one to select the input data process (activated through 3 buttons). The processes available are Attribution, which sends the value to the memory position where the PC is currently indicating; Jump, which sends the value indicated to the PC; and Run, which sends the PC value back to the first position and sets a run flag to '1'. To get a better view of these inputs check fig. 1.

3. Synthesis Results

The designed architecture was described in VHDL and synthesized to EP2C70F896C6N Altera Cyclone II FPGA. The Altera DE2-70 Board (fig. 4) is a development and education board, composed of almost 70,000 Logical Elements (LEs), multimedia, storage and network interfaces and memory components. It is suited for academic use, since it has a diverse set of features [7]. The chosen board has much more logical resources than needed, but it was used due to the number of switches, allowing an instruction word width of 8 bits, and for its large amount of displays that were used with a BCD converter, enabling the user to have a view of the values present in the PC and in the accumulator.

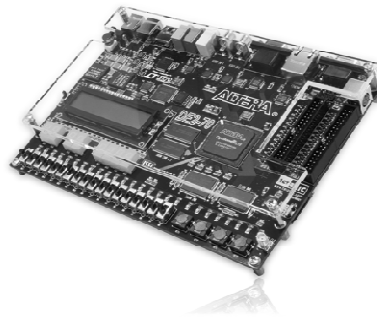


Fig. 4 – DE2-70 Altera's Cyclone II Board.

Tab. 2 presents the synthesis results for the main internal modules and for the whole architecture, showing the maximum operation frequency achieved and the amount of hardware consumed. The tab. 2 shows the results in terms of ALUTs (Adaptive Look-up Tables) and DLRs (Dedicated Logic Registers). As one can see, the Neander processor uses few resources of the FPGA. However, the display's BCD converter uses more hardware resources (ALUTs) and has a low frequency, since it uses integer divisions to calculate decimal and hundred positions of a given value.

Tab.2 – Synthesis results

Module	#ALUTs	#DLR	Frequency
Control Unit	262	31	194 Mhz
Datapath	544	544	172 Mhz
Display	2969	0	5.9 Mhz
Neander computer	3680	578	5.8 Mhz

4. Conclusions and Future Works

This work presented a design and prototyping experience using Cyclone II DE2-70 board from Altera. The Neander processor is described in VHDL and synthesized in FPGA. To successfully implement it, several simulations were conducted, making use of an FPGA to assist these tests, reducing the validation time. The experience was successfully implemented and acquired satisfactory results. The designed architecture has a low frequency, due to its BCD converter, but it consumed just about 4% of the used hardware. For future work, we plan to implement case studies of a higher complexity to explore the full capacity of the Cyclone II DE2-70 board, as well as validating the architectures with ModelSim software.

5. References

- [1] L. Carro, "Projeto e Prototipação de Sistemas Digitais", Editora da Universidade/UFRGS, Porto Alegre, 2001. (in Portuguese).
- [2] K. Skahill, "VHDL for Programmable Logic", Addison Wesley Longman, Menlo Park, 1996.
- [3] R. Lipsett. "VHDL: Hardware Description and Design", Boston: Klumer Academic Publishers, 1990.
- [4] Altera Corporation, "Cyclone II FPGAs at Cost That Rivals ASICs", available at <http://www.altera.com/products/devices/cyclone2/cy2-index.jsp>
- [5] Altera Corporation, "Design Software", available at <http://www.altera.com/products/software/sfw-index.jsp>.
- [6] R. F. Weber, "Fundamentos de Arquitetura de Computadores", Editora Sagra Luzzatto, Porto Alegre, 2004. (in Portuguese).
- [7] Altera Corporation, "DE2 Development and Education Board", available at <http://www.altera.com/education/univ/materials/boards/unv-de2-board.html>.



IC Process and Physical Design

Passivation effect on the photoluminescence from Si nanocrystals produced by hot implantation

¹V. N. Obadowski, ^{1,2}U. S. Sias, ³Y.P. Dias and ³E. C. Moreira

vininovicki@gmail.com, uilson@cefetrs.tche.br, yarapd10@hotmail.com, emoreira.unipampa@gmail.com

¹Instituto Federal Sul-rio-grandense

²Instituto de Física – Universidade Federal do Rio Grande do Sul

³Universidade Federal do Pampa

Abstract

The intense research activity on Si nanostructures in the last decades has been motivating by their promising applications in optoelectronics and photonic devices. In this contribution we study the photoluminescence (PL) from Si nanocrystals (Si NCs) produced by hot implantation, after a post-annealing step in a forming gas ambient (passivation process). When measured at low pump power (~ 20 mW/cm²) our samples present two photoluminescence (PL) bands (at 780 and 1000 nm, respectively). Since each PL band was shown to have different origin we have investigated the passivation effect on them. We have found that only the 1000 nm PL band is strongly influenced by the passivation process in its intensity. In addition we have studied samples implanted at 600 °C annealed at 1150 °C for different time intervals and further passivated. The results show that the passivation effect on the 1000 nm PL band is strongly dependent on the preannealing time.

1. Introduction

The extensive investigation on Si nanostructures in the last decades has been motivating by their promising applications in optoelectronics and photonic devices [1-3]. The study of structures consisting of Si nanocrystals (Si NCs) has mostly devoted to improve their quantum efficiency for photoluminescence (PL) as well to understand their light absorption and emission processes. Although the exact mechanism for light emission still remains some kind controversial, nowadays it is well established that basically the emission energy is dependent on either the NCs size [4-6] or radiative processes at the Si NCs/matrix interface [6-9]. Concerning the Si nanocrystals interface, the SiO₂ matrix is ideal, since it can passivate a large number of dangling bonds that cause non-radiative quenching [10]. Nevertheless, the SiO₂ matrix is not enough in passivating non-radiative surface defects that compete with radiative processes [11]. It has been shown that a nanocrystal effectively becomes "dark" (stops luminescing) when it contains at least one defect. This is because in the visible PL range the non-radiative capture by neutral dangling bonds is much faster than radiative recombination [11]. Consequently, hydrogen passivation of nanocrystals has been shown to increase considerably their luminescence efficiency by inactivating these defects [12-16].

On the other hand, Si NCs embedded in SiO₂ matrix have been extensively studied as a function of the implantation fluence, annealing temperature and annealing time [7, 12-14]. However, in all the previous works the implantations were performed at room temperature (RT) and only one PL band centered at around $\lambda \sim 780$ nm was observed. More recently, we have taken another experimental approach [17]. The Si implantations into SiO₂ matrix were performed at high temperatures (basically between 400 and 800 °C), at a fluence of 1×10^{17} Si/cm², being the spectra obtained in a linear excitation regime (low pump power regime - power density of 20 mW/cm²). As a consequence two PL bands were observed, one with lower intensity at $\lambda \sim 780$ nm and the other with higher intensity at λ around 1000 nm. Transmission electron microscopy analyses (TEM) of the hot-implanted samples have revealed that the NCs size distribution was broader with a larger mean size as compared to RT implantations.

Since on the above described experimental conditions appear two PL bands, some interesting features deserve investigation: the behavior of both PL bands under hydrogen passivation and the influence of preannealing time at 1150 °C on the passivation process.

In the present work, for a given Si implantation fluence we have submitted the samples to a forming gas (FG) ambient. PL measurements were carried out before and after the passivation process. Finally, for the same implantation fluence we have submitted the samples to several preannealing times (at 1150 °C), then performed the FG treatment and observe the behavior of both PL bands.

2. Experimental details

A 480 nm thick SiO₂ layer thermally grown on (100) Si wafer was implanted with 170 keV Si ions at a fluence of 1×10^{17} Si/cm² providing a peak concentration profile at around 240 nm depth and an initial Si excess

of about 10 %. Samples were implanted at room temperature (RT) and at 600 °C. The as-implanted samples were further annealed at 1150 °C under N₂ atmosphere in a conventional quartz-tube furnace for times ranging from 10 up to 600 min. This is a standard thermal treatment used to produce luminescent nanocrystals, with the high-temperature anneal required to nucleate and precipitate the nanocrystals and also to remove implantation damage (such as non-bridging oxygen hole centers and oxygen vacancies) [18]. After the high-temperature annealing step, a large number of defects is still present on the Si NCs [14]. Subsequent hydrogen passivation anneals were performed in high-purity (99,98 %) forming gas atmosphere (5% H₂ in N₂) at 475 °C for 1 h.

PL measurements were performed at RT using a xenon lamp and monochromator in order to get a wavelength of 488 nm (2.54 eV) as an excitation source. The resulting power density on the samples was about 20 mW/cm². The emission was dispersed by a 0.3 m single-grating spectrometer and detected with a visible-near infrared silicon detector (wavelength range from 400 to 1080 nm). All spectra were obtained under the same conditions and corrected for the system response.

3. Results

In fig. 1 we show typical PL spectra of a sample implanted at 600 °C and 30 min N₂-annealed at 1150 °C, before and after passivation in FG. As can be observed, the long wavelength PL band (around 1000 nm) is strongly affected by the passivation process as compared to the short wavelength one. Through a two-Gaussian fitting procedure of the PL spectra we have observed that the intensity of the 1000 nm - PL band increases by a factor of almost 20 followed by a redshift of around 150 nm, while the PL band centered at 780 nm increases by a factor of two, without peak shifting. Since the PL band located at 780 nm was weakly affected by the passivation process, from now we will only discuss the results concerning the long wavelength band.

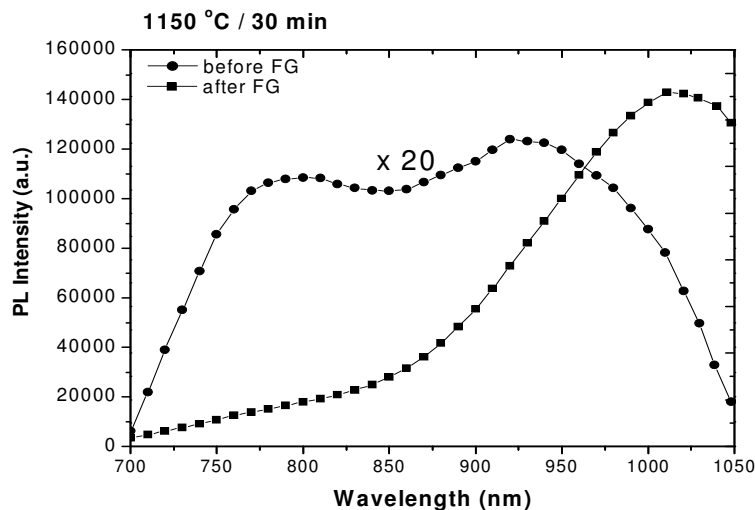


Fig. 1 – Typical PL spectra of a sample implanted at 600 °C and 30 min-annealed at 1150 °C before (circle) and after (square) the passivation process. Note the change in the scale to the sample before the FG treatment.

In fig. 2 (a) and (b) are summarized, respectively, the intensities and positions of the long wavelength PL peak of samples implanted at 600 °C (circle) and at RT (square) as a function of the annealing time at 1150 °C, done previously to the FG treatment. In the figure are shown the results before (dashed lines) and after (full line) the passivation process. From the figure we can point out the following features: i) the PL intensity enhancement due to the FG treatment is higher for the hot-implanted samples (a factor of two) than the RT one and follows the behavior before passivation with the annealing time; ii) the PL increase with the passivation and saturates after 2 h of preannealing at 1150 °C; iii) the PL peak position of the hot-implanted sample is about 50 nm redshifted in relation to the RT-implanted one before the passivation. As a consequence of the FG treatment this redshift is enlarged for the first 2 h of preannealing and then remains stable.

Fig. 3 shows the ratio between the PL spectra obtained after and previous to the FG treatment. As can be observed, the passivation effect is intensified to the long wavelength direction. Fig. 3 (a) shows that the relative PL intensity diminishes as the pre-annealing time is longer. In fig. 3(b), are compared, for the same time of preannealing, the relative PL intensities of samples implanted at RT and 600 °C. Similar behavior is found for both samples, but the relative PL increase is higher for the sample implanted at RT.

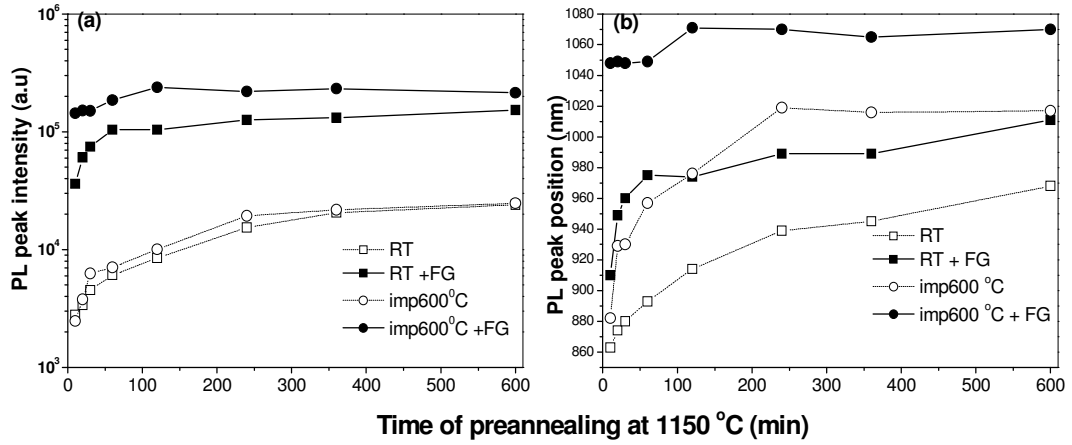


Fig. 2 – (a) PL peak intensity and (b) position as a function of the annealing time at 1150 °C, before (open symbols) and after (full symbols) passivation process in FG of samples implanted at RT (squares) and at 600 °C (circles).

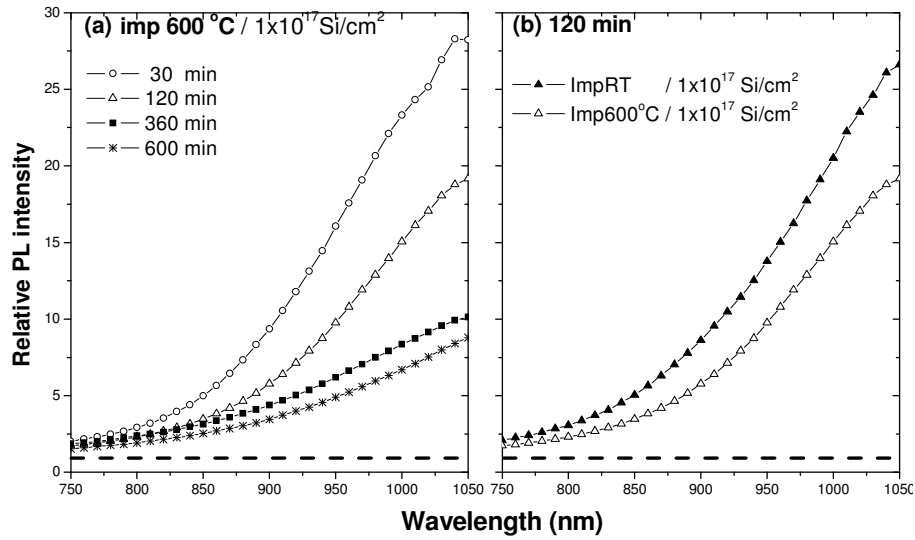


Fig. 3 – Ratio between the intensities of the PL spectra after and before the passivation process. a) Samples implanted at 600 °C preannealed for different time intervals at 1150 °C. b) Samples implanted at RT and at 600 °C preannealed for 120 min at 1150 °C.

It is important to point out that the FG passivation does not produce any modification in the Si NCs size distribution [19], as it was confirmed by TEM analysis. Furthermore, thermal treatment of the FG-passivated samples at 1150 °C brings the PL spectra to the original shape before the passivation process.

4. Discussion and conclusions

It was already reported [11] that the passivation of non-radiative states and/or defects at the Si NCs/matrix interface is an efficient method to increase the PL intensity of the Si NCs without changing the original mechanism of the PL emission.

At first sight, the strong change on the PL shape observed in fig.1 could be attributed to some variation in the nanocrystals size after the passivation process. However, TEM observations (not shown here) demonstrate that the FG treatment does not change the characteristics of the Si NCs distribution. Then, it should be concluded that the FG treatment transforms a sizeable quantity of large non-radiative nanocrystals in radiative ones. Since large nanocrystals have extensive surfaces, they can present a significant number of defects that quench their possible PL emission. This feature explains why the 1000 nm PL band increases its intensity and simultaneously suffers a strong redshift.

On the other hand, it is known [14] that the pre-annealing time at high temperature (1150 °C in this work) not only contributes to the Si NCs formation, but also improves the interface quality between the nanocrystals

and the matrix. This feature explains the results displayed in fig. 2. However, as also shown by the same figure the further FG treatment is by far more efficient than just a long annealing time performed at 1150 °C.

By taking the ratio between the spectra of passivated samples in relation to non-passivated ones (fig. 3), we observe that the relative PL increasing is a factor of two for the spectrum region around 800 nm and suffers a progressive increasing with λ . Such relative PL change in the long wavelength region is higher for samples that were annealed by short annealing times previous to the passivation process. This is an indication that the nanocrystals present in the samples annealed by short times after implantation are subject to contain more non-radiative defects. Through the same procedure mentioned above, we have compared samples implanted at RT with hot-implanted ones, annealed under the same conditions. It can be observed that the relative PL increase is higher for the sample implanted at RT, which is an indication that nanocrystals grown in hot-implanted samples have less non-radiative interface defects.

5. References

- [1] L. Pavesi, L. Dal Negro, C. Mazzoleni, G. Franzò, and F. Priolo, *Nature (London)* 408, 440 (2000).
- [2] A. T. Fiory and N. M. Ravindra, *J. Electron. Mater.* 32, 1043 (2003).
- [3] L. Brus, in *Semiconductors and Semimetals*, edited by D. Lockwood (Academic, New York, 1998), Vol. 49, p. 303.
- [4] M. L. Brongersma, A. Polman, K. S. Min, E. Boer, T. Tambo, and H. A. Atwater, *Appl. Phys. Lett.* 72, 2577 (1998).
- [5] J. Heitmann, F. Müller, L. Yi, and M. Zacharias, *Phys. Rev. B* 69, 195309 (2004).
- [6] P. M. Fauchet, *Materials Today* 8, 6 (2005).
- [7] T. Shimizu-Iwayama, D. E. Hole, and I. W. Boyd, *J. Phys.: Condens. Matter* 11, 6595 (1999).
- [8] M. Zhu, Y. Han, R. B. Wehrspohn, C. Godet, R. Etemadi, and D. Ballutaud, *J. Appl. Phys.* 83, 5386 (1998).
- [9] X. Wu, A. M. Bittner, K. Kern, C. Eggs, and S. Veprek, *Appl. Phys. Lett.* 77, 645 (2000).
- [10] A.R. Wilkinson and R. G. Elliman, *Phys. Rev. B*, 68, 155303(2003).
- [11] M. Lannoo, C. Delerue and G. Allan. *J. Lumin.* 70, 170 (1996).
- [12] E. Neufeld, S. Wang, R. Apetz, Ch. Buchal, R. Carius, C. W. White and D. K. Thomas. *Thin Solid Films* 294, 238 (1997).
- [13] K. S. Min, K.V Shcheglov, C. M Yang, Harry A. Atwater, M. L. Brongersma and A. Polman. *Appl. Phys. Lett.* 69, 2033 (1996).
- [14] M. López, B. Garrido, C.García, P. Pellegrino, A. Pérez-Rodríguez; J. R Morante, C. Bonafos, M. Carrada and A. Claverie. *Appl. Phys.Lett.* 80, 1637 (2002).
- [15] S.P. Withrow, C.W. White, A. Meldrum, et al., *J. Appl. Phys.* 86, 396 (1999).
- [16] S. Cheylan and R. G. Elliman, *Appl. Phys. Lett.* 78, 1225 (2001).
- [17] U.S. Sias, L. Amaral, M. Behar and H. Boudinov, E.C. Moreira and E. Ribeiro, *J. Appl. Phys.* 98, 34312 (2005).
- [18] M.Y. Valahnk, V. A. Yukhimchuk, V. Y. Bratus, A. A. Konchits, P. L. F. Hemment and T. Komoda. *J. Appl. Phys.* 85, 168 (1999).
- [19] Y. Q.Wang; R.Smirani; G. G Ross. *Physica E*, 23, 97 (2004).

The Ionizing Dose Effect in a Two-stage CMOS Operational Amplifier

^{1,2}Ulisses Lyra dos Santos, ²Luís Cléber C. Marques, ¹Gilson I. Wirth
{ulisses.cleber}@cefetrn.br, wirth@ece.ufrgs.br

¹Universidade Federal do Rio Grande do Sul - UFRGS

² Instituto Federal Sul-Rio-Grandense - IF

Abstract

This work aims to design an operational amplifier to AMS.35p technology and verify the influence of the total ionizing effect (due to radiation) in the amplifier performance. The influence of the dose effect in the transistors was modeled using random vectors, one for each transistor, with a Gaussian distribution, where the threshold voltage (V_{th}) was changed, cumulatively, up to the point of non-operation of the circuit. Circuit performance degradation could be observed, and it was possible to identify the transistor type (NMOS or PMOS) that most contributes to performance degradation under the total dose effect considered – oxide with positive charge excess. All simulations were run with spice type SMASH simulator, and the vectors were created in Matlab.

1. Introduction

In CMOS circuits under radiation, the dose effect generates pairs of free electrons and holes in the oxide layer (SiO_2). The free electrons and the holes are influenced by electric field induced in the oxide by the gate voltage. The free electrons, for their high mobility, are drained off the oxide. But the holes, with their lower mobility, are partially trapped in the oxide. After radiation, a high positive charge stays in the oxide, which has the same effect of a positive voltage applied to the gate. In fact, this effect causes a change in the threshold voltage (V_{th}) of the devices. It is important to notice that the radiation effect is cumulative and it can get to a point where it is no longer necessary a pulse in the gate for the transistor to be on.

This work is organized as follows. In section 2, it is shown the design of an operational amplifier (op-amp) to be used as test device under radiation. In section 3, the dose effect is tested in the op-amp, firstly in all transistors and then only in NMOS devices only and in PMOS devices only, aiming to verify performance degradation. For the tests, the threshold voltage of the devices is changed. Finally, section 4 shows the conclusions.

2. Design of a two-stage CMOS operational amplifier

Fig. 1 shows a two-stage compensated operational amplifier, also known as a Miller transconductance amplifier. The designed amplifier presents P-type differential inputs and a second stage with a N-type drive transistor. This aims to maximize the slew-rate and to minimize the $1/f$ noise [2].

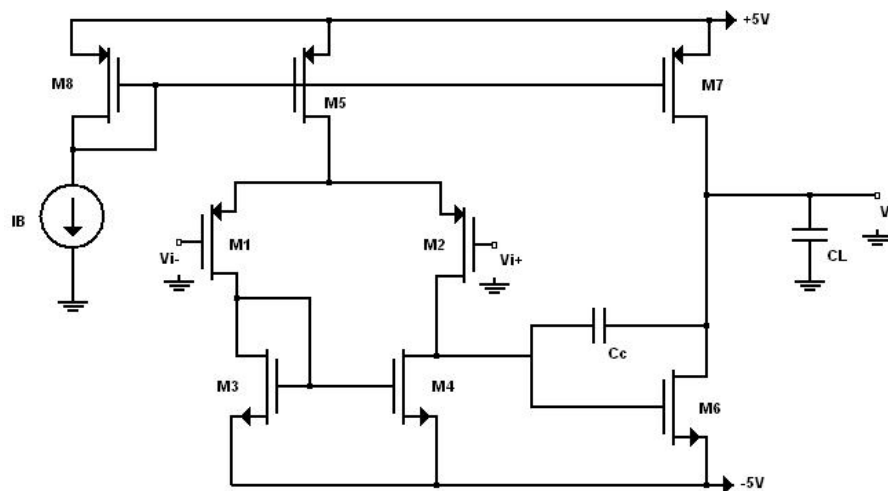


Fig. 1 – Miller Amplifier

To the design were considered the following circuit specifications:

- Voltage supply of $\pm 3.3V$
- GBW of 1MHz
- Phase Margin (PM) of 60°
- DC gain of 60dB
- Charge capacitance of 20pf

The software used in simulations was the spice-type Smash 5.2.1 simulator, from Dolphin Integration [1]. The design was done to AMS.35p technology, considering strong inversion and ACM MOSFET model equations [2].

2.1. Compensation capacitance calculation

To a good phase margin [3]:

$$0,2C_L \leq C_C \leq 0,6C_L$$

where C_L is the load capacitance.

2.2. First stage

The transconductance of the differential transistors is a function of GBW and C_C [3]:

$$g_{m1} = 2.\pi.GBW.C_C$$

The bias current can be obtained through the universal relation to MOSFETs [2], $\frac{I_F}{\phi_t \cdot g_{ms}} = \frac{1 + \sqrt{1 + i_f}}{2}$,

which, in saturation, can be written as [2]:

$$\frac{I_F}{\phi_t \cdot g_m} = \frac{1 + \sqrt{1 + i_f}}{2.n}$$

where n is the slope factor, considered 1.3, i_f is the current density, made $i_f=100$ (strong inversion) [2], and ϕ_t is the thermal voltage, $\phi_t = 25,9 \times 10^{-3}$ at $27^\circ C$ temperature, for all transistors. With this bias current are calculated transistors dimensions (M1, M2, M3, M4, M5, M8), with [2]:

$$\frac{W}{L} = \frac{I_F}{i_f \cdot I_{SQ}}$$

The currents I_{SQ} are dependent of the technology [2]:

$$I_{SQN} = \mu_N \cdot C'_{OX} \cdot n \cdot \frac{\phi_t^2}{2}$$

$$I_{SQP} = \mu_P \cdot C'_{OX} \cdot n \cdot \frac{\phi_t^2}{2}$$

2.3 Second stage

The second stage design must take into account the PM specification. The zero, which is in the right side semi-plan, must be positioned well above the GBW (about a decade). Doing that, we are also positioning the second pole, because C_C was fixed as a fraction of C_L [3]:

$$f_{zero} = \frac{g_{m6}}{2\pi C_C} \quad \text{and} \quad f_{p2} = \frac{g_{m6}}{2\pi C_L}$$

Given that $GBW \cong \frac{g_{m1}}{2\pi C_C}$, the ratio $\frac{g_{m6}}{g_{m1}}$ determines the ratio between f_{zero} and GBW:

$$\frac{g_{m6}}{g_{m1}} = \frac{f_{zero}}{GBW}$$

It was considered $g_{m6} = 10 \times g_{m1}$. Thus, the zero is a decade above GBW. With this g_{m6} is then found I_6 . Thus it is possible to calculate the dimensions of M6 and M7, through the relation [3]:

$$\frac{\left(\frac{W}{L}\right)_7}{\left(\frac{W}{L}\right)_5} = \frac{1}{2} \frac{\left(\frac{W}{L}\right)_6}{\left(\frac{W}{L}\right)_4}$$

After properly sizing all transistors, the SMASH simulation was performed. As can be seen in Fig. 2, the operational amplifier is working according to the specifications, with a DC gain of about 96dB and a PM of about 64°.

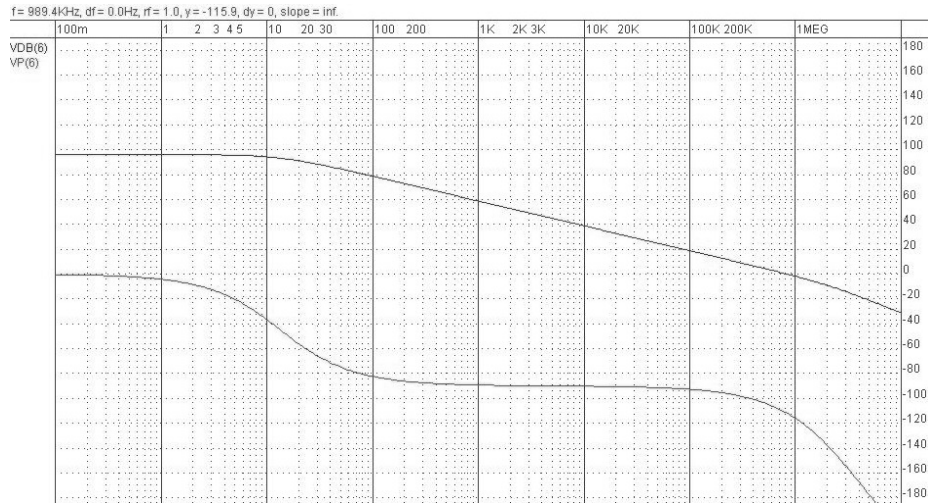


Fig. 2 – Dimensions of the transistors altered maintaining relationship W/L and C_C 4pf.

3. Total ionizing dose effect in the designed op-amp

The dose effect happens due to the accumulation of ionizing radiation with time, leading to a malfunction of the device [4]. It is measured in rads (*radiation absorbed dose*) [4].

To verify the dose effect in the designed op-amp, changes were made in the V_{th} of the devices, firstly in all the transistors and afterwards only in the N-type transistors and also only in the P-type transistors.

The V_{th} 's were changed through vectors of random numbers with Gaussian distribution, with zero average value and standard deviation equals to 1, been the values between 0 and 1. The vectors sizes were changed according to the type of analysis that was performed and the module in each vector value was normalized according to the need. The values of each generated vector were added to the original V_{th} (V_{th} N-type = 0,4655, and V_{th} P-type = -0,617) of the transistors library. The changes were made cumulatively, up to the point

of malfunction of the device, being the new values of the V_{th} for the transistors N-type and P-type respectively: -0,27484 and -1,35734 .

4. Conclusions

The dose effect can change the threshold voltage of a MOS device. Thus, the V_{th} 's were changed in different modes, firstly in all transistors of the operational amplifier. Thus it was verified a fast degradation of circuit performance, mainly in the circuit gain. The changes were made cumulatively, up to the point of malfunction of the device.

In a second moment, changes were made only in the N-type transistors. The degradation was a little lower as compared to the first case, but getting to the point of malfunction of the circuit practically at the same point.

Finally, changes were made only in the P-type transistors. In this case, the degradation in the circuit performance was much slower than in the first two cases.

We can conclude that the dose effect, making the oxide positively charged, has a much higher influence in NMOS transistors than in PMOS transistors.

As future work, other parameter variations could be measured, like slew-rate and common mode input voltage. Also, it can be verified mismatch effects among the transistors, mainly in the differential pair.

5. References

- [1] http://www.dolphin.fr/product_offering.html, accessed in January de 2009.
- [2] L. C. C. Marques, "Técnica de Mosfet Chaveado Para Filtros Programáveis Operando à Baixa Tensão de Alimentação", Tese de doutorado, Florianópolis-SC 2002.
- [3] Allen, Philip E.; Holberg Douglas R., Cmos Analog Circuit Design, Holt, New York 1987.
- [4] Da Silva, Vitor César Dias, Structures resistant CMOS to the radiation using conventional production processes, Theory of M.Sc., Military Institute of Engineering, Rio de Janeiro, RJ, Brasil, 2004.
- [5] Sedra, Adel S.; Smith, Kenneth, C., Microeletrônica 4^o ed., Pearson Makron Books, São Paulo 2000.
- [6] Velasco, Raul; Foulilat, Pascal; Reis, Ricardo, Radiation Effects on Embdedded System, Springer, 2007
- [7] C. Galup-Montoro, A. I. A. Cunha, and M. C. Schneider, "A current-based MOSFET model for integrated circuit design," in Low-voltage low power integrated circuits and systems, E. Sánchez-Sinencio and A. G. (eds), IEEE press, Piscataway, 1999.

Verification of Total Ionizing Dose Effects in a 6-Transistor SRAM Cell by Circuit Simulation in the 100nm Technology Node

^{1,2}Vitor Paniz, ²Luís Cléber C. Marques, ¹Gilson I. Wirth
{Vpaniz,Cleber}@ceftrs.tche.br, wirth@ece.ufrgs.br

¹Universidade Federal do Rio Grande do Sul - UFRGS

²Instituto Federal Sul-Rio-Grandense - IF

Abstract

This paper aims to verify the behavior of a SRAM cell exposed to radiation, specifically with respect to the total ionizing dose effect. The influence of the dose effect in the transistors was modeled using random vectors, one for each transistor, with a Gaussian distribution, where the threshold voltage (V_{th}) was changed in all transistors. The CMOS memory cell is a static 6-transistors cell, designed with minimal size, using the eecsberkeley100nm technology. It was verified, through simulations, the cell behavior with respect to V_{th} changes. The results are partial ones because it was only considered the radiation total ionizing dose effect in the response times, and the tests were performed considering only writing in the memory cell. For all simulations the Spice3 simulator was used, and the random vectors were created in Matlab.

1. Design of a 6-transistor CMOS memory cell

Fig. 1 shows the schematic of a memory cell. The cell is composed of a latch, transistors Q1 to Q4, and two bypass transistors, Q5 and Q6. Its behavior is described as follows:

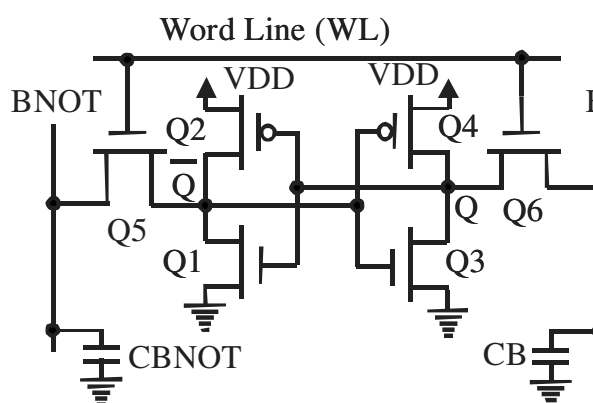


Fig. 1 – Schematic of a memory cell.

Writing process:

The logic value to be written at node Q must be set in column B. If, for instance, a zero logic level (0) is to be written, this logic level is put in B while a one logic level (1) is put in BNOT. Next, the word line WL is set to 1, which makes Q5 and Q6 to conduct. Notice that the 0 in B is applied to the gates of Q1 and Q2, making Q2 to conduct and Q1 to cut-off. At the same time, the 1 in BNOT is applied to the gates of Q3 and Q4, making Q3 to conduct and Q4 to cut-off. After stabilization, the WL signal can be disabled and Q will hold a 0, while QNOT will hold a 1.

Reading process:

To a faster reading process, a pre-charge is performed in B and BNOT columns, normally with $\frac{1}{2}$ of VDD. Consider the case where the Q output holds a 0 and the QNOT output holds a 1. After the pre-charging, WL is set to 1, making Q5 and Q6 to conduct. Then, Q will start to discharge the capacitance CB through Q3 and Q6 while, at the same time, CBNOT will be charged (from $\frac{1}{2}$ VDD) through Q2 and Q5. To complete the process, it is necessary a sense amplifier (not shown), which accelerates the reading process.

2. Radiation effects in the MOSFETs

Historically, the main effect of ionizing radiation on MOS devices is the variation of the threshold voltage and degradation of the inversion layer mobility. This is caused by the electrostatic effect of the interface traps and by the exchange of interface charges. Even though the radiation can be little significant in the threshold

voltage, care must be taken in upcoming CMOS technologies. Those present thin oxide layer, where radiation can affect the insulation oxide, resulting in high leakage currents [1].

The MOS transistor is an active device that can control the current flow between *source* and *drain* terminals. In digital circuits, they are generally used as switches, which can be open or closed, depending on the voltage applied between *gate* and *bulk*. For instance, when a certain voltage (higher than V_{th}) is applied to the gate terminal of a NMOS transistor (referred to the bulk), the transistor will be conducting; when the gate voltage is below V_{th} , the transistor will be cut-off, considering the strong inversion model [2].

The threshold voltage depends on the design of the device and on the material used but, generally, its value is between 10 to 20% of VDD [3].

The oxide that insulates the gate of the channel is made of silicon dioxide (SiO_2). When the device is exposed to radiation, problems can occur. First, the oxide gets ionized by the dose it absorbs, i. e., pairs free electrons-holes are generated. The free electrons and the holes are influenced by electric field induced in the oxide by the gate voltage. The free electrons, for their high mobility, are drained off the oxide. But the holes, with their lower mobility, are partially trapped in the oxide. After the radiation dose, there stays a high positive charge, which have the same effect of a positive voltage applied to the gate. Depending on the radiation amount, the device conducts even without any voltage applied to the gate [4]. At this point, the drain-source current cannot be controlled by the gate anymore, and the transistor gets permanently on (conducting). In the PMOS transistor there is a similar effect, but with opposite result, i.e., the transistor can get permanently off.

2.1. Threshold voltage variation

This is the most important effect, given that the threshold voltage is the gate voltage needed to create the inversion layer (the channel) and thus turn on the transistor. Variations in V_{th} can change the circuit operation and, depending of the dose, they can prevent the circuit from working [4].

The accumulation of positive charge (holes) trapped in the oxide creates a vertical electric field in the oxide surface, attracting electrons to the interface Si/SiO_2 (channel region). These electrons help to decrease the positive charge concentration next to the substrate surface, modifying the balance between the charge carriers in the doped silicon. This decreasing of positive charges in the substrate surface makes easier to achieve the inversion layer, thus decreasing the threshold voltage of the NMOS transistor [4]. In the PMOS devices, the effect is opposite, and the gate voltage must be more negative to compensate the higher amount of negative carriers in the channel.

In extreme levels of radiation, it becomes impossible turn on PMOS transistors and turn off NMOS transistors.

In the present technologies, with smaller minimum dimensions, the transistors become less susceptible to radiation, due to thinner thickness and better quality of the oxide layer [4], being this better quality due the higher intensities of the electrical field that the transistor must support. Thus, the new technologies are expected to be less sensitive to total radiation dose effects that the older ones [4].

2.2. The ionizing dose effect in the memory cell

To visualize the ionizing dose effect in the memory cell, changes were made in the threshold voltages, using random vectors with Gaussian distribution. The V_{th} 's were changed, cumulatively, in all transistors, subtracting the values of the Gaussian distribution in the V_{th} values, whose initial values used were -0,303V to the PMOS device and 0,2607V to the NMOS device. The obtained results are partial ones because it is only being considered the total ionizing dose effect, leaving out the other radiation effects. Also, tests were only performed in the writing process.

In the graphs of Fig. 2, it can be observed the default behavior of the memory cell. In Fig. 3 is shown one of the graphs obtained with V_{th} 's variations. In this graph, the transistors had their V_{th} 's changed to 0.179206, -0.42033, 0.174522, -0.43794, 0.18999 e 0.14457, respectively, to Q1, Q2, Q3, Q4, Q5 and Q6.

When comparing Fig. 2b with Fig. 3, it can be noticed the variation in the writing delay, from 0.00268ns (default V_{th} 's) to 0.00227ns.

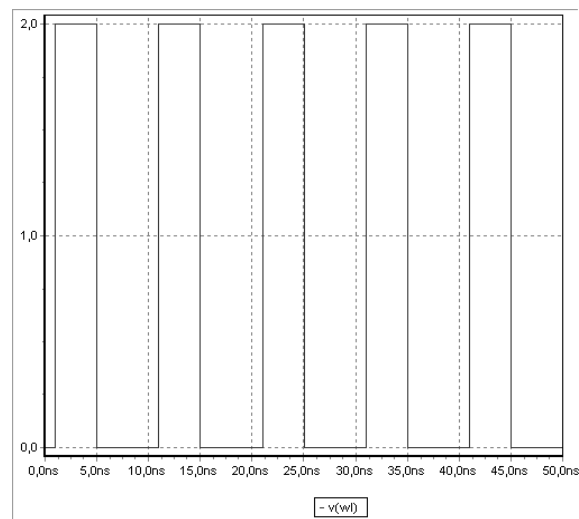


Fig. 2a – WL signal.

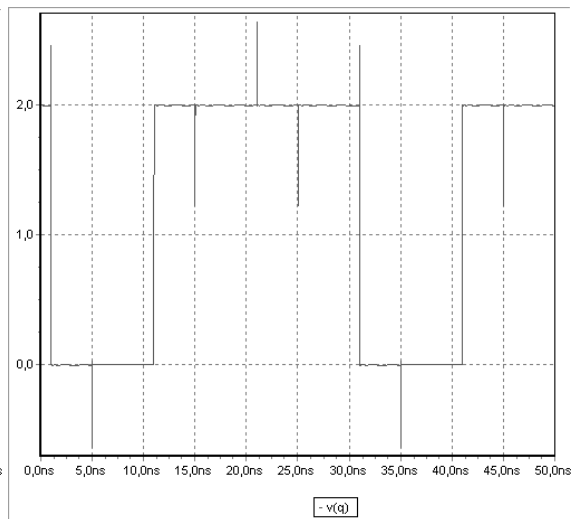


Fig. 2b – Q signal.

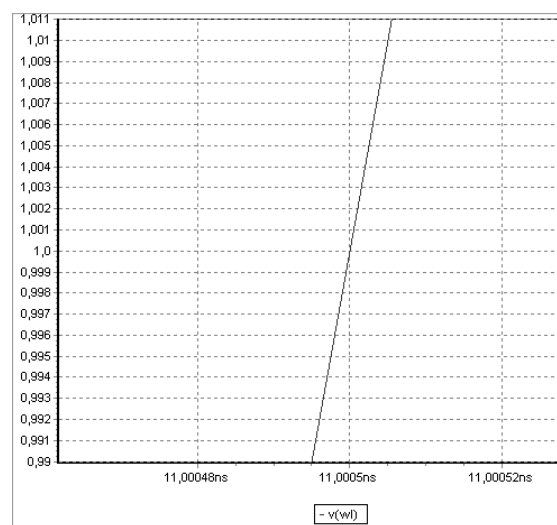


Fig. 2c – Zoom in rise of WL signal.

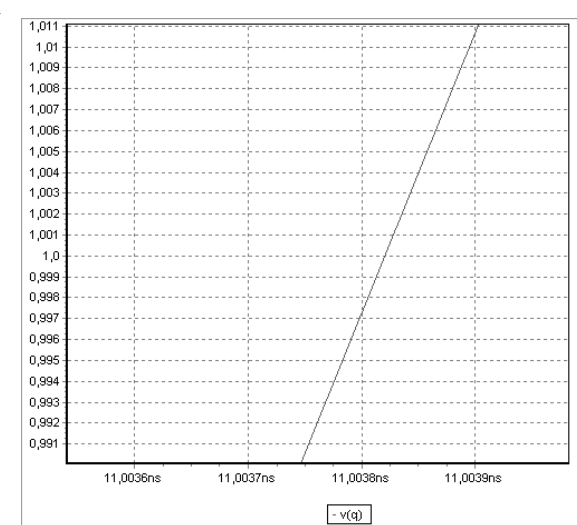


Fig. 2d – Zoom in fall of WL signal.

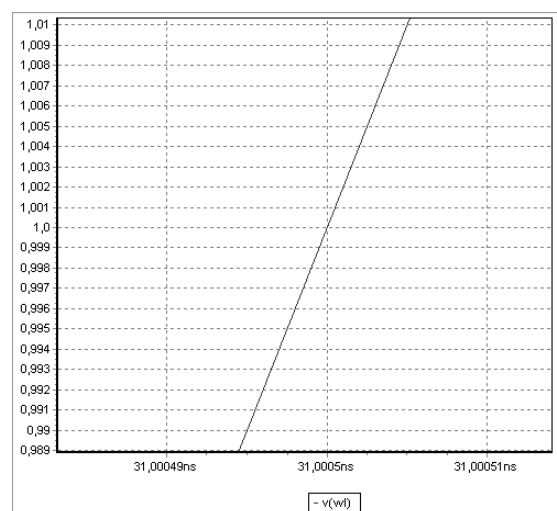


Fig. 2e – Zoom in rise of Q signal.

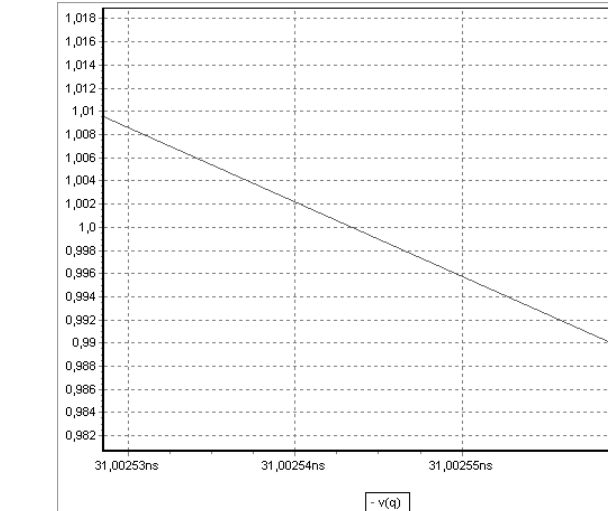


Fig. 2f – Zoom in fall of Q signal.

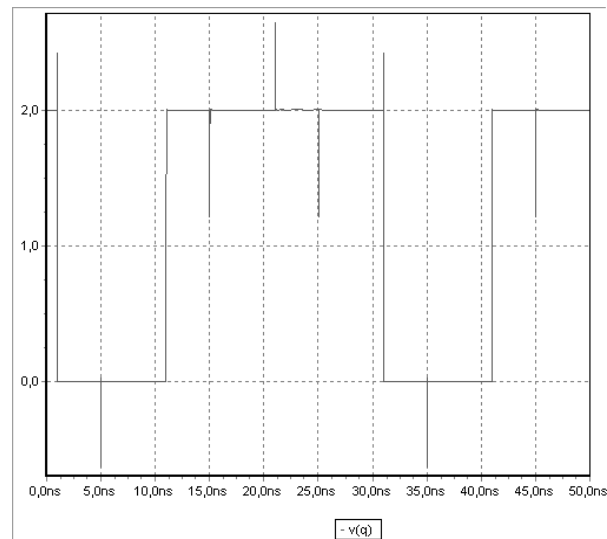


Fig. 3 - Graphs obtained with V_{th} 's variations.

3. Conclusions

The simulation were performed considering the total ionizing dose effect, which is cumulative [4] and, in extreme cases, will make the NMOS transistors to be permanently on and the PMOS transistors permanently off. The simulations are considering only the dose effect in the time response of the memory cell, not considering other harmful effects that radiation causes. We could observe a decrease in the time response of the cell, but that cannot be considered a better working of the cell, due to the effects neglected. Making further tests, with modifications from the random vectors, it was observed that the memory cell worked until negative values were obtained to the V_{th} 's of the NMOS transistors, making them to conduct even without a positive gate voltage.

4. References

- [1] R. Velasco, P. Foulliat, R. Reis, Radiation Effects on Embedded System, Springer, 2007.
- [2] L. C. C. Marques, "Técnica de Mosfet Chaveado Para Filtros Programáveis Operando à Baixa Tensão de Alimentação", Tese de doutorado, Florianópolis-SC 2002.
- [3] Adel S. Sedra, Kenneth C. Smith, Microeletrônica, 4^o ed., Pearson Makron Books, São Paulo, 2000.
- [4] V. C. Dias da Silva, "Estruturas CMOS resistentes à radiação utilizando processos de fabricação convencionais", Tese de M.Sc., Instituto Militar de Engenharia, Rio de Janeiro, RJ, Brasil, 2004.

Delay Variability and its Relation to the Topology of Digital Gates

Digeorgia N. da Silva, André I. Reis, Renato P. Ribas
{dnsilva, andreis, rpribas}@inf.ufrgs.br

Nangate Research Lab - Instituto de Informática, UFRGS

Abstract

The aggressive shrinking of MOS technology causes the intrinsic variability present in its fabrication process to increase. Variations in the physical, operational or modeling parameters of a device lead to variations in electrical device characteristics, what therefore causes deviations in performance and power consumption of the circuit. The purpose of the work is to characterize some digital gates implemented in different topologies according to the variability in their process parameters, such as gate channel length and width, oxide thickness, and ion doping concentration that may cause the threshold voltage to be different from that expected. The results obtained may be used as guidelines for implementing transistors networks that are more robust to parameters variability.

1. Introduction

Any manufacturing process presents a certain degree of variability around the nominal value given by the product specifications. This phenomena is accounted for in the description of the product characteristics in the form of a window of acceptable variation for each of its critical parameters. Regarding MOS fabrication process, variation effects do not scale proportionally with the design dimensions, causing the relative impact of the critical dimension variations to increase with each new technology generation [1]. Variations in the physical, operational or modeling parameters of a device lead to variations in electrical device characteristics, such as the threshold voltage or drive strength of transistors. Finally, variations in electrical characteristics of the components lead to variations in performance and power consumption of the circuit.

Different logic styles result in transistor networks with different electrical and physical behavior and there are different types of circuits that can be used to represent a certain logic function. The impact of parameters variations of a cell on its metrics is not the same for implementations in different logic styles. Also, different cells topologies even for the same logic style result in different variability behavior.

The traditional design efforts guided only by the best, worst and the nominal case models for the device parameters over- or underestimate the impact of variation. Conventional sizing tools size the logic gates to optimize area and power consumption while meeting the desired delay constraint [2]. Static timing analysis is usually used to find the critical points of the circuit that affect the critical path delay. The transistor widths are then sized to meet the desired delay constraint while keeping the power consumption and area within a limit. However, due to random process parameter variation, a large number of chips may not meet the required delay. Choi [3] proposes a statistical design technique considering both inter- and intra-die variations of process parameters. The goal is to resize the transistor widths with minimal increase in area and power consumption while improving the confidence that the circuit meets the delay constraint under process variations. Rosa [4] presents some methods to evaluate different network implementations, in relation to their performance, area, dynamic power and leakage behavior without considering parameters variations.

The purpose of this work is to analyze how the variability in the parameters affects the performance of the gates according to (i) their topology (transistor network arrangement) and logic style, and (ii) the position of the transistor with a transient input signal in relation to the output node. Also, complex logic gates are analyzed in comparison to the use of basic gates to implement the same logic function. These data provide us guidelines to the development of topologies that can be more immune to variability than others.

The paper is organized as follows. Section 2 describes the methodology applied in order to study the performance variability of logic gates implemented in different topologies. Section 3 shows the results achieved and some discussion on them, and section 4 concludes the work.

2. Methodology

Electrical timing simulations of the logic gates are performed with HSPICE, a standard circuit simulator that is considered the “golden reference” for electric circuits. The parameters L (channel length), W (channel width), T_{ox} (oxide thickness), V_{th} (threshold voltage) and N_{dep} (doping concentration of the substrate) are varied and timing measurements (delay) are taken. For each gate, nominal delay is compared to the results acquired

when the parameters suffer variations. The technology node used in this work is 45 nm and the model file is a Predictive Technology Model (PTM) [5] based on BSIM4.

This paper presents some analysis performed on basic logic gates in order to find out how the parameters variations impact the performance of the cells. Rise and fall delays of different gates – inverter, 2-, 3- and 4-input NAND, 2-, 3- and 4-input NOR besides a 2-input XOR gate – were measured in a corner-based methodology. The measurements were taken for values of the process and electric parameters that suffered variations from 1 to 10% of their nominal quantities and only positive discrepancies were considered. It must be emphasized that no absolute delay value is presented here, only the variability of the delay is compared and analyzed. Also, no spatial correlation between the parameters were taken into account, what means that a device placed in the vicinity of another may come up with different variations in the parameters. Although there are correlations between some process parameters and threshold voltage, these correlations were not considered here and each parameter was set to vary at a time.

3. Results

3.1. Inverter

The ramp signal used as the input for the inverter in this analysis (input transition time $\neq 0$) is the output of a small chain of inverters (two minimum-sized gates) that receives a step signal at its input. The fan-out is a minimum-sized inverter. The first set of simulations were run for a minimum-sized inverter in which the channel length (L) of the transistors equals the gate width (W), so $L_n = L_p = W_n = W_p$. Simulations were also performed for the case $W_p = 2*W_n$. Fig. 1 and 2 show the percentage of rise and fall delay variation for a minimum-sized inverter.

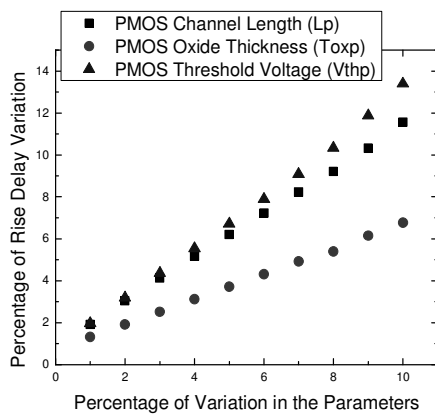


Fig. 1 – Percentage of rise delay variation of an inverter versus the percentage of variation in the parameters L_p , Tox_p , and Vth_p for $W_p = W_n$.

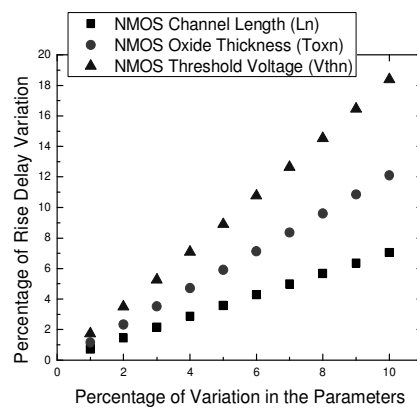


Fig. 2 – Percentage of fall delay variation of an inverter versus the percentage of variation in the parameters L_n , Tox_n , and Vth_n for $W_p = W_n$.

The parameters that mostly impact the rise delay of the propagated signal are L_p , Tox_p and Vth_p (PMOS). In the case of the fall delay, variations in L_n , Tox_n and Vth_n (NMOS) have reasonable impact. The variations on delay presented linear dependence on the parameters variations. The measurements of both rise and fall delay showed that when $W_p = W_n$ the performance of the inverter is more sensitive to variations in its parameters than for $W_p = 2*W_n$.

The sizing of the gates studied in the rest of the paper was performed in a way to keep similar current drive strengths of NMOS and PMOS transistors networks.

3.2. 2-Input NAND and NOR Gates

Rising- and falling-edge output signals go through essentially different paths in NAND and NOR gates. In the former, the series transistors are in the pull-down network and this array is responsible for a falling-edge output signal. In the latter, the series transistors are in the pull-up network and are responsible for a rising-edge output. The comparison between the influences of the variations in the parameters on NOR and NAND gates delays is physically more correct if one considers equivalent array of transistors: series-to-series or parallel-to-parallel. In this case, the fall delay variations of the NAND gate are compared to the rise delay variations of the NOR gate and vice-versa. The analysis is performed on the switching of only one transistor of the arrays (series

and parallel) at a time. Fig. 3 shows the variation on the delay of the signal that propagates through the parallel networks of 2-input NAND and NOR gates. Fig. 4 and 5 show variations on delay of signals through series array of transistors for transitions applied far from and close to the output node respectively.

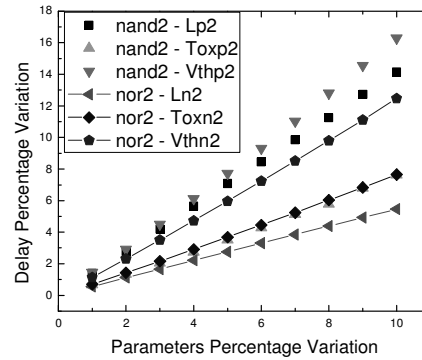


Fig. 3 – Delay variations (%) for the parallel transistors networks of 2-input NOR and NAND gates, considering variations in the channel length, oxide thickness and threshold voltage of the parallel transistors.

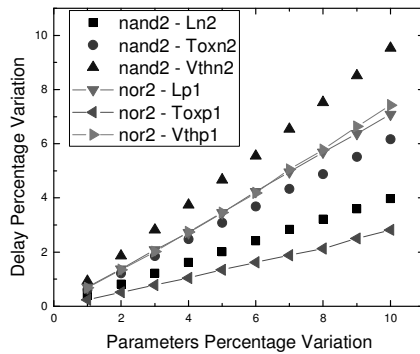


Fig. 4 – Delay variations (%) for the series transistors networks of 2-input NOR and NAND gates with a signal transition applied far from the output node.

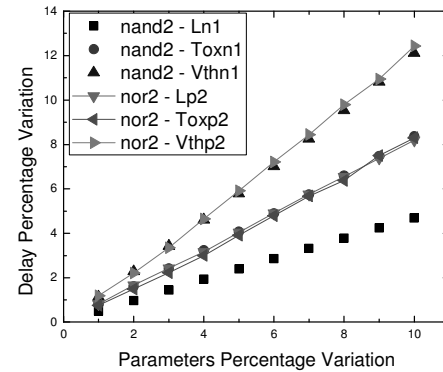


Fig. 5 – Delay variations (%) for the series transistors networks of 2-input NOR and NAND gates with a signal transition applied close to the output node.

The channel length of the PMOS transistors in the NOR gate impacts the delay more than the channel length of the NMOS transistors in the NAND, and the opposite happens to the oxide thickness variations: changes in T_{OX} of the NMOS transistor in the NAND gate cause higher discrepancies in the delay than changes in T_{OX} of the MOS in the NOR gate. The delay variations of the NOR gate depend strongly on variations in both series transistors and not only on variations in the transistor that is switching. The delay variations in the NAND gate depend mainly on the transistor in which the transient signal is being applied.

3.3. 3-Input NAND and NOR Gates

For a transient input applied close to the output node, variations in the oxide thickness and threshold voltage of a NAND gate have about twice the impact on the delay than variations in the same parameters of a NOR gate. The effect of the variations in the channel length is similar for both topologies.

3.3.1. Series Array with Transient Input Signal Applied to the Middle Transistor

Under this condition, the NOR gate is sensitive to a higher number of parameters than the NAND gate. However, variations in the transistor to which the transient input is applied affect much more the delay of the NAND than of the NOR gate, especially variations in the oxide thickness and in the threshold voltage.

3.3.2. Series Array with Transient Input Signal Applied to the Transistor Far from the Output

The NAND gate delay is more strongly dependent on the variations of the transistor that is switching than the NOR gate delay. The NOR gate delay is more sensitive than the NAND gate to variations in parameters of the transistors that are not switching in this case.

3.3.3. Series Array with Transient Input Signal Applied to the Transistor Close to the Output Node

Variations in the oxide thickness and threshold voltage of a NAND gate have about twice the impact on delay than variations in the same parameters of a NOR gate.

3.4. 4-Input NAND and NOR Gates

No matter the number of inputs, NOR gates are more sensitive to variations in the channel length than NAND gates when delay measurements are considered. The opposite happens to variations in the oxide thickness and threshold voltage, cases where NAND gates are much more sensitive.

3.5. Variability Robustness Comparisons

3.5.1. N-Input NAND X (N+1)-Input NAND

The variability analysis of NAND gates designed in CMOS logic style with different number of inputs shows that the number of series transistors (NMOS network) and parallel transistors (PMOS network) impacts the delay of the cell in the presence of parameters variations. In the case of the parallel PMOS network, more robustness was presented by topologies with greater number of transistors. The opposed happens to the NMOS network – series transistors – where a stack with fewer transistors is less sensitive to variations than a topology with greater number of transistors.

3.5.2. N-Input NOR X (N+1)-Input NOR

The NOR gates showed the same tendencies as the NAND gates for PMOS and NMOS networks, though the arrangement of the networks is different. It was observed that, no matter the configuration (series/parallel) of the network, a fewer number of NMOS and a higher number of PMOS transistors present less sensitivity to variations in the parameters.

3.5.3. The Position of a Switching Transistor in a Series Transistors Network

The analysis of the series transistor networks in NAND and NOR gates showed that the position of the switching transistor in relation to the output node also influences the sensitivity of the gate to parameters variations. The best situation (higher robustness) happens when the switching transistor is as far as possible from the output and less robustness is present when the closest-to-the-output transistor switches. The results for delay variations are not the same as the results for the nominal delay. It is well known that a better timing (lower nominal delay) is achieved when a critical path signal is used as the input of transistors that are close to the output node of a gate. A trade-off is wanted since it is not interesting to have the timing of the cell with a high mean value even though it presents low variability.

3.6. Decomposition of an 3-input NAND gate into two 2-inputs NAND gates

While studying topologies with different number of inputs a question has arisen: would it be better, in terms of variability, to replace a gate with higher number of inputs by smaller gates along that implement the same logic function? Variations in the parameters of a three input NAND resulted in higher delay variability for most of the parameters variations than when two 2-Input NAND gates (plus an inverter) were used. Fig. 6 and 7 show that the 3-input gate (nand3) is much more sensitive to variations in the channel length and threshold voltage of the transistors than the combination of 2-input gates (02nand02).



Fig. 4 - A 3-input NAND gate implemented with two 2-input gates.

Though the results point out to the combination of 2-input gates as the more robust implementation, it is important to perform statistical analysis in order to assure that. By applying variations to the parameters of each gate at a time in a corner-based methodology – as used here – one may get optimistic values. On the other hand,

if the parameters of both gates are varied, pessimistic values may result. Statistical analysis on logic gates is then part of our future work.

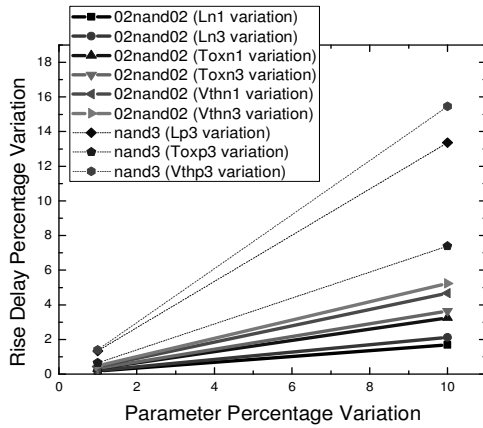


Fig. 6 - Rise delay variations (%) for a 3-input NAND and a configuration of two 2-inputs NAND gates, considering variations in the channel length, oxide thickness and threshold voltage of the transistors.

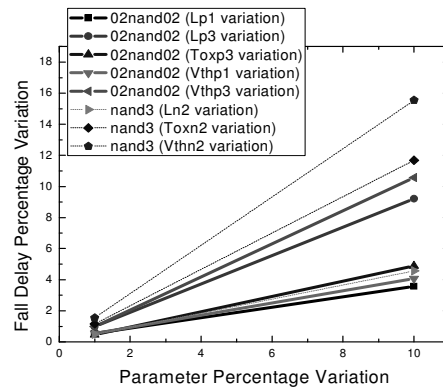


Fig. 7 - Fall delay variations (%) for a 3-input NAND and two 2-inputs NAND gates, considering variations in the channel length, oxide thickness and threshold voltage of the transistors.

3.7. 2-input XOR

A 2-input XOR was implemented in different logic styles and topologies, as shown in Fig. 8. The comparisons between delay variations of the topologies are presented in Fig. 9-12.

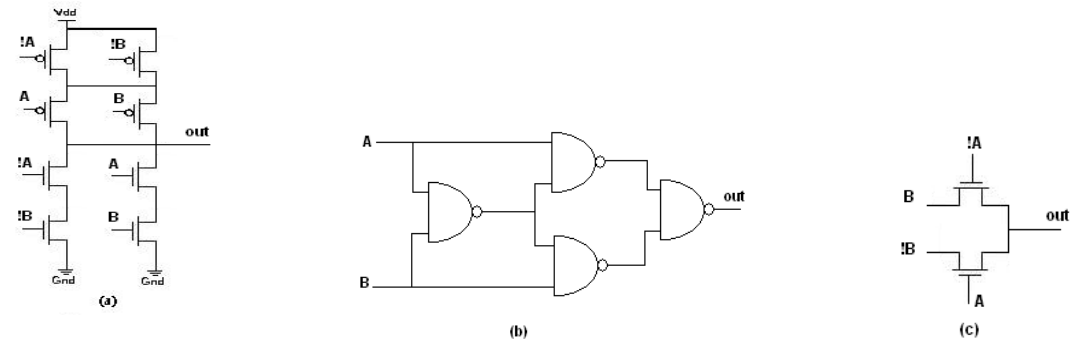


Fig. 8 - 2-input XOR implemented in different logic styles and topologies: (a) complex CMOS gate; (b) basic CMOS gates and (c) pass-transistor logic (PTL).

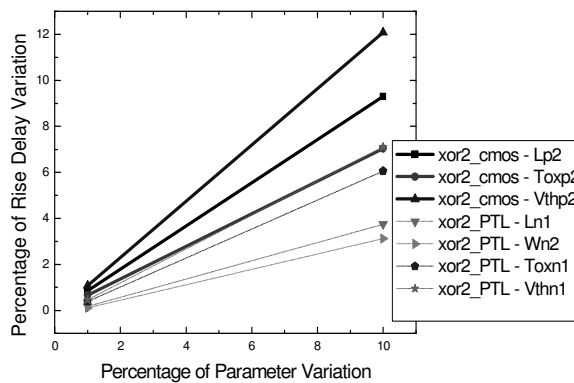


Fig. 9 - Rise delay variations (%) for a 2-input XOR implemented as a complex gate (CMOS) and in pass-transistor logic (PTL), considering variations in the channel length, channel width, oxide thickness and threshold voltage of the transistors.

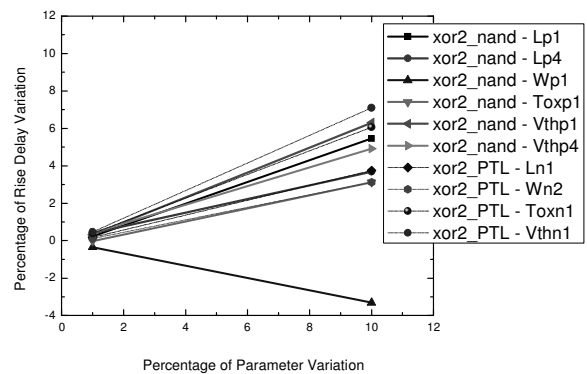


Fig. 10 - Rise delay variations (%) for a 2-input XOR implemented as a complex gate (CMOS) and in pass-transistor logic (PTL), considering variations in the channel length, channel width, oxide thickness and threshold voltage of the transistors.

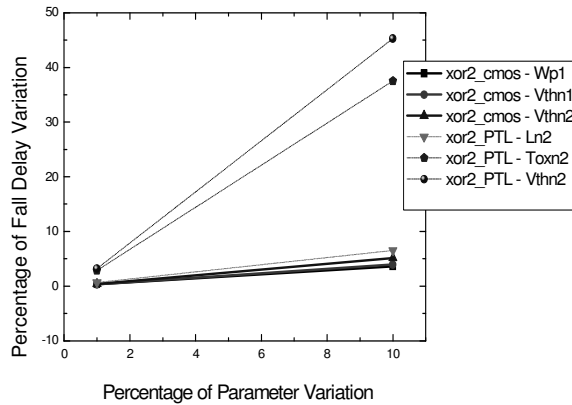


Fig. 11 - Fall delay variations (%) for a 2-input XOR implemented as a complex gate (CMOS) and in pass-transistor logic (PTL), considering variations in the channel length, channel width, oxide thickness and threshold voltage of the transistors.

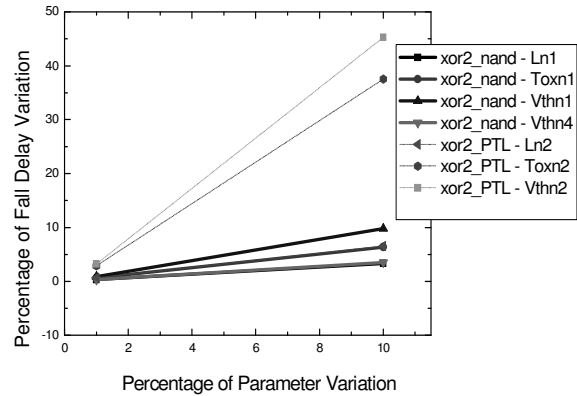


Fig. 12 - Fall delay variations (%) for a 2-input XOR implemented as a complex gate (CMOS) and in pass-transistor logic (PTL), considering variations in the channel length, channel width, oxide thickness and threshold voltage of the transistors.

The rise delay of a XOR gate implemented as CMOS complex gate is very susceptible to variations in the threshold voltage and channel length of the PMOS transistors. In a basic gate implementation, the rise delay is slightly less impacted by variations in the parameters. The fall delay of the gate in a PTL style is much more sensitive to variations in the parameters than the other implementations. In the case of implementing a 2-input XOR that is more immune to parameters variations, a configuration with basic logic gates is a better choice.

4. Conclusions

The presented results of variability in logic gates submitted to parameters variations demonstrated that the performance variability of a gate is dependent on the type of the gate, the number of inputs and on the position, in relation to the output, of a transistor in a transient state. Also, some parameters affect the gate metrics more than others. Since parameters variations are random variables, it is important to perform statistical analysis of the gates in order to compare the results achieved with the ones presented by this corner-based methodology.

5. References

- [1] K. Argawal et al., "Parametric Yield Analysis and Optimization in Leakage Dominated Technologies", *IEEE Transactions on VLSI Systems*, New York, USA, v. 15, n.6, June 2007, p. 613-623.
- [2] S. Sapatnekar, and S. Kang, *Design Automation for Timing-Driven Layout Synthesis*, Kluwer Academic Publishers, 1992, p. 269.
- [3] S. Choi, B. Paul, and K. Roy, "Novel Sizing Algorithm for Yield Improvement under Porcess Variation in Nanometer Technology," *Proc. International Conference on Computer-Aided Design*, 2004, pp. 454-459.
- [4] L. Rosa Jr., "Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles," Ph.D. Thesis, UFRGS, Porto Alegre, 2008.
- [5] Predictive Technology Model; <http://www.eas.asu.edu/~ptm>.

Design and Analysis of an Analog Ring Oscillator in CMOS 0.18 μ m Technology

Felipe Correa Werle, Giovano da Rosa Camaratta, Eduardo Conrad Junior, Luis Fernando Ferreira, Sergio Bampi
{fcwerle, grcamaratta, econradjr, lfferreira, bampi}@inf.ufrgs.br

Instituto de informática - Grupo de microeletrônica (GME)
Universidade Federal do Rio Grande do Sul
Porto alegre, Brasil

Abstract

This paper presents an analysis of differences between the results of simulations and of a ring oscillator of 19 stages designed and manufactured in a test chip with various analog and RF modules. Five-prototype chips in IBM 0.18 μ m CMOS technology were measured after fabrication. The circuit simulations were done using the Virtuoso Spectre Circuit Simulator of the Cadence™, varying several parameters such as load capacitance and resistance of the buffer, voltage, and also the statistical variation of the manufacturing process. The results show a linear increase in frequency when increasing supply voltage of the oscillator. Comparing our measured results and those obtained using Specter simulations, there was an important difference between the expected frequency of oscillation (simulated frequency: 915MHz; measured frequency: 695MHz): about 24% less. Most of this difference was due to a "debiasing" in the VDD of the circuit under test.

1. Introduction

Ring oscillators are simple circuits, commonly used in process control circuits (PCM, process control monitors). These oscillators serve as simple monitors for quality control in the production of integrated circuits. These circuits estimate errors in the process due to changes on its basic frequency. Normally used to generate controlled pulses and intervals, these circuits need to have its frequency of oscillation controlled to enable synchronization in DLLs (Delay Locked Loops), for example.

The frequency of the oscillator can be easily calculated and simulated as a function of the inverters delays, but other factors of deviation from nominal process conditions can generate a substantial difference in the frequency measured after manufacture. Correctly calculating the capacitance and resistance in the output of the oscillator prevents reduction in performance. If the resistance is greater than estimated, the charges will flow more slowly to the capacitor. Some variations may occur in the manufacturing process of the circuit, causing changes in the threshold voltage, the effective length of the channel, and the mobility of carriers in the transistors. All these result in differences between the observed frequency and the one provided by the simulation.

This work presents an analysis of the differences between the results of simulations and the measures of a ring oscillator [4], designed and developed in an analog chip-test. The chip was prototyped using the IBM 0.18 μ m CMOS technology and the service of manufacturing Multi-Project Wafer (MPW) of MOSIS. We used an encapsulation 64-pin QFN (Quad Flat no Lead), and 10 samples were encapsulated while 5 were non-encapsulated for measurement. As shown in Fig. 6, the chip has a total area of 6.8 mm², including 64 pins. The methodology used in the design of RF modules, amplifiers and oscillators (all included in the chip-testing) was based on the characteristics gm / ID, developed in previous work [1]. In this chip, several analog circuits [2] [3] and other test structures and circuits [4] were also designed and implemented, but will not be commented in this article.

We performed simulations of oscillators using the software Virtuoso Spectre Circuit Simulator of the Cadence™, varying several parameters such as capacitance and resistance of output, the power circuit and also considering changes in the manufacturing process.

2. Analysis: Schematic and extracted

The dimensions of the transistors used in the design of the ring oscillator are shown in Tab. 1. In the design of ring oscillator it is recommended to use a prime number of inverters to avoid harmonic frequencies which may distort the waveform, with the odd sub-harmonics. The dimensions of transistors influence the delay, and since the delay determines the frequency, a proper sizing is required. These values indicate a period of oscillation of 1,093ns equivalent to a mean frequency close to 915 MHz.

Tab. 1 - Sizes of Ring Oscillator and Buffer.

Width of the transistor Pmos: ($W_p = 600\text{nm}$) Nmos: ($W_n = 400\text{nm}$) $L_p = L_n = L$ minimum = 180nm
N (number of stages) = 19 inverters
Buffer in the output of oscillator with seven stages for the measurement pads.
W_p of seven stage Buffer: 900nm , $2.7\mu\text{m}$, $8.1\mu\text{m}$, $24.3\mu\text{m}$, $73.1\mu\text{m}$, $223.6\mu\text{m}$ and $1290\mu\text{m}$
W_n : 600nm $1.8\mu\text{m}$, $5.4\mu\text{m}$, $16.2\mu\text{m}$, $49.3\mu\text{m}$, $150.8\mu\text{m}$ and $870\mu\text{m}$
Buffer: $L_p = L_n = L$ minimum = 180nm

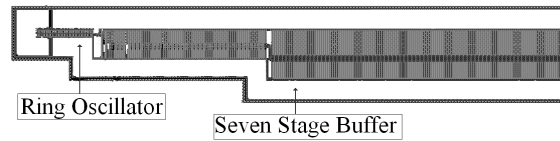


Fig. 1 – Layout with capacitance.

Using the simulator of the Cadence™ and its tools, we made the Schematic of a ring oscillator and performed various simulations to collect data of frequency of oscillation and the wave form. These data provided by the program used just ideal transistors, without parasitic capacitance or resistance; it just take into account the capacitance that we put in the output. After making the layout of the oscillator and the output buffer, we extract parasitic capacitance of the circuit (see Fig. 1).

In these simulations various parameters of the oscillator output were changed, such as the resistance, which made the signal hardly decrease its frequency. In the electrical simulation the resistance varied between 50Ω to 2500Ω , and we obtained results presented in Fig. 2(A). The solid line is the result of the post-extraction electric circuit and the dotted line is the result of the simulated circuit schematic. The range of 50Ω to 2500Ω load is connected in the Buffer, as permitted variations in the load circuit.

This simulation shows that resistances higher than previously expected alter the output in just a few hertz, around 20MHz which means 2.18% of the average frequency of oscillation. Similarly the resistance in the output also did not significantly alter the measurement of average delay of inverters, as $\pm 29.6\text{ps}$ in terms of nominal P-VDD-Temp.

The output capacitance in the circuit represents the load of a measuring instrument (frequency meters, oscilloscope, spectrum analyzers or network, etc.) and it can change the data measured in the ring oscillator. To estimate the effect of capacitance in the ring oscillator we changed the output capacitance between 50pF and 50fF (these values are similar to "unloaded" and load circuits, respectively).

What is observed with the changes in the capacitance is an increase of the delay, followed by a reduction in the amplitude of the wave (Fig. 2(B)). When comparing the frequencies obtained with different capacitance load in the buffer, we perceive that there was not a significant change in its value.

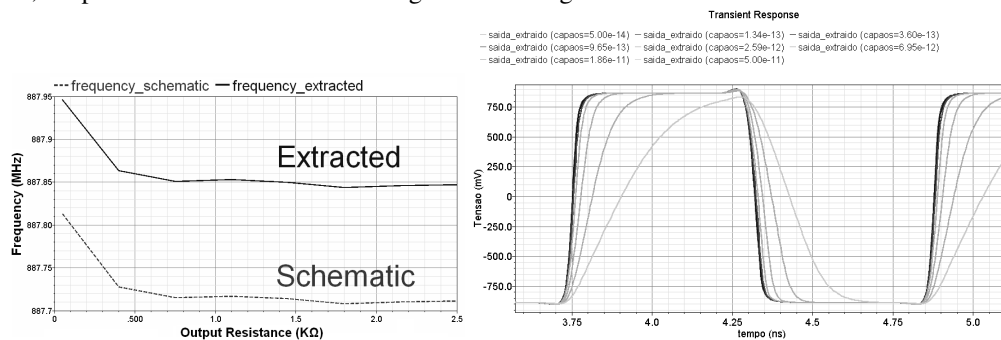


Fig. 2 – (A) - Frequency of oscillation vs. Output resistance. (B) - Simulated transient voltage in the ring oscillator.

A major cause of large variations in the frequency of the ring oscillator is the power of the circuit. With VDD reduction, the frequency decreases linearly until VDD reaches a value where the inverters, composing the ring oscillator, fail to generate a more significant reversal signal to the next inverter.

To estimate the variation in frequency, simulations were made using different values of VDD, such as described in Fig. 3(A). The graph shows an almost linear variation in the frequency (about 70MHz per 100mVolts , above $V_{DD} = 0.7\text{volts}$). The data of electrical simulations allow the evaluation of the effects of variations and errors in the oscillators' measurement.

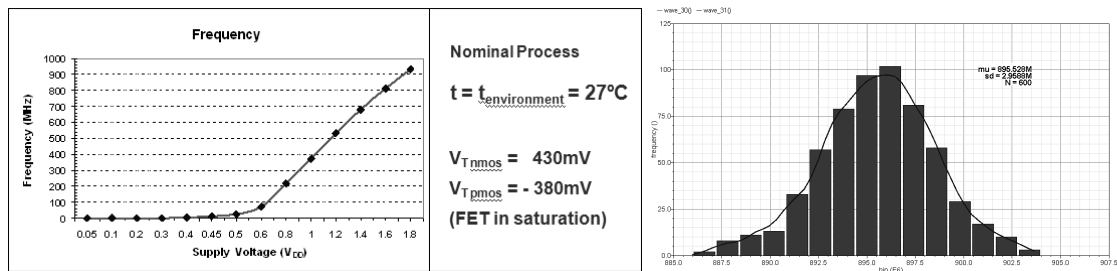


Fig. 3 – (A) - VDD vs. frequency of oscillation. (B) - Monte Carlo frequency of occurrence vs. Frequency of oscillation.

3. Monte Carlo evaluation

We used the Monte Carlo Mismatch method for disruption of electrical parameters, using the Specter simulator of Cadence™ to measure and estimate the variations in the manufacturing process. With this software we estimate how the results in the ring oscillator change after manufacture. The transistors' ideal physical data are disrupted through functions of statistical distribution and proportions previously measured by the chip manufacturer. This software allows recognition of the statistical variation in the oscillation frequency caused by procedure errors.

For this simulation we used a nominal VDD of 1.8v and we measured the frequency output of the ring oscillator in six hundred mismatch simulations (Monte Carlo iterations). In each simulation the physical values were changed automatically by the electric simulator. A trend line in the Monte Carlo analysis (Fig. 3(B)) shows that the average rate expected for the ring oscillator is at 895.5 MHz.

The simulation shows a standard deviation (1σ) corresponding to 2.96 MHz which is a very small variation in frequency, about 0.33%. This variation achieves 1% of error in the worst case of 3σ .

4. Measurements

After the simulations, extraction and electrical post-layout verification, the test-chip was manufactured. One-circuit test is in the ring oscillator in 0.18 μm CMOS technology from IBM (Fig. 1). We used Pmos transistors of 0.6 μm width and Nmos of 0.4 μm width, both with a nominal length of 0.18 μm . We collected information about the frequency of oscillation and the average delay of the spread of an inverter with different VDD. The ring oscillator and the multi-stage buffer occupy an area of only 0.0073 mm² of the chip area (6.8mm²), as illustrated in Fig. 4(A).

The measurements were performed varying the power in steps of 20mV for data collection. Measurements were made in the output buffer with an oscilloscope and a spectrum analyzer. Only the encapsulated chips provided by MOSIS (5 in total) were measured. The number is insufficient for a statistical analysis of the variation between dies on the same wafer.

5. Analysis of measurements and simulations

The measurements taken allowed to plot the curves of frequency (Fig. 4(B)) as a function of the voltage VDD. The power of the circuit varied systematically from 0.9V to 2V as the data were collected.

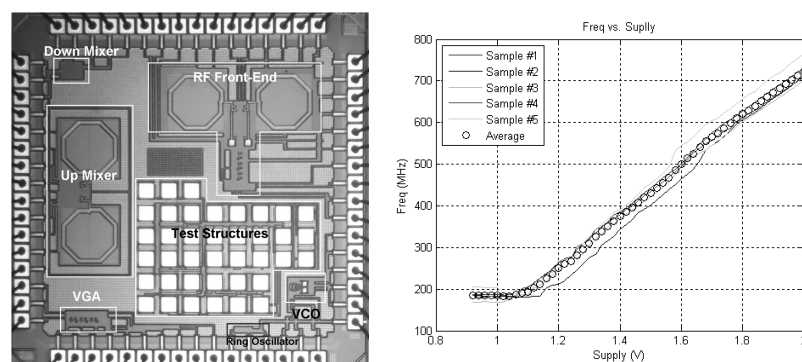


Fig. 4 – (A) - IBM 0.18 μm chip. (B) – Measured frequency of oscillation vs. VDD (V).

As we lower the power supply of the circuit, the frequency also reduces. When close to supply voltage VDD of 1.1V to the circuit, the frequency becomes insensitive to changes in VDD. When comparing the measured graphs with the simulation graphs, we see a similar frequency behavior when varying the VDD (Fig. 3(A) and Fig. 4(B)). The ring oscillator measured does not reach the frequency expected by the Monte Carlo simulation (of approximately 900 MHz), reaching only 695MHz at VDD = 1.8V.

The final measured oscillator frequency was 24% below the originally simulated. This reduction is due to IR drops in the VDD and inductance loops in the GND lines that feed the oscillator, unfortunately no way to measure this resistance in the chip was planned or build, so we can't get an exact value. Since the effects of parasitic capacitance, resistance and manufacturing errors together are close to 3%, it is not possible to predict its value only through changes in the frequency. So it isn't the only factor for the difference between the frequency of oscillation simulated (915MHz) and measured (695MHz). The "debiasing" in the power supply is the most aggravating factor in the final frequency obtained.

6. Conclusion

After comparing the results with the analysis of the ring oscillator in the chip and the results of simulations of Virtuoso Spectre Circuit Simulator™, the effect of "debiasing" is clear. It is also possible that this drop in VDD has occurred due to imperfections of the source used for measurements, as well as a malfunction on in the test board, which may have generated the same effect. Imperfections in the film of the metal pad VDD can cause the voltage drop. One way to avoid this kind of problem is use one force-sense line to supply the circuit, with this lines we could force one voltage on the chip pin and measure the voltage that directly arrive (after the wire resistances) on the circuit.

7. Acknowledgments

We acknowledge the support and collaboration of our colleagues Dalton Martini Colombo and Juan Pablo Martinez Brito, as well as the research funding provided by CNPq and FINEP for the sponsorship of research grants for the undergraduate students working in this project.

8. References

- [1] F. P. Cortes, E. Fabris and S. Bampi. "Applying the gm / ID method in the analysis and design of the Miller amplifier, the Comparator and the Gm-C band-pass filter." IFIP VLSI Soc 2003, Darmstadt, Germany, December 2003.
- [2] F. P. Cortes and S. Bampi. "Analysis, design and implementation of baseband and RF blocks suitable for a multi-band analog interface for CMOS SOCs." Ph.D. Thesis, PGMICRO - UFRGS, Porto Alegre, Brazil, 2008.
- [3] L. S. de Paula, E. Fabris and S. Bampi. "A high swing low power CMOS differential voltage-controlled ring oscillator." In: 14th IEEE International Conference on Electronics, Circuits and Systems - ICECS 2007, Marrakech, Morocco, December, 2007.
- [4] G. R. Camaratta, E. Conrad Jr, L. F. Ferreira, F. P. Cortes and S. Bampi. "Device characterization of an IBM 0.18µm analog test chip. In: 8th Microeletronic Student Forum 2008 (SForum2008), Chip in the Pampa, SForum2008 CD Proceedings, Gramado, Brazil, September 2008

Loading Effect Analysis Considering Routing Resistance

Paulo F. Butzen, André I. Reis, Renato P. Ribas

{pbutzen, andreis, rpribas}@inf.ufrgs.br

Instituto de Informática – Universidade Federal do Rio Grande do Sul

Abstract

Leakage currents represent emergent design parameters in nanometer CMOS technologies. The leakage mechanisms interact with each other at device level (through device geometry, doping profile), gate level (through internal cell node voltage) and also in the circuit level (through circuit node voltages). Due to the circuit level interaction of the subthreshold and gate leakage components, the total leakage of a logic gate strongly depends on circuit topology, i.e., the number and the nature of the other logic gates connected to its input and output. This is known as loading effect. This paper evaluates the relation between loading effect and the magnitude of gate leakage and also the influence of routing resistance in the loading effect analysis. The experimental results show that the loading effect modifies the total leakage of a logic gate up to 18%, when routing resistance is considered, and its influence in total cell leakage is verified only when gate leakage is three or less orders of magnitude smaller than ON current.

1. Introduction

Leakage currents are one of the major design concerns in deep submicron technologies due to the aggressive scaling of MOS device [1-2]. Supply voltage has been reduced to keep the power consumption under control. As a consequence, the transistor threshold voltage is also scaled down to maintain the drive current capacity and achieve performance improvement. However, it increases the subthreshold current exponentially. Moreover, short channel effects, such as drain-induced barrier lowering (DIBL), are being reinforced when the technology shrinking is experienced. Hence, oxide thickness (T_{ox}) has to follow this reduction but at expense of significant gate tunneling leakage current. Furthermore, the higher doping profile results in increasing reverse-biased junction band-to-band tunneling (BTBT) leakage mechanism, though it is expected to be relevant for technologies below 25nm [3].

Subthreshold leakage current occurs in off-devices, presenting relevant values in transistor with channel length shorter than 180nm [4]. In terms of subthreshold leakage saving techniques and estimation models, the 'stack effect' represents the principal factor to be taken into account [5]. Gate oxide leakage, in turn, is verified in both on- and off-transistor when different potentials are applied between drain/source and gate terminals [6]. Sub-100nm processes, whose T_{ox} is smaller than 16Å, tends to present subthreshold and gate oxide leakages at the same order of magnitude [7]. In this sense, high-K dielectrics are been considered as an efficient way to mitigate such effect [8].

Since a certain leakage mechanism cannot be considered as dominant, the interaction among them should not be neglected. There have been previous studies in evaluating the total leakage currents at both gate and circuit level. In [9], the leakage currents in logic gates have been accurately modeled. However, this work did not incorporate loading effect in circuit analysis. Mukhopadhyay in [10] and Rastogi in [11] considered the impact of loading effect in total leakage estimation at circuit level. Those works claims that the loading effect modifies the leakage of individual logic gates by ~ 5% – 8%. However, it is only true when the gate leakage has a significant value. This work has two important contributions. First, the loading effect is analyzed for different gate leakage magnitudes. This analysis shows when loading effect achieve the importance levels presented in [10]. The routing influence in loading effect is the second topic explored in this work. It shows that loading effect could be worsening than presented in previous works [10-11].

2. Loading Effect

In logic circuits, leakage components interact with each other through the internal circuit node (input/output cell) voltages. The voltage difference at the input/output node of a logic gate due to the gate leakage of the gates connected to its input/output is defined as loading effect. In fig. 1, the output node voltage N of gate G0 is connected to input of G1, G2 and G3 gates. The gate leakage of G1, G2 and G3 change the voltage at node N. This effect is defined as loading effect and it modifies the leakage current of all gates connected to evaluated node. Each leakage component moves in different directions. In gate G0, it reduces subthreshold, gate and BTBT leakages. In gates G1, G2, and G3 it does not affect BTBT leakage, reduces gate, and increases subthreshold leakage.

An analysis reported in the literature [10] shows that the loading effect modifies the leakage of a logic gate by 5% – 8%. However, this influence is directly related to the magnitude of the gate leakage current. Tab. 1

presents devices characteristics of three variation of 32nm Berkeley Predictive BSIM4 model [12] that are used to explore the influence of gate leakage magnitude in loading effect. These variations change the value of the oxide thickness (T_{ox}) to achieve different values of gate leakage (I_{gate}) without compromise the ON current (I_{ON}) and the subthreshold leakage (I_{sub}). There are two types of gate leakage in tab.1: the I_{gate_ON} represent the gate current when the transistor is conducting and I_{gate_OFF} represents the gate current when the transistor is not-conducting. Tab. 2 presents the DC simulation results performed in HSPICE for circuit depicted in Fig. 1. The inverters have the same size: 1um to NMOS transistors and 2um to PMOS transistors. This table shows the relation between the gate leakage current (through the three different devices models presented in tab. 1) and the voltage at node N and also the total leakage current in gates G0 and G1. Gates G2 and G3 have the same total leakage of gate G1. It is possible to conclude that the loading effect starts to modify the leakage analysis when the gate leakage current is three orders of magnitude smaller than the ON current. Tab. 2 also illustrates the different loading effect contribution in total cell leakage. The total leakage in gate G0 decrease, while it increase in gates G1, G2 and G3. The leakage increment in gate G1 when “Variation 3” technology process is used does not follow the tendency presented in other results due to the gate leakage dominance. The subthreshold current in gateG1 increase significantly, but the reduction in gate leakage (the dominant leakage) compensate that increment. The experiment when input signal “IN” is “1” follows the same characteristics and is suppressed in this paper.

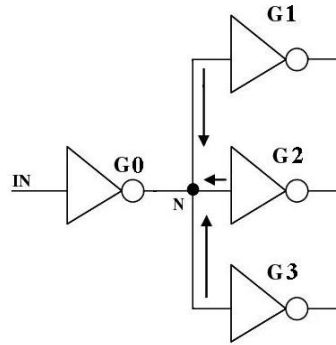


Fig. 1 – Illustration of the loading effect in node N.

Tab.1 – Device Characteristics.

Transistor Type	Parameter	Variation 1	Variation 2	Variation 3
	T_{ox} (Å)	14.00	12.00	10.00
NMOS	I_{on} (uA/um)	1367.20	1345.60	1284.50
	I_{sub} (uA/um)	0.82	0.87	0.92
	I_{gate_ON} (uA/um)	0.18	1.13	7.62
	I_{gate_OFF} (uA/um)	0.05	0.33	2.26
PMOS	I_{on} (uA/um)	412.98	376.83	331.53
	I_{sub} (uA/um)	0.64	0.72	0.83
	I_{gate_ON} (uA/um)	< 0.01	0.05	0.51
	I_{gate_OFF} (uA/um)	< 0.01	0.02	0.25

Tab.2 – Loading voltage in node N and the total leakage currents in gates G0, G1, G2, and G3 (illustrated in fig. 1 when $IN = 0$) for different devices presented in tab. 1.

Process	V(N) (V)		Diff (%)	I _{Leak} (G0) (uA)		Diff (%)	I _{Leak} (G1) (uA)		Diff (%)
	NLE *	LE **		NLE *	LE **		NLE *	LE **	
Variation 1	1.00	0.998	- 0.2	0.88	0.88	0.0	1.46	1.51	3.4
Variation 2	1.00	0.996	- 0.4	1.30	1.28	- 1.5	2.61	2.71	3.8
Variation 3	1.00	0.977	- 2.3	4.20	3.90	- 7.1	9.78	10.00	2.2

* NLE = No Loading Effect ** LE = Considering Loading Effect

3. Routing Resistance Influence

Previous analysis shows that the loading effect can modify the leakage current in circuit gates when the gate leakage current achieve values about three orders of magnitude smaller than ON current. There is another point that has not been explored in works reported in the literature. As the evaluation point is the leakage interaction

between gates, the resistance of wires used to connect those gates have to be taking account for a more precisely analysis. Fig. 2 illustrates the previous statement adding a resistor between the gates depicted in Fig. 1.

Horowitz in [13] presents an excellent analysis of routing wires characteristics in nanometer technologies, where is concluded that wire resistance (per unit length) grows under scaling. From data presented in that work, the resistance from equivalent wire segments increases 10% – 15% for each technology node. In new technology generations, the regularity is a key to achieve considerable yield. In this case, the routing resistance tends to become even bigger. Experimental data have showed a routing resistance varies from units of ohms to kilohms for an 8 bit processor designed using FreePDK design kit [14]. Since the resistance in 32nm tends to become worst, the value used in proposed experiment is 2K Ω . In the same way, gate leakage is predicted to increase at a rate of more than 500X per technology generation [6]. Both characteristics show that the routing resistance could not be neglected in loading effect analysis. Tab. 3 presents the DC simulation results performed in HSPICE for circuit depicted in Fig. 2. The resistor value is 2K Ω and the inverters have the same size: 1 μ m to NMOS transistors and 2 μ m to PMOS transistors. This table shows the relation between the gate leakage current (through the three different devices models presented in tab. 1) and the voltage at node N0 and N1. The total leakage current in gates G0 and G1 are also presented. The gates G2 and G3 present the same leakage value of gate G1.

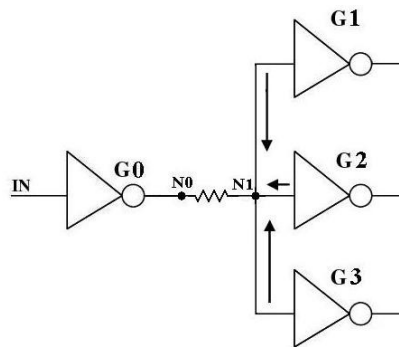


Fig. 2 – Illustration of the loading effect in node N considering the routing resistance.

Tab.3 – Considering routing resistance in loading voltage in node N, N0 and N1 and the total leakage currents in gates G0 and G1 (illustrated in fig. 2 when IN=0) for different devices presented in tab. 1.

Process	V(N0) (V)	V(N1) (V)	I _{Leak} (G0) (μ A)	I _{Leak} (G1) (μ A)
Variation 1	0.998	0.997	0.88	1.54
Variation 2	0.996	0.989	1.29	2.90
Variation 3	0.980	0.942	3.94	11.61

When compared to tab. 2, it is evident the influence of routing resistance in the leakage current through loading effect voltage shift. As mentioned before, the loading effect increases some leakage components and decreases others. However, when routing resistance is considered, the total leakage in circuit gates become worst. The decrement is not as significant and the leakage increment is more severe. The loading effect when routing resistance is considered modifies the total leakage of a logic gate up to 18%.

The total leakage current in gate G0 (a) and G1 (b) when routing resistance is not considered is compared to the total leakage when routing resistance is considered (LE + R) for different values of gate leakage (through the three different devices models presented in tab. 1).

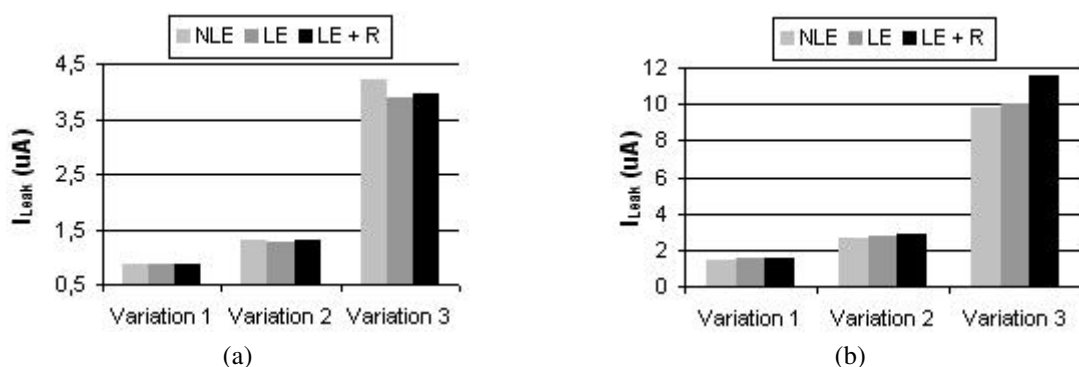


Fig. 3 – Total leakage in gate G0 (a) and G1 (b) for different values of gate leakage.

4. Conclusions

In this paper the loading effect has been reviewed, the direct relation to of gate leakage has been reinforced and the influence of routing resistance, not explored in previous works reported in the literature, are analyzed. The first analysis shows that the loading effect starts to modify the leakage analysis only when the gate leakage current is three or less orders of magnitude smaller than the ON current. The second analysis explores the influence of the routing resistance in the loading effect. The experiments show that the loading effect modifies the total leakage of a logic gate up to 18% when routing resistance is considered. The importance in consider the resistance, mainly in new technology generations, is increasing due to the gate leakage tends to become more severe and the routing resistance increase more than usual due to regularity required by manufacturability rules.

5. Reference

- [1] "2007 International Roadmap for Semiconductor (ITRS)", <http://public.itrs.net>.
- [2] K. Roy *et al.*, "Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-submicron CMOS Circuits", *Proc. IEEE*, vol. 91, no. 2, pp. 305-327, Feb. 2003.
- [3] A. Agarwal *et al.*, "Leakage Power Analysis and Reduction: Models, Estimation and Tools", *IEE Proceedings – Computers and Digital Techniques*, vol. 152, no. 3, pp. 353-368, May 2005.
- [4] S. Narendra *et al.*, "Full-Chip Subthreshold Leakage Power Prediction and Reduction Techniques for Sub-0.18 μ m CMOS", *IEEE Journal of Solid-State Circuits*, vol. 39, no. 3, pp. 501-510, Mar. 04
- [5] Z. Cheng *et al.*, "Estimation of Standby Leakage Power in CMOS Circuits Considering Accurate Modeling of Transistor Stacks", *Proc. Int. Symposium Low Power Electronics and Design, (ISLPED 1998)*, 1998, pp. 239-244.
- [6] R. M. Rao *et al.*, "Efficient Techniques for Gate Leakage Estimation", *Proc. Int. Symposium Low Power Electronics and Design, (ISLPED 2003)*, 2003, pp. 100-103.
- [7] A. Ono *et al.*, "A 100nm Node CMOS Technology for Practical SOC Application Requirement", *Tech. Digest of IEDM*, 2001, pp. 511-514
- [8] E. Gusev *et al.*, "Ultrathin High-k Gate Stacks for Advanced CMOS Devices," *IEDM Tech. Digest*, 2001, pp. 451– 454.
- [9] P. F. Butzen *et al.*, "Simple and Accurate Method for Fast Static Current Estimation in CMOS Complex Gates with Interaction of Leakage Mechanisms", *Proceedings Great Lakes Symposium on VLSI (GLSVLSI 2008)*, 2008, pp.407-410.
- [10] S. Mukhopadhyay, S. Bhunia, K. Roy, "Modeling and Analysis of Loading Effect on Leakage of Nanoscaled Bulk-CMOS Logic Circuits", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no.8, pp. 1486-1495, Aug. 2006.
- [11] A. Rastogi, W. Chen, S. Kundu, "On Estimating Impact of Loading Effect on Leakage Current in Sub-65nm Scaled CMOS Circuit Based on Newton-Raphson Method", *Proc. Design Automation Conf.(DAC 2007)*, 2007, pp. 712-715.
- [12] W. Zhao, Y. Cao, "New generation of Predictive Technology Model for sub-45nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816-2823, Nov. 2006.
- [13] M.A. Horowitz, R. Ho, K. Mai, "The Future of Wires", *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.
- [14] "Nangate FreePDK45 Generic Open Cell Library", www.si2.org/openeda.si2.org/projects/nangatelib.



SOC/Embedded Design and Network on Chip

A Floating Point Unit Architecture for Low Power Embedded Systems Applications

Raphael Neves, Jeferson Marques, Sidinei Ghissoni, Alessandro Girardi

{raphaimdcn2, jefersonjpm}@yahoo.com.br,
{sidinei.ghissoni, alessandro.girardi}@unipampa.edu.br

Federal University of Pampa – UNIPAMPA
Campus Alegrete
Av. Tiaraju, 810 – Alegrete – RS – Brasil

Abstract

This paper presents a proposed architecture and organization of a floating point unit (FPU), which follows the IEEE 754 standard for representation of floating point numbers. The development of this FPU was made with emphasis on embedded systems applications with low power consumption and fast processing of arithmetic calculations, performing basic operations such as addition, subtraction, multiplication and division. We present the implementation of the floating-point detailing the data flow and the control unit.

1. Introduction

Support for floating-point and integer operations in most computer architectures is usually performed by distinct hardware components [1]. The component responsible by floating point operations is referred to as the Floating Point Unit (FPU). This FPU is aggregated or even embedded in the processing unit and is used to accelerate the implementation of different arithmetic calculations using the advantages of binary arithmetic in floating point representation, providing a considerable increase in the performance of computer systems. Floating-point math operations were first implemented through emulation via software (simulation of floating-point values using integers) or by using a specific co-processor, which should be acquired in separate (offboard) from the main processor. Because every computer manufacturer had its own way of represent floating point numbers, the IEEE (Institute of Electrical and Electronics Engineers) issued a standard for representation of numbers in floating point. Thus, in 1985, IEEE created a standard known as IEEE 754 [2]. Intel adopted this standard on its platform IA-32 (Intel Architecture 32-bit), which had a great impact for its popularization.

This paper presents a proposal of architecture and organization of a floating point unit which works with 32 bits floating point numbers, or the single precision IEEE standard 754. The FPU developed have application on embedded devices [7] with low power dissipation and high performance, including 4 arithmetic operations: addition, subtraction, multiplication and division. The rounding mode can be specified according to the IEEE standard 754 [2].

2. Floating Point Standard

The IEEE developed a standard defining the arithmetic and representation of numbers in floating point format. Although there are other representations, this is the most commonly used in floating point numbers. The IEEE standard is known as IEEE 754. This standard contains rules to be followed by computer and software manufacturers for the treatment of binary arithmetic for numbers in floating point on the storage, methods of rounding, the occurrence of overflow and underflow, and basic arithmetic operations. The standard IEEE 754, single precision, is composed by 32 bits, being 1 bit to represent the sign, 8 bits for exponent and 23 bits for the fraction, as shown in fig. 1.

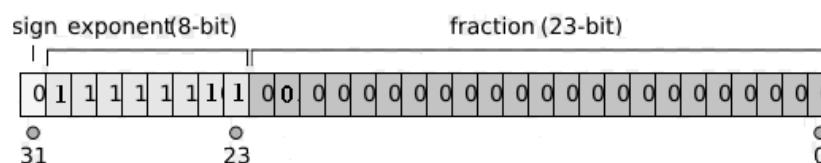


Fig. 1 – IEEE Standard for floating point numbers in single precision.

A number in single precision floating point is represented by the following expression:

$$(-1)^s 2^e \cdot 1.f \quad (\text{eq. 1})$$

where $f = (b^{-1}_{23} + b^{-2}_{22} + \dots + b^{-n}_i + \dots + b^{-23}_0)$ is the fraction, being $b^{-n}_i = 0$ or 1, s is the signal bit (0 for positive; 1 for negative), e is the biased exponent ($e = E + 127$ (bias), E is the unbiased exponent). We call mantissa M the binary number composed by an implicit 1 and the fraction:

$$M = 1.f \quad (\text{eq. 2})$$

The IEEE standard 754 provides special representation for exception cases, such as infinity, not a number, zero, etc. There is also the IEEE standard 754 representation of floating point numbers in double precision which is composed by 64 bits [2]. The focus of this work is the representation in 32-bit single precision.

3. Floating Point Operations

Floating-point arithmetic operations are widely applied in computer calculations. In this work, the proposed FPU supports addition, subtraction, multiplication and division. Sum and subtraction are the most complex floating point operations, because there is a need to align the exponents [3]. The basic algorithm can be divided into four parts: verify exceptions, check the exponents, align the mantissa and add the mantissa considering the signal. In fig. 2 is shown the flowchart of the sum and subtraction.

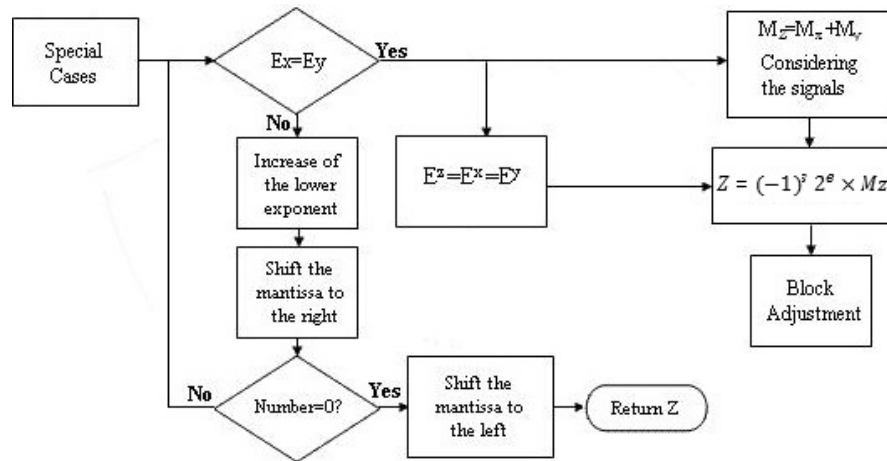


Fig. 2 – Flowchart of the sum and subtraction.

The data flow of multiplication is simpler. After the signal is defined, we add the exponents and the mantissa is multiplied. These operations occur in parallel. In multiplication, it is necessary to subtract the bias (127), since during the sum of the exponents it is added twice. This subtraction is necessary to make the exponent as the standard IEEE 754. After going through these steps, the result is sent for the adjustment block. The operation of division follows the same data flow, with the difference that the mantissas are divided, the exponents are subtracted and add bias(127). We can observe that after completing these steps, the result is sent for the adjustment stage. This stage includes detection of overflow, underflow, besides performing rounding and normalization.

4. Proposed Architecture

This section presents the data path and control unit of the proposed FPU. The fundamental principle of the design is to develop an architecture capable to support the flow described in section 3. So, we used the following strategy: the functional units (adder, shifter, registers) are shared between different operations [6]. For example, to perform the multiplication we need to add the exponents, so the same hardware will be reused for the sum of mantissas in the addition operation. The advantage of using this technique is the economy of functional units, which also means saving silicon area. The disadvantage is the increase in the number of cycles required to perform the operation. So with the implementation of this methodology is expected to result in future an insignificant degradation in speed and decrease power considerably.

4.1. Data Path

The architecture of the FPU consists of five main blocks: exception verifier, integer adder/subtractor, integer multiplier, integer divider and shifter. To connect these elements and solve data conflicts we used

multiplexers (MUX). Registers (Reg) are used to store data. Fig. 3 shows the architecture of the FPU data path. The control unit is described in the next section.

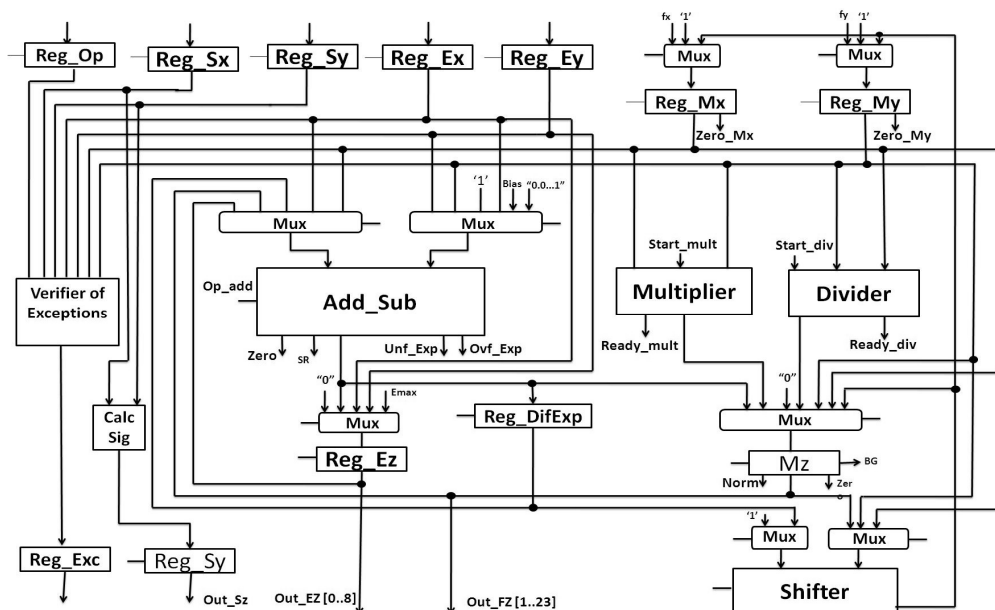


Fig. 3 – The operative part of the FPU.

The basic operations of the FPU start recording the operands in the input registers. In the next cycle these values are read by the exception verifier, so if there is some kind of special case (Not a Number, infinity, zero, etc) it will be reported. Thus, a signal is sent to the control that returns a flag of exception and the transaction is completed. On the other side, if no special case is identified it will launch the desired operation. As mentioned before, the hardware is shared between operations. One of the most important block is the adder/subtractor (Add_Sub). It is responsible for adding both the mantissa and the exponents, comparison between two values and, moreover, it is also used for normalization and rounding.

At architecture level and considering that all the blocks perform the operation in one clock cycle, we can say that the multiplication and division are faster than addition. Tab. 1 shows the number of clock cycles required to perform these operations. However, we know that integer multiplication and division are so far complex and can not be implemented in practice in a single clock cycle [4]. Compared to works of literature, the FPU has a regular income [5] [6].

Tab.1 – Number of clock cycles necessary to perform operations

Operation	Clock cycles for execution
Add	15
Multiplication	12
Divide	12

4.2. Control Unit

In fig. 4 we showed the operative part of the FPU. However, there must be control unit dealing with the control signals of multiplexers, registers and other basic blocks. There are various ways to implement the control unit in a processor. In this design we choose to implement a microprogrammed control. With this type of control we have microinstructions that define a set of signals to control the datapath. Moreover, a microprogram is executed sequentially. In some cases we need to branch execution based on signals from the data path. For this, we use dispatch tables when necessary. Tab. 2 shows each field of the microinstructions with the size of the word and its function.

Tab. 2-Fields of the microinstructions for the microprogrammed control unit.

Field Name	Number of Bits	Function
Mux	19	Control of the multiplexers.
Seq	4	Indicates the next microinstruction to be executed
Op_Add	1	Controls operation of the block Add_sub.
Start_Mult	1	Starts the operation of multiplication.
Start_Div	1	Starts the operation of Division.
Reg	12	Controls writing in registers.
E_D	1	Indicates shift direction (right or left)

Therefore, the operation of the control unit is based on input signals that, according to the program counter, enable the line of a ROM memory containing the microinstructions. The memory output contains control signals for the data path. In fig. 4 is shown the internal organization of the control unit. An adder is necessary to increment the program counter and a dispatch table implements the branches.

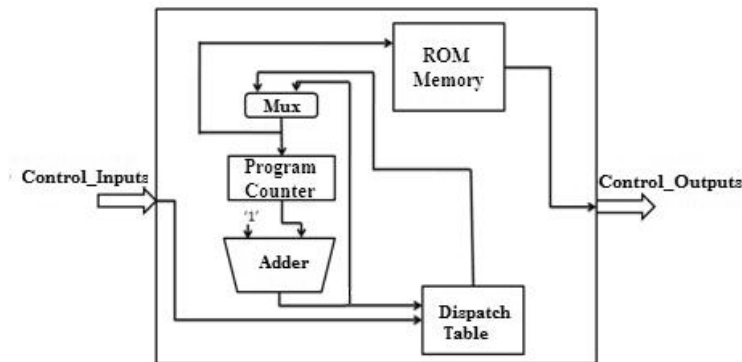


Fig. 4 – Internal blocks of the control unit.

5. Conclusion

In this work we presented a new alternative of FPU architecture for embedded systems that support operations in floating point standard IEEE 754. In this design we used techniques for reducing dissipated power avoiding performance degradation. Thus, this architecture was developed with the objective of achieving low power consumption for low power applications. The hardware is shared between operations and there is only one functional block for each basic operation, such as integer addition, shift, etc.

As future work, we intend to implement the proposed architecture in an RTL description and prototype it in FPGA. Also, some hardware acceleration techniques can be implemented in order to increase performance without demanding increasing in power.

6. Acknowledgments

This work is part of Brazil-IP initiative for the development of emerging Brazilian microelectronics research groups. The grant provided by CNPq is gratefully acknowledged.

7. References

- [1] R.V.K Pillai, D. Al-Khalili and A.J. Al-Khalili, A Low Power Approach to Floating Adder Design, Proceedings of the 1997 International Conference on Computer Design (ICCD '97); 1997.
- [2] IEEE Computer Society (1985), *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Std 754, 1985.
- [3] William Stallings, *Arquitetura e Organização de Computadores*, 5th Ed., Pearson Education, 2005.
- [4] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*, 3th Ed., 2003.
- [5] Asger Munk Nielsen, David W. Matula, C.N. Lyu, Guy Even, An IEEE Complicant Floating-Point Adder that Conforms with the Pipelined Packet-Forwarding Paradigm, *IEEE TRANSACTIONS ON COMPUTERS*, VOL. 49, NO. 1, JANUARY 2000.
- [6] Taek-Jun Kwon, Joong-Seok Moon, Jeff Sondeen and Jeff Draper, A 0.18 μ m Implementation of a Floating Point Unit for a Processing-in-Memory System, Proceedings of the International Symposium on Circuits and Systems - ISCAS 2004.
- [7] Jean-Pierre Deschamps, Gery Jean Antoine Bioul, Gustavo D. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*. Ed.

MIPS VLIW Architecture and Organization

Fábio Luís Livi Ramos, Osvaldo Martinello Jr, Luigi Carro

{flramos,omjunior,carro}@inf.ufrgs.br

PPGC – Instituto de Informática
Universidade Federal do Rio Grande do Sul

Abstract

This work proposes an architecture and organization for a MIPS based processor using a VLIW (Very Long Instruction Word) approach. A program that assembles the instructions according to the hardware features of the proposed VLIW organization was built, based in an existent program that detects the parallelism in compilation time. The organization was described in VHDL based in a MIPS description called miniMIPS.

1. Introduction

Enterprises and universities look for even more efficiency and performance from the developed processors. Many different strategies were and are used with this purpose, as, for example, multi-cycle machines, pipeline approach, among others.

One of these strategies consists in multiplying the features of the processor: when before only one operation was executed in some of the stages of a pipeline, now two or more operations can be executed in parallel. According to this strategy, VLIW machines [1] that are able to process multiple instructions in parallel were proposed, increasing the processor performance.

This work proposes an implementation of VLIW architecture and organization based in the processor MIPS. The second section will show more details of the VLIW scheme, followed by a discussion on MIPS architecture on section 3. Section 4 explains the developed compiler and section 5 the hardware proposed. Finally, section 6 shows the conclusions of this work.

2. VLIW Architecture

VLIW architectures are able to execute more than one operation on each processing cycle. It is able to do so because they have multiple parallel processing units.

Exploitation of ILP (Instruction Level Parallelism) is necessary to profit from these extra units. In other words, it is necessary to discover the dependencies between each instruction of the program to avoid execution errors.

Differently than the super-scalar architectures – where the dependencies are discovered in execution time by the processor hardware (and making the hardware even more complex) – in VLIW machines the dependencies are discovered in compilation time. In this case, the compiler is in charge of discovering the parallelism inside a program and organizing the new code in machine language.

On this architecture, the compiler is specific for the VLIW machine target, and needs to know which features the processor offers. If, for example, the processor has four ALUs, the compiler needs to know this in advance because when it assembles the VLIW word – after discovering the dependencies between every instruction in the code – it will try to allocate in a same word up to four of ALU instructions to use to the maximum the features offered by the processor.

Obviously, as this new instruction word assembled unites more than one original instruction (from now on called atom of the VLIW instruction), the VLIW instruction is longer than that from a simple-scalar equivalent, and that is the reason of the architecture name.

The compiler, when assembling the word, allocates each atom according with the resource that will be used. In this way, each position inside the VLIW is bound to a specific resource. Exemplifying, one possibility is a VLIW processor with word of three atoms, where the first is bound to instructions that use the ALU, the second position for Branch instruction and the third for Load/Store.

3. MIPS Architecture

MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC processor architecture developed by MIPS Technology [2].

The MIPS architecture is based on the pipeline technique with canonic instructions to generate a CPI (Cycles per Instruction) almost equal to 1. This means that after the pipeline is full, at nearly every clock cycle, one instruction is finished. This limit cannot be reached because of the unpredictability of branches, data dependencies and cache misses.

The original MIPS pipeline is composed by 5 stages: Instruction Fetch, Instruction Decode, Execution, Memory Access and Write Back.

4. VLIW Compiler

The compiler proposed in this work is based on a simulator developed in UFRGS that can, using a MIPS code, simulate and construct a data dependency graph between the instructions, and was implemented using the ArchC simulator [3] developed in UNICAMP.

ArchC is an open source architecture developing language (ADL) based in the SystemC language. Using the ArchC, the DIM platform was developed, that, among other features, can provide the creation of a data dependency tree of a code simulated in ArchC.

The proposed compiler uses this tree to assemble a new code compatible with the VLIW machine described, whose definition and justification will be found in the sequence of this text.

4.1. Defining the Functional Units

For the definition of the functional units of the VLIW machine to be built, dependency trees created for a number of benchmarks obtained from the ArchC website [3] were analyzed. The analyses were focused in four of these benchmarks: Data Cryptography of standard AES, herein called RijndaelE; Decryption AES, here called RijndaelD; Hash functions calculus (Sha); and a text search algorithm (String).

Based on the data generated by DIM, estimations of the execution time for these algorithms could be made for some VLIW machine configurations, even without using the generation of specific code for some processor.

Some of the preliminary data that helped us to define an organization that could be efficient enough, and without waste, to obtain a significant increase in the performance compared to the original MIPS processor, are shown in tab. 1. These data were obtained from traces of a simulation of these algorithms.

Tab. 1 – Performance evaluation.

	MIPS	Branch + 1	Branch + 2	Branch + 3	Branch + 4
RijndaelE	30187866	95,37%	49,41%	34,47%	27,11%
RijndaelD	27989687	95,76%	49,12%	34,07%	26,72%
Sha	13009686	93,53%	47,55%	33,96%	27,96%
String	236249	79,98%	43,90%	34,25%	30,25%

In tab. 1 are the total number of executed instructions in each benchmark using an unaltered MIPS; The relative number of executed instructions for a MIPS VLIW with one unit exclusively for branch and one more capable of executing every other instructions; and similar with more multivalent units.

Analyzing these numbers we judged as the best relation cost-benefit the utilization of three functional units besides the branch unit to the composition of the organization of the VLIW version of MIPS. However, this would require the utilization of a data memory with triple access for read/write. This fact would certainly increase the cost of the device and the time of memory access.

With these facts an analysis was done considering a possible limitation of the functional units: not all three multivalent functional units would process the Load/Store operations.

To aid with the decision, a new table (tab. 2) was elaborate, that shows us how benefic would be the utilization of multiple function units that realize the Load/Store operations.

Tab. 2 – Performance evaluation.

	MIPS	1 Load/Store	2 Load/Store	3 Load/Store
RijndaelE	30187866	40,26%	34,98%	34,47%
RijndaelD	27989687	41,21%	35,16%	34,07%
Sha	13009686	39,64%	35,95%	33,96%
String	236249	48,47%	43,36%	34,25%

Considering the benefits of having more than one functional unit capable of executing the Load/Store operations, the amount of functional units of the VLIW MIPS was defined as follows: one functional unit capable of processing only branches; one functional unit for all atoms, except branches; and two units for every instruction, except branches and Load/Store.

4.2. Code Generation

Once the machine code for the original MIPS is generated – using a MIPS R3000 compiler –, it is executed in the ArchC modified with the DIM, that will simulate the code execution, separate the codes in Basic Blocks

[1] (sets of instructions that contains only one entry point, and ends exactly in a jump operation – this means that once the execution of a Basic Block is started, all the instructions inside the block will be executed), and build a data dependency tree for each of these Basic Blocks.

The process consists in traveling inside the graph and composing one VLIW instruction with only instructions that have no dependencies at all. After the creation of an instruction, operations used are removed from the tree and the process is repeated until there are no more instructions in the graph.

When there are no instructions enough to be executed in parallel as the hardware is able to, Nops are inserted (nop - no operation) in the code in order to maintain the same size of the VLIW instruction. This makes the VLIW binary codes bigger than scalars machine codes, demanding higher capability of code memory.

Another additional care is that as the structure and position of the binary code are altered, the jump and branch addresses must be recalculated to ensure the code consistency.

As following an example of VLIW code generation. Fig. 1 shows a MIPS code to be converted to a VLIW code (The indexes in parenthesis are just for a better comprehension of the process).

```
addiu(1) 29 29 -1
sw(1)    -1 17 29
sw(2)    -1 16 29
sw(3)    -1 18 29
lbu(1)    3  6 -1
addu(1)   6  0 16
addu(2)   4  0  6
andi(1)   4  3 -1
addu(3)   5  0 17
addiu(2) 24 16 -1
beq(1)    -1  4  0
addu(4)   0  0  3
```

Fig. 1 – Example of a MIPS code.

This code is passed for the DIM that creates a data dependency graph as shown in fig. 2:

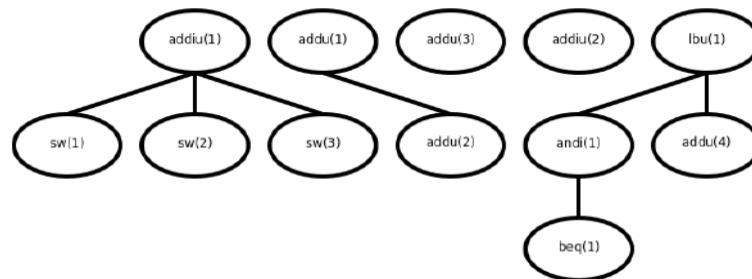


Fig. 2 – Dependency Graph.

As the next step, the compiler here proposed parallelizes the instructions resulting in the code of fig. 3:

```
addiu(1)  addu(1)  lbu(1)  nop
addu(3)   addiu(2) sw(1)   nop
addu(2)   andi(1)  sw(2)   nop
addu(4)   nop     sw(3)   beq(1)
```

Fig. 3 – Example of a MIPS VLIW code.

5. MIPS VLIW

After the research presented in the previous chapters and the generation of the specific compiler to generate the VLIW code to a MIPS processor, it was necessary to implement the machine itself.

For this purpose it was used one VHDL implementation of MIPS called miniMIPS [4]. The miniMIPS is an implementation of a processor based in the MIPS architecture available in OpenCores.

In possession of the miniMIPS, its VLIW version was created. As the new word would be composed by 4 atoms and being the miniMIPS a 32 bits architecture, the size of the memory word of the instruction will have now the size of 128 bits, extending the memory length. The two first atoms are for arithmetic and logic instructions; the third atom can be for arithmetic/logic instructions or memory access; and the fourth is only for branch/jump operations (fig. 4):

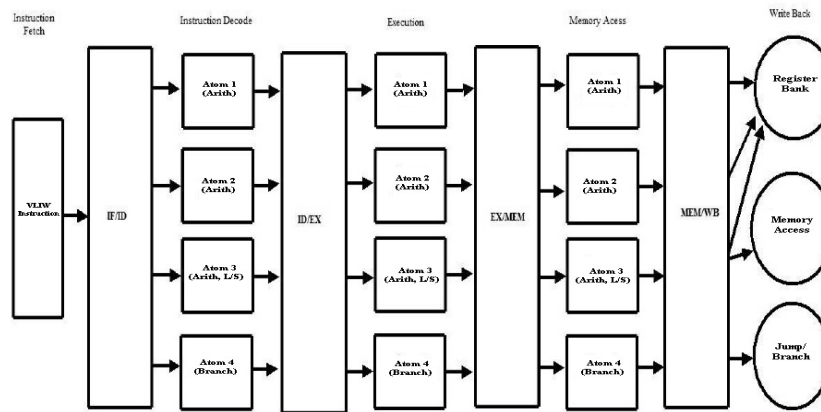


Fig. 4 – MIPS VLIW Organization.

Considering the four instruction flows in parallel, first it was extended the output of data in the stage of Instruction Fetch, now breaking the VLIW word in four instructions that will follow distinct flows inside the processor organization.

Then, it was necessary to multiply the following stages by four to process the atoms. First, each stage of decoding takes one of the atoms and decodes it according to the instructions available in the miniMIPS. After that, each decode instruction pass through its respective stage of execution generating the data necessary according to the type of instruction. The data necessary are searched in the Register Bank to the calculus of the operations, addresses and other values.

In the following stage, the third atom (the one for access memory instructions) takes its output signals to the memory access unit of the device. This unit is the one that allows the correct access to the RAM that is used by the miniMIPS. Being this the only instruction that does L/S, it is only necessary to pass its signals to the RAM control.

The two atoms of arithmetic and logic operations pass its signals to the control of the Register Bank for the writes that can be necessary (this is also valid for the third atom if they carry a logic/arithmetic operation or in case of a LOAD). The Register Bank now needs more inputs and outputs because of the increase of numbers of parallel accesses that may be submitted. Hence, it needs four times more outputs and three times more inputs. Finally, the branch/jump instructions pass its data to a unit that calculates the new PC value in the case that the jump was taken.

A brief comparison of the resources necessary for this new organization can be seen in tab. 3. These data come from a comparison between the syntheses of circuits using ISE for 2vp30ff896-7 FPGA.

Tab. 3 – Relation between MIPS and MIPS VLIW.

	MIPS	MIPS VLIW	Relation
#Slices	2610	9929	380,42 %
#Slices FF	2412	2767	114,72%
#LUTs	4559	18707	410,33%
Freq. (MHz)	76.407	72.148	94,43%

6. Conclusions

This work proposed a full implementation of a VLIW version on the classic MIPS. A VHDL description of the VLIW processor and a working compiler were developed.

The gains of performance were evaluated by simulation, for some benchmarks, varying the number and the function of units.

7. References

- [1] Patterson D. A. and Henessy J. L., Computer Organization and Design, 2005, Elsevier.
- [2] MIPS Technology. Mar. 2009; <http://www.mips.com/>.
- [3] ArchC site. Mar. 2009; <http://archc.sourceforge.net/>
- [4] MiniMIPS in OpenCores. Jul. 2008; <http://www.opencores.org/projects.cgi/web/minimips/over>.

SpecISA: Data and Control Flow Optimized Processor

Daniel Guimarães Jr., Tomás Garcia Moreira, Ricardo Reis, Carlos Eduardo Pereira
{dsgjunior, tgmoreira, reis}@inf.ufrgs.br, cpereira@ece.ufrgs.br

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brasil

Abstract

This article presents a complete flow to build one ASIP (Application-Specific Instruction-Set Processor). It details architecture and organization design. It proposes an optimized architecture for use with three specific programs and general use. It was developed to optimize three types of algorithms: arithmetical, search and sort algorithms. Ahead of such objective the SpecISA architecture has been proposed like a processor specification with an ISA (Instruction Set Architecture) capable to execute three specific kinds of programs improving them in performance. Parts of the implementation and some results in terms of allocated area are presented and discussed for the main blocks that compose the project.

1. Introduction

SpecISA is characterized by its capability to execute three different kinds of programs, for this a processor with a cohesive organization and a heterogeneous architecture it is necessary. The first type would be a program for the DCT-2D calculation (arithmetical). The second would be programs for searching in a sorted list, as for example, a words dictionary. The third kind would be programs to sort vectors. The main objective would be to develop an ISA for each of these programs for improving them in performance. Using this idea this work has been proposed to develop a single processor architecture capable to execute the three programs and for giving them significant velocity gains.

The design of this architecture started from manual analysis of the programs for which it would be utilized. The analysis searched for similar operations groups or some key point that could be used as focus to optimize its executions. From this analysis it was possible to perceive that one of the programs repeated arithmetical operations while the others repeated comparisons and displacements of values. Focused in giving performance to the first program the idea for creating SIMD instructions (Single Instruction Multiple Data) appeared. An instruction of this type realizes operations in parallel and reduces the programs execution time. As the SIMD model was adequate, its use was analyzed for the other programs. Such analysis showed that they also could be executed with SIMD instructions and that they could get an improvement in performance, furthermore the SORT instruction there is used to sort the data stored on the SIMD registers, 8 cycles it is the worst case. The PRT and CMP instructions operate over the SIMD register too, this approach it makes possible an improvement of performance because perform eight memory data in just one instruction with a few cycles. Then it was decided that the architecture would be developed with general purpose instructions and SIMD instructions. Thus, instructions have been created to use advantage of the maximum existing parallelism in the target programs routines.

With the definite ISA the next step was the system organization development. This step consisted of the VHDL implementation of the developed architecture. The VHDL code developed had an operative part and a logical part. The operative part was developed like a set of functional blocks and the logical part is a state machine used like control part. The implementation had as objective to validate the project and the proposed solution. The target programs, those which they will obtain the better performance using this architecture, are the DCT-2D calculation, the binary search through an ordered list and a new sort algorithm inspired in the bubble sort. The program that contains the DCT-2D calculation routine was implemented by a method that uses two DCT-1D to get the solution [1,2]. From this approach the program can resolve a DCT-2D without complex calculations, using only operations of addition, subtraction, multiplication and division for two. The DCT-1D routine realizes a lot of arithmetical operations in sequence, needing at least one instruction for each one of these operations. This fact made possible that SIMD technique was applied to speed up such calculations. An instruction that defines a set of operations over multiple inputs was developed. By this way reducing the program instructions total number and getting profits in performance.

The binary search works with a routine that it involves many comparisons. Usually, a comparison instruction would execute for each cycle. In this case, this fact was explored for creation of an instruction to realize multiple comparisons at the same time. The program improves its performance in this way.

The sort algorithm used was based on bubble sort but it has modifications for working with a bigger data set at the same time. A SIMD instruction was developed to sort an entire data block. It improves the program in terms of allocated memory and general efficiency.

The instruction set basic has 7 instruction formats. ADD, SUB and MOV instructions share the same instruction format this group. They execute its respective arithmetical operations using data contained in the registers addressed in the respective fields of the proper instruction. After the calculation, the result always is stored in the first instruction register. LDA and STA instructions load and store data in the memory. They contain beyond its proper opcode others two fields: work register (R1), where the data brought of the memory will be stored or from where the data will be taken to memory; and data memory address (END), that indicates in which data memory position will be referenced. SHIFT instruction only has a reference field to a register that it contains the data that will suffer the displacement from 1 bit, indicating a division for two. JMP instruction has an 8 bits address representing the address of instructions memory to where the PC will be directed unconditionally modifying the program flow. CNJE instruction is the only one that it has busy the 4 possible fields. Because beyond opcode it references 2 registers which contains the data that will be compared and another field indicating an address of instructions memory to where the program flow will jump case the comparison results in not an equality. NOP and HLT instructions only have the opcode field, not making anything and finishing the program flow. MIX instruction has a format similar to LDA and STA instructions. However it has as last field an instructions memory address instead of a data memory address. Beyond opcode this instruction has a register which contains the data that will modify the value contained in its ENDI field during its execution.

3. Organization

To validate this architecture project and to implement the SpecISA processor it was necessary to develop a new organization. The developed organization is based on fixed point arithmetic and composed for instructions and data memories, registers, multiplexers, functional units, bus extender, one hot to binary converter, a specific block (MIX) which realizes the addresses update and a control unit which generates signals for whole system coordination.

The Fig.2-a shows the responsible block for the instructions search cycle. It contains the instructions memory and the registers that assist in memory addressing logic and in the capture of the memory exit data. The other block, Fig.1-b, contains the data memory, registers needed to its correct addressing and data acquisition and the necessary multiplexers for the system logic. RDM 16 bits registers are necessary to store the data which will be written or read from the data memory. Beyond these, the control logic demands multiplexers and a block to extend the REM exit of bus from 13 to 16 bits. The REM register has its exit extended to put it in the addresses multiplexer entrances.

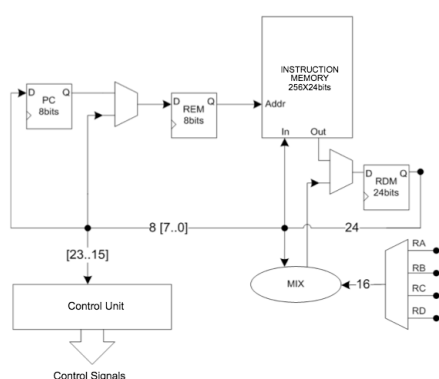
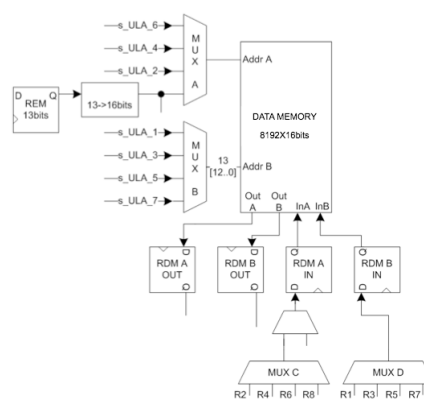


Fig. 2 – a) Instructions memory block.



b) Data memory and its elements block.

The Fig.3(a) shows the block where general purpose registers are implemented. The block that corresponds to the especial functions of this architecture is shown in the Fig.3(b). It contains 8 sets of registers (16 bits) with multiplexed entrances. This registers block is called RSIMD because its use is dedicated for operations that involve just SIMD instructions.

Depending on the operation these registers can receive data come from one of the two memory ports or data come from the any one of 8 ULA exits. The last block, Fig. 3(c), is composed by 8 half-blocks. The unit called here as half-block have three elements, two multiplexers and a functional unit. One functional unit and selection elements to its entrances compose each one of these half-blocks. This block is used by specific instructions however the two first functional units are used beyond this purpose. They are also used with the purpose general registers what it gives to the system a better utilization of its functional units.

High Performance FIR Filter Using ARM Core

Debora Matos, Leandro Zanetti, Sergio Bampi, Altamiro Susin

{debora.matos, leandrozanetti.rosa, bampi, susin}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul

Abstract

FIR filtering is one of the most commons DSP algorithms. In this paper we provide an alternative of use a general purpose processor to obtain high performance in FIR filter applications. An ARM processor VHDL description was used in this work. With the objective to obtain acceleration in the FIR filters processing, we proposed three new instructions based in ARM processor, called BUF, TAP and INC. Besides we add one instruction with the function to obtain the average between two values. This instruction showed advantageous in the proposed case study. With the new instructions it is possible to have gains of performance in FIR filter applications using a RISC processor. Moreover, using the ARM processor to accelerate FIR filters allows that the processor can be used for any other application, which would not be possible if we had used a DSP processor. Using the new instructions was obtained a reduction of approximately 36% in processing time with a low area overhead.

1. Introduction

Finite Impulse Response (FIR) filtering is very important in signal processing and is one of the most commons DSP (Digital Signal Processing) algorithms, being used from medical images to wireless telecommunications applications. As FIR filtering has a computation intensive operation, and since many applications require this filtering in real-time, to have an efficient implementation of FIR filters has received a lot of interest.

RISC (Reduced Instruction Set Computer) processors shows advantages as performance, power consumption and programmability since it has only load/store instructions to reference to memory, presents fixed instruction formats and it is highly pipelined [1]. However, when RISC processors are used to implement FIR filtering it needs to make many accesses to memory, what cause a large latency. A solution in FIR filter applications is to use a circular buffer. Nevertheless, using the RISC instructions set for this design it need of many branch instructions and memory accesses. As the ARM (Advanced RISC Machine) processor used in this work has a 3-stage pipeline [2], the occurrence of branches with frequency damages this function since to each branch, the processor need to flush the pipeline. In the literature we found some works to obtain the HW reduced, low power or higher performance in FIR filtering, as shown in [3] and [4]. In DSP applications is common to use a SIMD processor, as presented in [5], but as we also want to have a RISC processor to others types of applications, we propose to use an ARM processor modified.

In order to obtain a higher performance in applications that use FIR filters we designed special instructions using an ARM processor described in VHDL (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language). We modified the ARM instructions set and added more 4 instructions to facilitate FIR filtering implementations. To analyze a FIR filter application we use the calculation of the motion compensation interpolation [6]. These instructions allow designing FIR filters obtaining a large reduction in the processing time. We compared the FIR filtering algorithm using the ISA (Instruction Set Architecture) of the ARM processor with more 4 special instructions. In the end of this paper, we showed the number of cycles necessary to implement the design and the area overhead obtained with these modifications.

In this work was necessary to know exactly as the instructions were implemented in the VHDL description to allow adding more instructions to ARM ISA. ARM processor presents the MLA instruction with the function of multiply and accumulates. This is a good option to FIR filtering, until this is the main principle of this filter. The processor described in VHDL was based in the ARM7TDMI-S [1].

This paper is organized as follows. Section 2 shows the function of FIR filtering. The news instructions added to core and the modifications in the ARM architecture are described in section 3. In section 4 are presented the results followed by conclusion in section 5.

2. FIR filtering

FIR filtering is one of the primary types of filters used in Digital Signal Processing. This filter presents some peculiar features as coefficients, impulse response, *tap* and multiply-accumulate. The filtering coefficients are the set of constants used in the multiplication with the data sample. Each *tap* execute an operation of multiply-accumulate, multiplying a coefficient by the corresponding data sample and accumulating the result. To each *tap* a multiply-accumulate instruction is executed. One *tap* involve many access in the memory. One

solution is to use a circular buffer and, for this, it needs to access two memory areas; one contained the input signal samples and other with the impulse response coefficients. The filter is implemented with a well known scheme as shown in Fig. 1, where $x[n]$ represents the input signal, $y[k]$ is the output signal, h is the filter coefficients and n refers to the number of taps used.

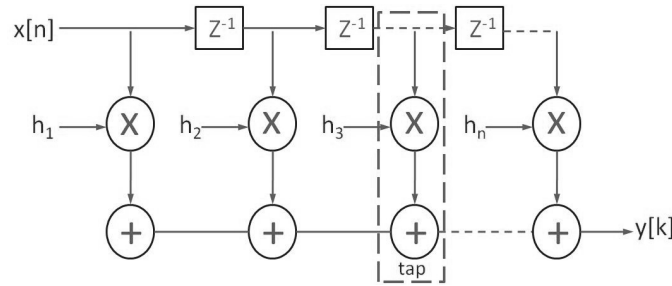


Fig. 1 – FIR filter function.

3. New ARM instructions

The instructions have been defined for the use of FIR filtering in DSP. In this work we consider as case studies the algorithm of interpolation to motion compensation used in the H.264 video encoding [6]. The motion compensation is realized in two stages, in the first stage is used a filter with 6 *taps* to find the values with $\frac{1}{2}$ pixel, in the second stage is calculated $\frac{1}{4}$ pixel realizing the average between the values calculated in the stage before.

In order to implement the new instructions correctly we analyzed similar instructions and followed the pipeline structure of the processor. The block that had more alterations was the *Control Logic*. This block is responsible to decode the instructions and send the controls signals for the others blocks. The second analysis was to verify which signals should be activated in each instruction implemented. To add new instructions, the ARM processor has an instruction codification reserved as shown in Fig. 2(a), in accordance with [2]. The bits 25 to 27 and 4 are fixed definitions for these instructions. We defined a sub-op-code to the new instructions with 2 bits (position 23 and 24), and with this, 4 new instructions could be added. Fig. 2 present the encoding for the instructions created.

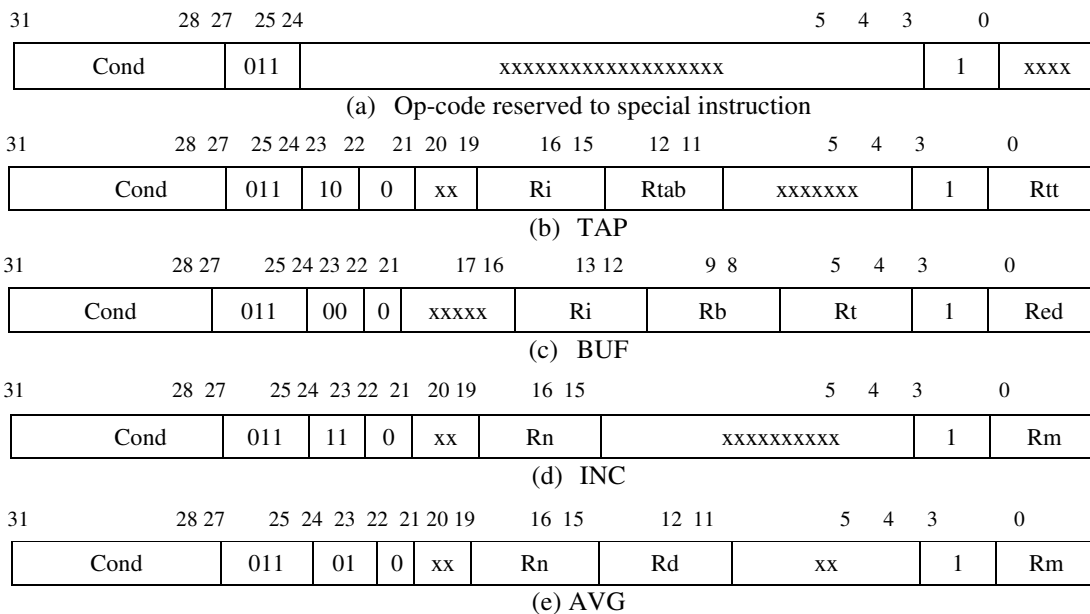


Fig. 2 – Encoding to new ARM instructions.

3.1. TAP instruction

TAP instruction realizes the control of the coefficients table pointer. This instruction is very advantageous in use of FIR filters since it points which *tap* must be used in each stage of filtering. In this instruction are used 3 registers: *Rtap*, *Ri* and *Rtt*. These registers are the same available in the ARM architecture. The codification of this instruction is illustrated in Fig. 2(b) where *Rtap* register points to the table position and it is incremented to each *TAP* instruction. *Rtt* indicates the table size and *Ri* has the address of the table top. This instruction was

created to use a circular buffer to the tap coefficients, and then it needs to verify if the address pointed by *Rtap* is the last of the table. When the address reaches the end of the table, the pointer receives the first address again. The indication that the end of table was reached is showed by the sum between *Rtt* and *Ri* that is salved in a special register, called *Rfilter*, included to the architecture to allow the *TAP* and *BUF* instructions.

3.2. BUF instruction

BUF instruction has the function to load the data of a memory position to another memory area defined to the buffer. After this, the next memory address and the buffer pointer are updated. For this instruction were need 4 registers: *Ri*, *Rb*, *Rt* and *Red*, as presented in Fig.2(c). *Ri* has the initial address of the buffer, *Rb* points to the next address of the buffer, *Rt* contains the buffer size and *Red* indicates the address where the data are in the memory. The first 4 cycles used in this instruction are similar to ARM load/store instructions. In the third cycle *Red* register is also incremented. However, more 2 cycles were need since besides to load the data of the memory, this instruction also controls the buffer address like the *TAP* instruction. In such case, *Rt* and *Ri* are summed and this result is salved in *Rfilter*. This register is used to verify when the last buffer address was reached. When the address pointed by *Rb* is equal to *Rfilter*, *Rb* receive the initial address of the buffer, otherwise, it have its address incremented. In this instruction are used 6 cycles, as the ARM core operate with 3 cycles of pipeline, it is need to use 3 NOP instructions after the BUF instruction. Even so using the BUF instruction we will be need to use 6 clock cycles, we obtain a great reduction in the total number of instructions used in the FIR filtering implementation.

3.3. INC instruction

Observing the assembly for FIR filtering we verify that the increment of two registers in the sequence is very common to control the buffer and table pointers. In such case, we implemented an instruction that increment two registers simultaneously. The op-code of this instruction is presented in Fig.2 (d) and it is realized in the same time of the ALU instructions, i. e., 3 clock cycles.

3.4. AVG instruction

AVG instruction realizes the average between two registers. This instruction was implemented duo to need verified in the interpolation of the motion compensation to calculate $\frac{1}{4}$ of pixel, in conformity with the H.264 encoder [6]. To implement this instruction, as it is similar to *ADD* instruction, we use the same control signals used in ALU instructions. When is an *AVG* instruction, the result is shifted to right in order to obtain the value divided by 2. To this verification is sent a control signal from *Control Logic* to *ALU* defining to be *AVG* instruction. Fig. 2 (e) shows the op-code used in this instruction.

4. Results and Comparisons

To obtain the instructions we add some logic to architectural blocks of the ARM as in the *Control Logic*, in the *Registers Bank* and in the *ALU*. Besides, we create the *Rfilter* register to the *TAP* and *BUF* instructions since only two registers can be access to each time in the *Registers Bank*. Besides, we need to include another adder to realize the sum between the registers used to identify the end of the buffer. This adder was created in order to maintain the correct function of the pipeline. The others instructions use only the blocks present in the ARM processor. All news instructions were created in accordance with the pipeline of the ARM architecture, i. e., in 3 clock cycles. Except the *BUF* instruction that use 6 clock cycles because it realize many functions. With pipeline, after it is full, all instructions are executed in 1 clock cycle. Thus, to realize the average between 2 values it would be need use the *ADD* instruction followed of the *LSR* instruction that shift the value to right to realize the division by 2. Using the *AVG* instruction we gain 1 cycle to each $\frac{1}{4}$ pixel calculated. We obtain the same gain with the *INC* instruction since without this option we need to use 2 *ADD* instructions. Then with this new instruction we can save 1 cycle to each occurrence of the double register increment. This necessity occurs frequently in applications with many memory accesses, as happens in the FIR filtering application.

BUF instruction needs of 3 *NOP* instructions to complete its execution, and then it is executed in 3 pipeline clock cycles. Even so, we economize many cycles using this instruction. If we utilize ARM instructions to the same function of the *BUF* instruction, we will need use 9 instructions (7 clock cycles or 9 clock cycles, depending of the branch). With the new instruction we reduced 4 cycles to each use of this instruction. As in a FIR filtering we would need use this instruction to each data loaded of the memory, the cycles reduced present a great advantages with this proposal.

The *TAP* instruction was implemented in 3 clock cycles in accordance with the ARM pipeline, then, if we use the ARM ISA to execute the same function, we need of 6 instructions, that represent 6 clock cycles. In such case, with the new instruction, we reduced 5 clock cycles. In the same manner, in a FIR filtering application, we need to use *TAP* instruction to each use of *tap*. We verify the correct function of these instructions through of simulation using the Modelsim Tool.

We analyze the FIR filter algorithm using the ARM ISA with the new instructions. With the ARM ISA are used 96 cycles for the calculation of each $\frac{1}{2}$ pixel however, using the ARM ISA with the new instruction are need only 62 cycles considering the FIR filter with 6 taps as used in the motion compensation interpolation [6]. Besides we realized an estimative of the number of cycles necessary to the calculation of the motion compensation interpolation to one HDTV frame. Each HDTV frame is composed of 1920×1088 pixels, then 2,085,953 (1919×1087) calculations are realized to find the $\frac{1}{2}$ pixel values. With the AVG instruction we save 50% of the cycles necessary to calculate the $\frac{1}{4}$ pixel. Tab. 1 present the gains obtained with the new instructions. In this table we realized an estimative to a processing rate of 30 frames/s that represent the rate necessary to execute HDTV in real-time.

Tab. 1 – Number of cycles used to calculate $\frac{1}{2}$ pixel and $\frac{1}{4}$ pixel using ARM ISA and ARM ISA with new instructions.

	ARM ISA	ARM ISA with new instructions	Cycles Reduction (%)
$\frac{1}{2}$ pixel (cycles)	6.00×10^9	3.88×10^9	35.5%
$\frac{1}{4}$ pixel (cycles)	125.16×10^6	62.58×10^6	50.0%
Total	7.25×10^6	4.44×10^9	35.7%

In accordance with tab. 1 we verify a large reduction obtained with the instructions proposed. In the application used as case study we use only 6 *taps*, but there are applications with a large taps number, obtaining a much greater reduction in the processing time with the use of the instructions proposed. However, to reach these results, as commented previously, we need to add/ alter the logic of the ARM Core. We synthesized the ARM architecture using the ISE 8.1 tool to device Virtex4 XC4VLX200 Xilinx. Tab.2 show the area results obtained for both architectures. The maximum frequency was the same for both architectures, being this approximately 43MHz.

Tab.2 – Area results to Virtex 4 XC4VLX200 device.

	ARM ISA	ARM ISA with new instructions	Area Overhead (%)
Slices	3599	4517	25.5%
Flip-Flops	1855	1890	1.8%
LUTs	6834	8525	24.7%

5. Conclusion

Processing a FIR filter using a RISC processor brought us as motivation to obtain the advantageous of a general-purpose processor to DSP applications. In this work we showed four news instructions based in an ARM Core. These instructions were designed to be implemented in FIR filtering applications. Besides to reduce drastically the processing time in FIR filter design, with this alternative, it is possible to have a RISC processor available to any another application. With the instructions proposed, the reduction in the processing time to Motion Compensation Interpolation was of approximately 36%. To obtain these instructions we add some HW in the ARM architecture. This work also presented as advantageous to know with more details the ARM VHDL architecture used, what will benefit others implementations with this processor.

6. References

- [1] ARM7TDMI Technical Reference Manual. ARM, Tech. Rep., Mar 2004: <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0029g/DDI0029.pdf>
- [2] ARM Instruction Set: http://www.wl.unn.ru/~ragozin/mat/DDI0084_03.pdf
- [3] J. Park et al., "High performance and low power FIR filter design based on sharing multiplication" in Low Power Electronics and Design, 2002. ISLPED '02, 2002. pp. 295- 300.
- [4] P. Bougas et al., "Pipelined array-based FIR filter folding" in IEEE Transactions on Circuits and Systems I, vol 52, 2005, pp. 108 – 118.
- [5] G. Kraszewski, "Fast FIR Filters for SIMD Processor with Limited Memory Bandwidth, Proceeding of XI Symposium AES, New Trends in Audio and Video" Bialystok, 2006, pp. 467-472.
- [6] B. Zatt et al., "High Throughput Architecture for H.264/AVC Motion Compensation Sample Interpolator for HDTV" in Symposium on Integrated Circuits and System Design (SBCCI '08), 2008, pp. 228 – 232.

Modeling Embedded SRAM with SystemC

Gustavo Henrique Nihei, José Luís Güntzel

{nihei,guntzel}@inf.ufsc.br

System Design and Automation Lab (LAPS)
Federal University of Santa Catarina, Florianópolis, Brazil

Abstract

This paper presents and evaluates two models of SRAM (Static Random Access Memory) developed in SystemC language. One model is able to capture the behavior of SRAM blocks at abstract levels, thus being more appropriate to model large pieces of memory. The other one is devoted to model memory blocks at the register-transfer level (RTL). Due to its capability of capturing more details, the latter model may be used, along with appropriate physical models, to provide good estimates of memory access time and energy. Both models were validated and evaluated by using complete virtual platforms running true applications. Experimental results show that the simulation runtimes of the RTL model is acceptable when the size of the memory is moderate (up to 1024kB). Considering such scenario, both memory models may be used together to model the whole memory space available in a hardware platform.

1. Introduction

A significant portion of silicon area of many contemporary digital designs is dedicated to the storage of data values and program instructions [1]. According to [2], more than 80% of the area of an integrated system may be occupied by memory and it is estimated that in 2014 this area will correspond to about 94%. The inclusion of memory blocks on chip increases data throughput in the system and therefore decreases the amount of access to off-chip memory, reducing the total energy consumption.

Unlike performance and area, which for many years can be measured in the design flow, there is still a lack of techniques and tools to estimate power with a good compromise between accuracy and speed, making this an area of intensive research, especially in higher levels of abstraction. The increasing complexity of circuits, the need to explore the design space and the shortening of time-to-market cycle require new design strategies, as component reuse and the adoption of more abstract representations of the system [3].

Traditional system design methodologies fail to meet the need to integrate so many features in a single system. The design complexity – both hardware and software – is the main reason for the adoption of development methodologies for electronic system level design [4]. System-level languages, such as SystemC [5], enable the design and the verification at the system level, independent of any details concerning specific hardware or software implementation. These languages also allow the co-verification with the RTL design.

This paper presents two models for SRAM blocks to be used in system-level design. The models were developed in SystemC and their construction and evaluation follow a meet-in-the-middle approach [6]. One model was developed at the functional level and has a transaction-based communication interface. By using such model, large memory blocks can be modeled allowing the whole system simulation from its SystemC description. The second model was developed at the register-transfer level (RTL) and hence is able to capture details of the physical structure of SRAM blocks. Therefore, the RTL model can be used to model on-chip memory blocks, as those used for caches and scratchpad memories, allowing accurate estimates of specific metrics, such as access time and energy. Both the functional level and RTL SRAM models were integrated into a previously designed virtual platform consisting of processors and a bus. This way, the SRAM models were validated and evaluated by using real applications running in a complete platform. The total time required to simulate the applications on the platform containing the developed SRAM models are presented and compared.

2. Modeling the memory block with SystemC

Fig. 1 shows the internal structure of a typical SRAM block [1]. In the RTL model the memory block is modeled as a set of 32-bit words. These words are organized in an M by N structure, whose aspect ratio can be easily reconfigured. Since the memory is word-addressed, it can be considered to have a three-dimensional structure. The developed model is synchronized by the clock signal. It has a bidirectional data port. The width of the data and address rows is 32 bits.

As shown on fig.1, the controller unit is the interface of memory with the outside world and it is also responsible for the control of the decoders. There are two decoders, one for row decoding, and another for column decoding. The word decoding process takes 2 processor cycles and is done by logic and arithmetic operations. Both the controller unit and the decoders operate as state machines synchronized by the clock signal.

Each word was designed as a SystemC module, and represents a 32-bit block with separate ports for data, address and control signals. Internally, data storage is structured by four 8-bit vectors. The memory words in RTL were modeled as asynchronous devices. A word module will only operate when both decoder lines are activated. A synchronous implementation was also considered. In spite of the similar behavior, the CPU usage would be significantly higher, because every word module would be activated in every clock transition, and each one of them would have to check its ports.

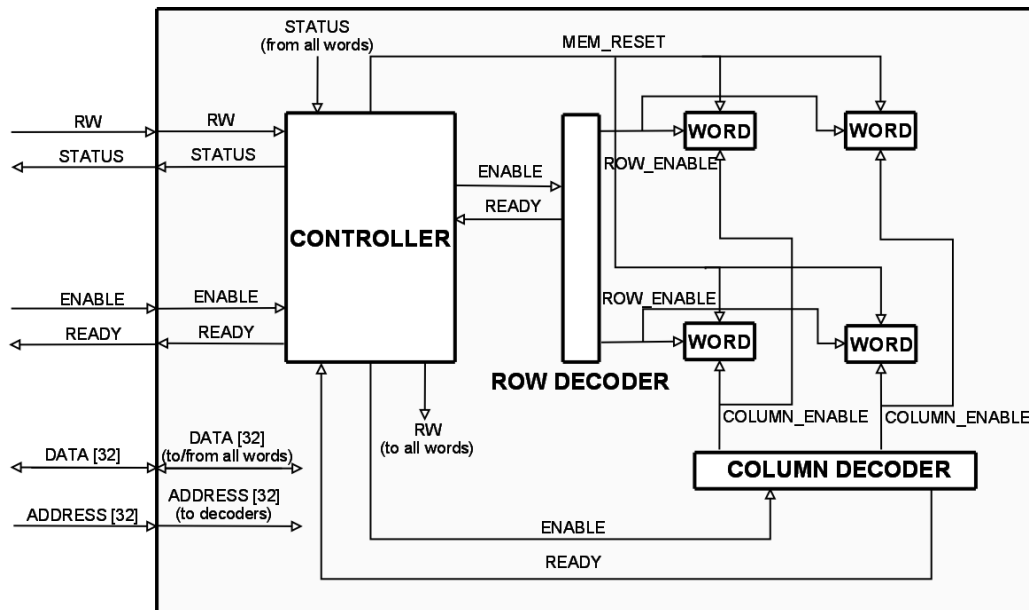


Fig. 1 – RTL model's block diagram

In the functional model the storage unit was modeled using a single array, with 1-byte elements. In this higher abstraction level, the memory has a linear aspect ratio. In a read operation, for example, the address points directly to the first byte of the word to be retrieved. Then, the next four bytes are written on the data port.

3. Infrastructure for the validation and evaluation of the models

Once the description of a hardware block is done, the next step, regardless of the level of description, is to check if it behaves as expected. Manual creation of a set of stimuli and results to check model correctness is an extremely tedious task that results in low productivity and may lead to error. To circumvent this problem, we use the idea of golden model. A golden model is a reference model that meets the previously specified requirements.

In order to validate the two developed memory models, we have created two versions of a platform by using the infrastructure provided by ArchC Reference Platform (ARP) [7]. This infrastructure allows the designer to simulate an application running on a complete model of the hardware, containing processor(s), memory and interconnection network. It is also worth to mention that the use of a platform makes the reuse of models easier, which was properly explored for the creation of both versions of platform used in model validation. These versions are shown in fig. 2.

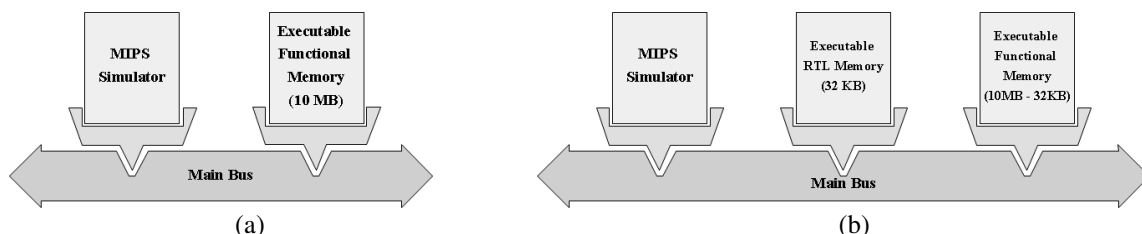


Fig. 2 – Versions 1 (a) and 2 (b) of the virtual platform, used to validate the two SRAM models

A first version of platform (fig. 2a), hereafter called “platform A”, was developed to validate the SRAM functional model. In this platform a single SRAM block modeled in the functional level is connected to the bus. This memory block covers the whole address space of the MIPS processor (allowed by the ARP), which ranges from 0x000000 to 0x9FFFFFF.

To validate the SRAM RTL model a second version of platform (fig. 2b) was created. This version, hereafter referred to as “platform B”, has two memory blocks connected to the bus, each of them modeled at a

different level of abstraction: a 32kB SRAM block, modeled at the RTL, and a (10MB minus 32kB) SRAM block, modeled at the functional level. The 32kB block has a square aspect ratio (words organized in 512 rows and 16 columns) and covers the address space spanning from 0x000000 to 0x007FFF. The (10MB minus 32kB) SRAM block covers the remaining address space.

4. Validation and evaluation results

The SystemC simulations used to validate and to evaluate both platforms were run on a computer equipped with a 2.33GHz Intel Core 2 Duo (E6550) processor, 2GB of main memory and Ubuntu Linux 32 bits (kernel 2.6.24) operating system. The software used in the development of the experiments are ArchC 2.0, SystemC 2.1.v1, SystemC TLM 2.0 (2008-06-09), GCC 4.2.3 and MIPS cross-GCC 3.3.1.

To run the SystemC simulations the four following application were selected from the MiBench benchmarks suite [8]:

- cjpeg, with the file input_small.ppm as stimulus,
- sha, with the file input_small.asc as stimulus,
- susan, with the file input_small.pgm as stimulus,
- bitcount, with the param 75000 as stimulus.

The application is compiled for the target processor (MIPS simulator) and loaded into the memory available in the platform, starting from the address 0x000000. This process is executed for both platforms, once for each application.

Fig. 3 shows the execution flow for the validation process. For each of the selected applications, the corresponding stimulus is provided for both the golden model and the platform under validation (platforms A and B). The golden model is a previously validated reference platform [9]. The outputs of the platform version under validation are compared to the outputs of the golden model. If both have the same result, the platform under validation behaves correctly. Otherwise, the errors are fixed and the validation flow is rerun.

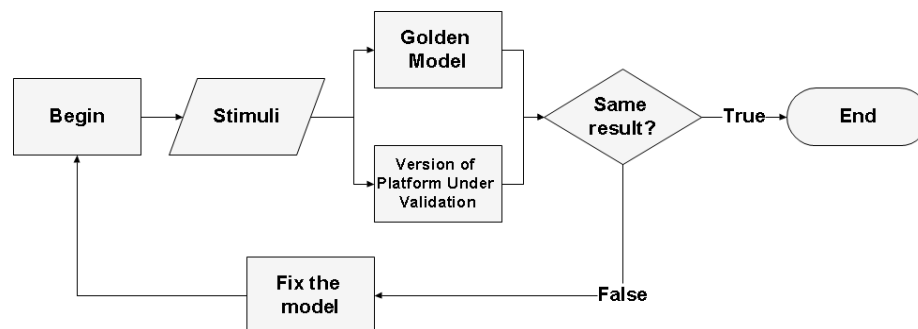


Fig. 3 – Execution flow for the validation of the two versions of platform

Once both versions of platform were validated, the runtimes resulting from their simulations with the four selected applications were stored. These runtimes are shown in tab. 1. As one could expect, the runtimes of platform B is longer for all considered applications. This is expected, since platform B has one part of the memory modeled at the RTL. As explained in section 2, the RTL SRAM model has several units modeled as sequential machines. Even disabled, they still continue to consume CPU time, because the ports must be constantly checked for new values.

The increase in runtime ranges from 2.76 to 15.38, with average increase equals 7.23. Although this is the price to be paid for a more detailed SRAM model, these values are not prohibitive, mainly when one consider the benefits arriving from the use of the RTL model, which allow better estimates of memory access time and energy (if appropriate physical models are used).

Tab.1 – Simulation runtimes for platform A and for platform B

Benchmark	Platform A: functional SRAM model only [sec]	Platform B: RTL+functional SRAM models [sec]	Increase in runtime (B)/(A)
SHA	14,54	81,41	5,60
susan	4,44	72,74	15,38
cjpeg	14,98	77,26	5,16
Bitcount	45,04	124,19	2,76

A second experiment was conducted to investigate the amount of main memory needed to simulate platform B and how it depends on the size of the SRAM block modeled at RTL. To evaluate such dependency platform B was simulated with cjpeg application in six scenarios, each of them with a different size of RTL SRAM block

(from 32kB to 1024kB). Tab. 2 shows the information on main memory. By observing tab. 2, one can infer that the main memory needed for simulation grows proportionally to the size of memory to be simulated. As previously stated, this is due to the large number of instantiated modules. For the simulation of 32 KB, for instance, there are 8,192 SystemC modules only for the storage block.

Tab. 2 –Memory footprint for the simulation of platform B

Size of Memory block modeled at RTL (KB)	Main memory needed for simulation (MB)
32	56
64	107
128	212
256	424
512	846
1024	1694

5. Conclusions and future work

Good estimates of memory access time and energy during system simulation require accurate models of SRAM memory blocks. This paper has evaluated the runtime and the total main memory required to simulate two hardware platforms that use two different models of SRAM memory blocks: a functional level model and an RTL model. The obtained results showed that the increase in simulation time resulting from the use of the RTL model is acceptable if we consider the benefits coming from this accurate model and if the size of the block to be modeled at this level is moderate. By moderate-sized blocks it is meant blocks with up to 1024kB, if one use a workstation similar to that described in section 4. This way, if the total memory available in the platform surpasses the memory modeled at the RTL, then the rest of the addressable space must be modeled at a more abstract level, as the functional level. It is important to remark that this behavior is a consequence of the limitations of the SystemC simulation kernel.

From the considerations stated in the latter paragraph it is possible to speculate that the RTL model is more appropriate to model on-chip caches and scratchpad memories (which are very frequently accessed), while the functional model is more suited to off-chip memory and to large on-chip memory blocks.

As future work it is envisaged some possible improvements at the memory models, as the implementation of an access time controller. Another possible work is the integration of the RTL memory model with a physical memory model as CACTI [10], in order to capture realistic information on access time and energy.

6. References

- [1] J.M. Rabaey, A. Chandrakasan, B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd. ed. [S.l.]: Prentice-Hall, 2003. 623-719 p.
- [2] E.J. Marinissen et al. "Challenges in Embedded Memory Design and Test", *Proc. Design, Automation and Test in Europe (DATE '05)*. Washington, DC, USA: IEEE Computer Society, 2005. p. 722–727. ISBN 0-7695-2288-2.
- [3] R. O. Leão. *Análise Experimental de Técnicas de Estimativa de Potência Baseadas em Macromodelagem em Nível RT*. Master Thesis – Federal University of Santa Catarina, May 2008.
- [4] B. Bailey, G. Martin, A. Piziali, *ESL Design and Verification*, 1st. ed. [S.l.]: Morgan Kaufmann Publishers, 2007.
- [5] SystemC, Open SystemC Initiative. Oct 2008; <http://www.systemc.org/>
- [6] P. Marwedel. *Embedded System Design*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 1402076908.
- [7] ARP: ArchC Reference Platform; <http://www.archc.org/>
- [8] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite", *Proc International Workshop on Workload*, 2001, pp 3-14.
- [9] ARP Dual MIPS Example; http://143.106.24.201/~archc/files/arp/dual_mips.01.arppack
- [10] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," Hewlett-Packard Laboratories, Technical Report, Apr. 2008.

Implementing High Speed DDR SDRAM memory controller for a XUPV2P and SMT395 Sundance Development Board

Alexsandro C. Bonatto, Andre B. Soares, Altamiro A. Susin
{bonatto,borin,susin}@eletro.ufrgs.br

Signal and Image Processing Laboratory (LaPSI)
Federal University of Rio Grande do Sul (UFRGS)

Abstract

This paper presents reuse issues in the reengineering of a double data rate (DDR) SDRAM controller module for FPGA-based systems. The objective of the work is the development of a reusable intellectual property (IP) core, to be used in the project of integrated systems-on-a-chip (SoC). The development and validation time of very complex systems can be reduced with the use of high level description of hardware modules and rapid prototyping by integrating modules into a FPGA device. Nevertheless, delay sensitive circuits like DDR memory controllers not always can be fully described by using a generic reusable code language. The external memory controller has specific timing and placement constraints and its implementation on a FPGA platform is a difficult task. Reusable hardware modules containing specific parameters for optimization are classified as firm-IPs. With our approach, it is possible to generate a highly reconfigurable DDR controller that minimizes the recoding effort for hardware development. The reuse of the firm-IP in two hardware platforms is presented.

1. Introduction

Systems-on-a-Chip (SoC) can integrate different hardware elements as processors and communication modules over the same chip. Each hardware element is a closed block defined as an intellectual property (IP) module. The reuse of IP modules in the construction of different SoCs is a fast way to reach system integration. The new generation high-performance and high-density FPGAs become prototyping devices suitable for verification of SoC designs. Using hardware description languages (HDL) as Verilog or VHDL as design-entry adds some benefits as it is a technology-independent system description that can be reused with a minimum of recoding. If logic elements are not instantiated from specific device libraries, hardware reuse can be easily reached. The hardware designer must use coding strategies in order to write a design-entry source code that can be synthesized. Thus, the understanding of the synthesis tool process and the knowledge of the FPGA features are important to design the hardware module. Also, the hardware developer must know the placement and timing parameters to reach the system specifications.

The DDR SDRAM controller uses architecture specific primitives and macros that are instantiated in the HDL from the vendor library models. A synthesis tool is not able to translate these specific macros in the design-entry by inference. Then, some recoding effort is necessary when reusing the controller IP on a different platform. Such features characterize a *firm-IP* module, which targets a specific architecture and has a higher level of optimization. Firm cores are traditionally less portable than soft cores, but have higher performance and more efficient resources utilization because they are optimally mapped, placed and routed. Moreover, optimization or unconstrained routing by synthesis and routing software may change critical timing relations for correct hardware behavior.

This work presents the implementation results of the DDR controller in two different FPGA development platforms, interfacing with an external DDR SDRAM. This paper is organized as follows: section 2 presents an overview of the DDR memory and controller; section 3 shows the controller implementation targeted to reuse and experiments and in section 4 the conclusions are discussed.

2. DDR SDRAM Controller IP

In this section it will be introduced the main characteristics of DDR SDRAM memory and controller [1]. Double data rate memories contain three buses: a data bus, an address and a command bus. The command bus is formed by the signals column-address strobe (CAS), row-address strobe (RAS), write enable (WE), clock enable (CKE) and chip-select (CS). The data bus contains the data signals (DQ), data mask (DM) and data strobe signals (DQS). Address bus is formed by address (ADDR) and bank address (BA) signals. These memories operate with differential clocks CK and CKn, which provides source-synchronous data capture at twice the clock frequency. Data is registered either in the rising edge of CK and CKn. The memory is command activated starting a new operation after receive a command from the controller.

Data words are stored in the DDR memory organized in banks, indexed with row, column and bank addresses. Fig. 2 illustrates the timing diagram for a RD operation in DDR memory. Data is transferred in bursts, sending or receiving 2, 4 or 8 data words in each memory access.

To access data in a read (RD) or write (WR) operation, the controller first sets the row address, this is called as row-address strobe (RAS) command (Step #1). After, the memory controller sets the column address, called as a column-address strobe (CAS) command (Step #2). In the case of a RD operation, data is available after the CAS Latency (CL) which can be 2, 2.5 or 3 clock cycles (Step #3). The data words D0:D3 are transmitted edge aligned with the strobe signal DQS after the CAS latency. DQS is a bidirectional strobe signal used to capture data DQ. Dynamic memories, as the DDR SDRAM, require refreshing the stored internal data periodically. This operation is performed by the memory controller using the auto-refresh command.

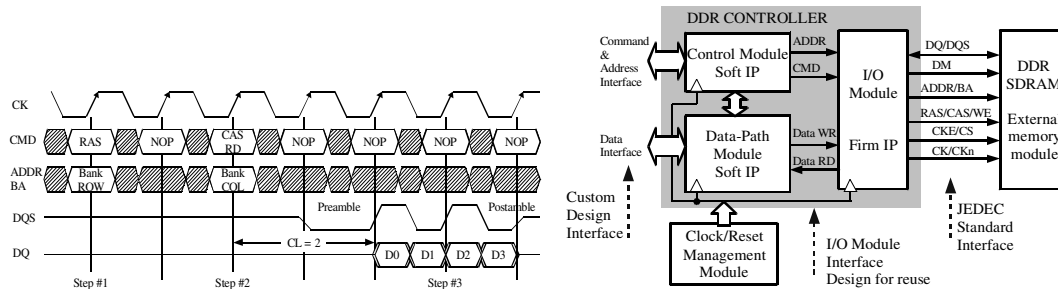


Fig. 2: (a) Timing diagram of reading data from DDR memory with CAS latency 2 and burst length 4;(b) DDR controller block diagram.

2.1. DDR controller architecture

The DDR controller contains the logic used to control data accesses and the physical elements used to interface with the external memory. Its design objectives the creation of a configurable IP module, to be used into different SoC designs, allowing the configuration of: DQ data bus width and the number of DQS strobes; the data burst length; the CAS latency; the number of clock pairs, chip-select and clock-enable signals to memory. The DDR controller architecture is structured in three sub-blocks, as illustrated in fig. 2b:

Control module – controls the data access operations to external memory translating the user commands and addresses to the external memory. Also it controls the write and read cycles and the data maintenance generating the auto-refresh command periodically. It is implemented as a soft-IP module described in HDL;

Data-path module – data sent and received from DDR memory are processed by this module. In transmission, data are synchronized using the double data rate registers. In reception, data are stored into internal FIFOs and it is synchronized by the controller internal clock. It is implemented as a soft-IP module described in HDL;

I/O module – contains the I/O and 3-state buffers located at the FPGA IOBs device and the double data rate registers used to write and read data, commands and to generate the clock signals to memory. It is a firm-IP module described in HDL with pre-defined cell and I/O location. Comparison of different technologies are shown in [2];

DDR Controller Clock Generation – four different clock phases are needed to sample correctly control and data signals [3]. Clock generation is a technology dependent resource and each FPGA manufacturer use different embedded modules to make internal clock management. Also, best design practices recommend that the clock generation and control has to be done externally to the DDR controller, in a separated module.

2.2. Read data capture using DQS

The most challenging task in interfacing with DDR memories is to capture read data. Data transmitted by DDR memory are edge aligned with the strobe signal, as is illustrated in fig. 2a. The controller uses both strobe signal transitions to capture data, rising and falling. The strobe signal must be delayed 90° with respect to DQ to be aligned with the center of data valid window.

In fig. 3 is illustrated the circuit topology used to capture the data read from DDR memory. A Delay Line is inserted into the DQS path in order to shift the strobe signal by 90°, as described in [4]. The DQS strobe timing pattern consists of a preamble, toggling and postamble portion, as illustrated in fig. 2a. The preamble portion provides a timing window for the receiving device to enable its data capture circuitry. The toggling portion is used as clock signal to register data using both strobe edges. The amount of delay between data and strobe signals has to satisfy minimum and maximum values to successfully capture data send by the external memory device. In the implementation step, the data and strobe delays are controlled not only by the delay-line size but also by the internal routing delay.

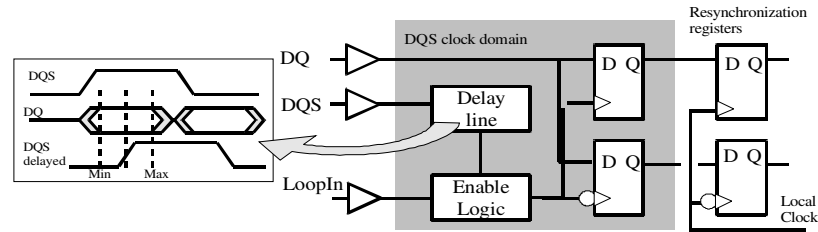


Fig. 3: Circuit topology to capture the data read.

The Enable Logic is used in the data capture circuit to avoid false switching activity at the capture registers. D-type registers synchronized with the delayed strobe signal are used to capture read data from memory. The external loopback signal is used to enable the data capture circuit as the same instant that the valid switching portion of DQS signals arrive to the controller. This signal is generated by the control module when it is performed a read operation.

3. Implementation with controller reuse

The DDR memory controller can only be considered validated after its successful operation on an actual circuit. The prototyping procedure and results are presented in this section. The design for reuse of the DDR controller, considering principally Xilinx's devices. The original project of the DDR controller is presented by Xilinx in [3]. From this first implementation, modifications were done in its VHDL source code with objective to reduce the quantity of used logical resources and to simplify the reuse, as presented before in [2].

Timing constraints are used to force the routing step to meet the system timing requirements for the FPGA implementation without force manual placement of controller technology-dependent, which reduces reuse of constraints. Also, they are used to set the system clock speed and the maximum allowed delay in specific nets. In the DDR controller implementation, the constrained nets are the DQS delayed used as clock to capture read data. If the user chooses another synthesis tool to implement the controller, the constraints file must be modified.

3.1. Controller implementation on the Digilent XUPV2P prototype board

The DDR controller was firstly implemented in a Digilent XUPV2P board, using a XC2VP30 device. The I/O module was implemented in structural format as a *firm*-IP by instantiating individually defined technology-dependent elements from the Xilinx Unisim library, for the Virtex-2 Pro FPGA family. The SSTL driver buffers, the DDR registers and the logic elements used to implement the delay line circuit are grouped in the *firm* module. Also, the clock generation circuit is realized by instantiating one DCM from the Unisim library in the VHDL code, on a module separated from the DDR controller IP.

The Digilent XUPV2P board contains a Virtex-2 Pro FPGA XC2VP30, a 100 MHz clock oscillator and 512 MB of external DIMM DDR SDRAM module. Before the board implementation, both functional and timing simulations were performed using Modelsim XE 6.0a software. The controller was implemented for DQ 64-bit wide, DQS 8-bits wide and achieves frequencies higher than 220 Mhz.

3.2. Controller implementation on the Sundance SMT395 prototype board

The Sundance SMT395 prototype board contains a Virtex-2 Pro FPGA XC2VP70, connected to four external onboard DDR SDRAM memory modules, models K4H511638B from Samsung. The memory modules are 16-bit data wide with two strobe signals and each strobe signal is used to synchronize eight data signals. The design-entry recoding to adapt the controller to another board begins with changing the data bus width, through the definitions declared with generics at the top-level entity. The DDR controller implementation is automatically adapted to the new board using *generics* in the to entity VHDL code. This includes the data bus wide configuration and also the number of clock signal pairs.

There are differences between prototyping boards which also can affect IP development and must be considered also. Some boards do not have the external loopback path to enable data capture from strobe signals. The loopback signal, as illustrates fig. 3, acts as a reference for the delay of a signal that is transmitted from the FPGA to the memory module and back to the FPGA. In this case, the signal must be modeled by an equivalent delay, inside the FPGA. Other differences are the number and width of memory modules, pin position, clock frequency (125 MHz) and data read latency (CL2) accepted by the memory in that operation frequency.

Other internal logic elements instantiated in the code as the Delay Line, the DDR registers and the STLL drivers, are maintained as the previous implementation in the XUPV2P board. Both FPGAs are from the same family and it is not necessary to perform manual replacement by the user. The user constraint file (UCF) used to guide the implementation tools at the placement and routing steps is modified to adapt the IO pins and specific

logic elements location to the new implementation. The IO pins location is adapted to the new board. In tab. 1 shows the results for the DDR controller implementation on both XUPV2P and SMT395 boards.

Tab.1 – DDR controller synthesis results.

Sub-block	Slices	Flip-flops	LUTs	# lines
Control	156	187	268	1559
Data-path	802	931	755	981
I/O	67	180	134	824
DDR controller	580	1299	1135	4111

As a measure of the reusability of the code, one can count the number of lines that the designer can keep from a design to the next one. The recoding effort in adapting the DDR controller to a new design is done over about 20% of the total source code lines, disregarding the clock generation module. The controller's soft sub-blocks are the control and the data-path and the firm sub-block is the I/O, which represents 11% of all the slices used by the controller. The synthesis results show that the reusable code part is most significant in this design.

3.3. Verification and board validation

The DDR controller verification is complete after the test over the development board. Both functional and timing simulations are not enough because they do not take into account the external propagation delays. First, for the design implementation validation, static timing analysis (STA) was performed at the circuit resulted from the place&route implementation step. For both FPGA implementations, STA can be used to verify if the internal routing delays in data and strobe paths do not exceed the limits tolerated for the implementation. In this work, the same verification methodology presented before in [5] was used. A built-in self-test (BIST) module was designed to validate the DDR controller implementation in the FPGA board.

4. Conclusions

This paper presented an approach to implement an reusable memory controller IP module in different development boards. The reuse of hardware modules for rapid system prototyping require minimum non recurrent engineering effort by structuring the design and isolating elements from vendor libraries at defined device locations into firm modules and redesigning the maximum number of elements as technology independent HDL code, in soft modules.

The migration from two different prototyping platforms showed that only very specific parts of the IP like delay lines needed to be reprojected, being already identified in a fast verification of the firm module. Other modules like the controller logic and state machines are encapsulated on soft modules and reused as-it-is into new designs. Also, as we have presented in this paper, the knowledge of the limitations of the synthesis tool is important to guide the implementation in order to meet the design requirements. Also, the integration of a BIST module with the controller is very useful to allow automatic test of the IP implementation when reusing it in other systems. The test approach uses the DDR controller IP, the interconnections between memory and FPGA and the external memory as a single circuit under test.

5. References

- [1] JEDEC, JESD79: Double Data Rate (DDR) SDRAM Specification, JEDEC Solid State Technology Association, Virginia, USA, 2003.
- [2] A. C. Bonatto, A. B. Soares, A. A. Susin, "DDR SDRAM Controller IP Designed for Reuse," in: *IP'08 Design & Reuse Conference*, Grenoble, França, Dez. 2008.
- [3] Application Note 802: Memory Interface Application Notes Overview, Xilinx, 2007.
- [4] K. Ryan, "DDR SDRAM functionality and controller read data capture," *Micron Design Line*, vol. 8, p. 24, 1999.
- [5] A. C. Bonatto, A. B. Soares, and A. A. Susin, "DDR SDRAM Memory Controller Validation for FPGA Synthesis," in *LATW2008: Proceedings of the 9th IEEE Latin-American Test Workshop*, Puebla, Mexico, Feb. 2008, pp. 177–182.

Data storage using Compact Flash card and Hardware/Software Interface for FPGA Embedded Systems

¹Leonardo B. Soares, ¹Vitor I. Gervini, ²Sebastião C. P. Gomes, ¹Vagner S. Rosa
{leobsoares, gervini, scpgomes, vsrosa}@gmail.com

¹Centro de Ciências Computacionais – FURG

²Instituto de Matemática e Física – FURG

Abstract

This paper presents a data storage solution for embedded systems developed in FPGA (Field Programmable Gate Array) boards. Actually, a lot of projects need to store data in a non-volatile memory. This paper presents a technique to store data using a Compact Flash card and the FAT16 file system and compare this technique with a solution where data is always transferred from a host system whenever is needed. The application proposed here is part of a major project developed by NuMA (Núcleo de Matemática Aplicada) based on a FPGA board. The application presented here as a case study stores data for trajectories that command a robot. These trajectories are sent by a serial interface, so is necessary to save some pre-defined trajectories to save some embedded system's resources and time or is essential users can save results about experiments with a robot. This application was developed using a Digilent XUP V2P development board.

1. Introduction

The project developed by NuMA (*Núcleo de Matemática Aplicada*) controls a robot that has five degrees of freedom. It was designed using a FPGA (*Field Programmable Gate Array*) board. This board has a software/hardware interface that commands a control law, electrical installations, and DC/AC motors as shown by Diniz [1] and Moreira and Guimarães [2]. Sending trajectories by a host computer using a serial bus linked to the board is needed to move the robot. The trajectories are very accurate and can take several minutes to download in the embedded system. These trajectories can be the same for a large set of experiments, but if it is stored in RAM, a new download is needed whenever the system is turned off. For this reason, it is very important that the embedded system can store some trajectories in a non-volatile memory.

The XUP V2P development board is a very versatile system. This board has a 30K logic cells FPGA and a lot of interfaces resources. One of these is a port that supports compact flash card. This port is controlled by a device controller called System ACE (System Advanced Configuration Environment). This controller would be the hardware interface and should be configured to enable electronics functionalities of compact flash port. Still, there is the software interface that includes an implementation in C language and specifics libraries to manipulate FAT16 file system's operations.

The software used to set hardware/software and to enable the compact flash card is the EDK (Embedded Development Kit) from Xilinx. It is a very good applicative to manipulate all functions in a FPGA board. This program has an interface that does an important hardware/software abstraction. Using its functionalities is an easy way to set electronics settings and to implement the embedded system's software.

2. System ACE Description

System ACE includes the hardware and a software interface. In the hardware interface, it is a device controller that commands compact flash card's functionalities. This controller was developed by Xilinx and its structure can be comprised in two parts. The former is a Compact Flash controller and the second is a Compact Flash Arbiter and this structure is detailed by fig.1.

The Compact Flash controller both detects if exists a card attached in the board and perform a scanning in the compact flash status. Still, this controller handles the Compact Flash access bus cycle and implements some level of abstraction through the implementation of commands like reset card, write, and read sectors. The Compact Flash Arbiter controls JTAG and MPU access in the data buffer located in Compact Flash.

The System ACE supports ATA Common Memory read and write functions and cards formatted with FAT12 or FAT16 filesystem. The filesystem support is implemented by the software part of the controller in EDK libraries and this makes the filesystem accessible by PCs.

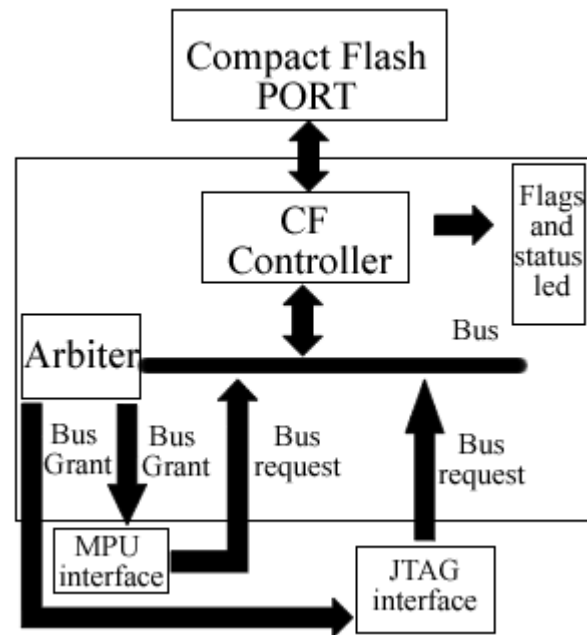


Fig. 1 – System ACE structure

3. Method Proposed

The EDK developed by Xilinx is a toolkit that makes possible integration between hardware and software. It shows the buses and devices installed on the embedded system. All devices are implemented in VHDL (Very high speed integrated circuits Hardware Description Language) and they are called IP (Intellectual Property). The software implementation was written in C language and some libraries from EDK were used to do the final application. The library that abstracts the Compact Flash's functions is xilfatfs. This library implements functionalities like read from the card, write into the card, open a file, and all functions supported by FAT16 file system. Then, the user must format the card with FAT16 file system.

The software implemented is very simple. It permits to manipulate files with a high level of abstraction. It is a loop that waits for a command from the user. Then, if the user wants read from the card he chooses this option. One thing that should be respected by the user is the FAT16 standard. For example, this file system supports file names in 8.3 format (maximum eight characters to name and three to the extension). For more details see the fig.2 that reveals all of operations supported by this application.

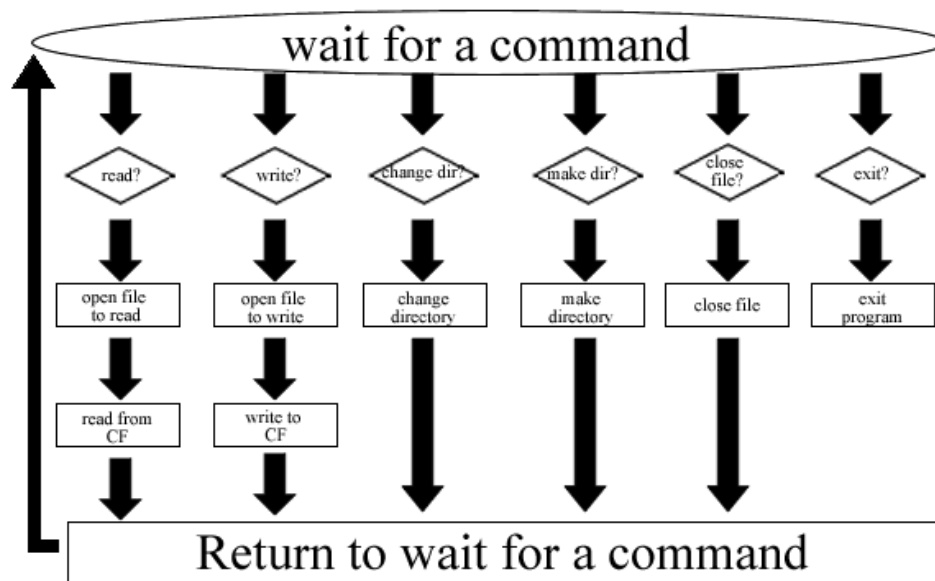


Fig. 2 – Algorithm

4. Experimental Results

The tests include two parts. The first test was the program's execution. This test was necessary to evaluate the system's performance. Then, it was done using the algorithm described above and setting hardware's feature. This hardware's feature is an IP that should be activated. It is the System ACE that is the responsible for the Compact Flash Controller.

The second test is an important step: to test the speed differences between files sent by serial RS 232 interface and files read by Compact Flash card. To make this test was utilized a PC connected by a serial cable in FPGA board. Then, was put a Compact Flash card containing files. After that, was developed a program in EDK. This program will count the time to send files by serial interface and to read files by Compact Flash card. To do this test were utilized two specifications: for the serial interface a serial connection with 57600 bits per second, without parity and flow control, eight data bits, and one stop bit. For the Compact Flash card was utilized all descriptions seen here. The PowerPC processor clock is set to 100MHZ. This structure is shown in fig 3. All tests were done with success.

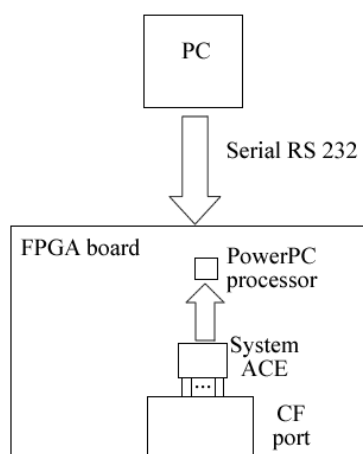


Fig. 3 – Serial interface and compact flash interface

The system developed had a good performance. The files were saved in the Compact Flash card and they were accessed in the card with a great performance improvement.

A comparative test between the serial interface and the Compact Flash Card was performed. There was a big difference between the two systems. Even at the highest speed (115200 bauds), the serial interface was significantly slower than the compact flash to download typical data sizes. Tab. 1 presents performance comparison between Serial Interface and Compact Flash interface. The speed gain in downloading data was 47 times.

Tab.1 – Comparison of time between Serial and CF interfaces for some data sizes.

Size of file (KB)	Serial Interface (ms)	Compact Flash Interface (ms)
1	72	1.5
10	722	15
20	1444	31
40	2888	61
100	7220	153

5. Conclusions

For embedded systems, an efficient way to store data is very important. Even in the proposed system were a host computer can be always connected, a significant speedup can be reached. This paper proves that a Compact Flash card can be a good resource to store reusable data in a FPGA based system. Combined with a good embedded software interface, the speed gain in the task of downloading data can be very good.

Future work will use the presented technique to make a dictionary based storage to compose the desired trajectory of the robot from trajectory segments stored in compact flash, reducing even further the need of trajectory download.

6. References

- [1] C. Diniz, “FPGA Implementation of a DC Motor controller with a Hardware/Software Approach,” *Proc. 22th South Symposium on Microelectronics* (SIM 2007), pp. 1-4.
- [2] T. Moreira, and D. Guimarães, “FPGA Implementation of AC Motor controller with a Hardware/Software codesign Approach,” *Proc. 14th Iberchip Workshop* (2008), pp. 1-4.
- [3] Xilinx Inc., “System ACE Compact Flash Solution”, Oct. 2008, http://www.xilinx.com/support/documentation/data_sheets/ds080.pdf.
- [4] Xilinx Inc., “OS and libraries document collection,” Sep. 2008, http://www.xilinx.com/support/documentation/sw_manuals/edk10_oslib_rm.pdf.

Design of Hardware/Software board for the Real Time Control of Robotic Systems

¹ Dino P. Cassel, ¹ Mariane M. Medeiros, ¹ Vitor I. Gervini, ² Sebastião C. P. Gomes, ¹ Vagner S. da Rosa

{dinokassel, mariane.mm, gervini, scpgomes, vrosa}@gmail.com

¹ Centro de Ciências Computacionais - FURG

² Instituto Matemática, Estatística e Física - FURG

Abstract

This paper presents an implementation of a system to control AC motors. The strategy was to design a single chip Virtex II pro based FPGA system with both hardware and software modules. The hardware/software partition was based on the need of some modules to conform timing requirements not achievable by software. The software includes motor control functions, memory management and communication routines, and was developed in C programming language. The hardware is described in VHDL and includes the Pulse-Width Modulation generation, incremental encoder pulse counting, and communication hardware. The Digilent XUP V2P development board was used for prototyping the complete system.

1. Introduction

The complexity in motor controllers has been increased recently, since new sophisticated algorithms are being used in motor control to obtain a better performance. The control law is commonly designed and implemented as software, in a high-level language. In spite of that, the real time restriction is crucial in motor control. This means that the controller should interrupt at a fixed rate (typically less than 1 ms) to capture current sensor signal and compute the torque value for driving the motor. A software implementation to decode sensor positions and generate the signal in order to drive the motor wastes time which could be used to compute the control law. This time lost can be higher if many motors have to be controlled with the same device. A hardware design for driving motor and capturing sensor signals could increase the controller performance.

FPGA can fit very well this application, due to the flexibility, cost and performance. Motor drive interfaces, such as PWM (Pulse-Width Modulation) and sensor interfaces (e.g. a logic for decode an encoder signal), can be easily developed and occupy low area [HAC 2005]. Some FPGA's contains a hardwired processor inside the chip in which is possible to run software and hardware simultaneously.

Then, a hardware/software co-design solution for a motor control is proposed in this work. Motor signal (PWM) and an interface with incremental rotary encoder (feedback logic) were first designed in VHDL and prototyped in a Xilinx FPGA. Besides, a graphical interface in C++ and a communication interface in Matlab were also developed.

The implemented system has some basic objectives: receive and interpret the data generated by user, implement a PID (Proportional-Integral-Derivative) Controller, calculate the speed, transform the final velocities into PWM signals, send these signals to the motors through the digital outputs available and also manage the memory system.

2. Hardware/Software approach

The strategy is to implement modules that require high update rate in hardware, to obtain better performance. For instance, to decode signal of an incremental encoder (for position measure) it's necessary at least 4 samples per pulse. With a 1024 PPR (Pulses per revolution) encoder, rotating at 2000 rpm, the sampling frequency should be at least 136 KHz. A PWM Generator operates at frequencies near of 20 KHz.

A complete system block diagram of the proposed system is presented in Fig. 1. This diagram shows the modules needed for DC motor control. All the modules in Fig. 1 were fitted into a single Virtex II FPGA, except for the DC motor (and its associated power bridge and incremental encoder) and the monitoring computer.

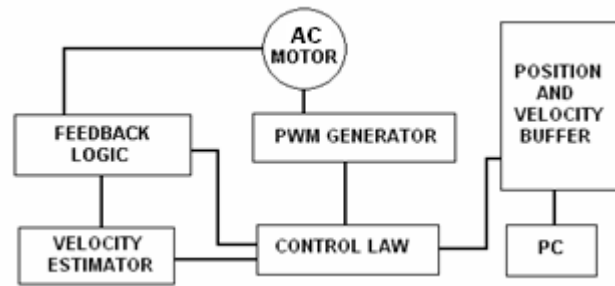


Fig. 1 – System block diagram.

A simple control law could be also development in hardware. In [SHA 2005] is proposed a motion control IC design in a dedicated hardware. But with the growing complexity of control law algorithms, if a complex control law algorithm must be tested, the time expended designing new VHDL logic is much higher than changing software routines. Due to the relatively slow response time of the control law algorithms (about some kHz), software implementation is usually more efficient, when a processor is available. The system architecture proposed provides flexibility to test new control algorithms maintaining a fast sample frequency on the interface modules.

3. Communication Computer/FPGA

The trajectory of reference that will run the control law is generated by the graphical interface installed on the computer. Once it was generated, it should be sent to the FPGA connected to this computer. This communication occurs through a serial port RS-232, the default port of FPGA, using a simple protocol of communication completely developed in Matlab.

The protocol sends a block of data to the FPGA and generates a checksum for the entire block. This checksum is calculated and verified in FPGA, which releases the receipt of the next block confirming the operation. The data on the FPGA is received as integer values once it was multiplied by a fixed value (specified by the needs of project, actually 100000) before had been sent by the interface. At the interface, it is divided again by the same value to be transformed into real numbers. Each trajectory of reference received by the FPGA, one for each motor, is placed in different parts of memory, defined in advance. This can be seen in Fig. 2.

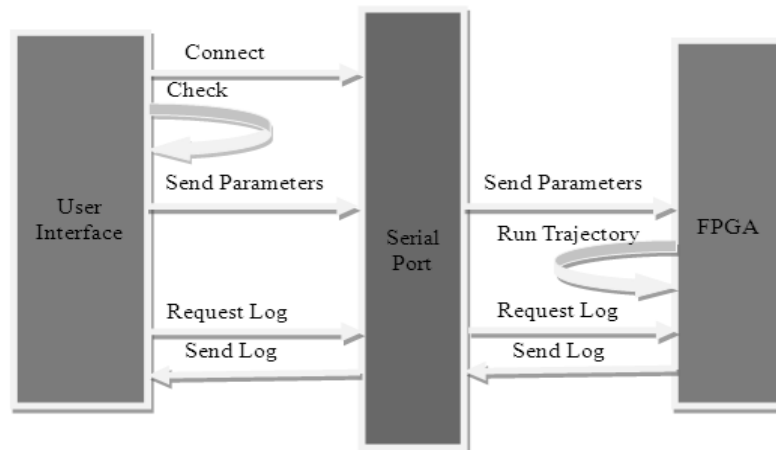


Fig. 2 – System communication diagram.

4. Control law

To control the AC motors of the platform was designed a type of proportional control for speed with correction of position using fuzzy logic, as shown in Eq. 1.

$$f_c = f_n + K_d(\dot{\theta}_r - \dot{\theta}) + \Delta F(\theta_r, \theta)$$

Eq. 1

The first part of the control law contains the desired frequency – f_n , the second part contains the gain generated by the error of the desired speed – $\theta'r$, and true speed – θ' . This second part is responsible for compensating the error in velocity resulting from the slip of motor. The third part of the control law – $\Delta f(.)$, is a fuzzy function that has as input parameters the real position – θ , and the desired position – θ_r . With these input parameters it returns a number of frequencies to be added in the frequency of control – f_c , to compensate the error in position.

There are some restrictions to implement the control of motors using the PowerPC processor available in the Virtex II pro FPGAs, as the lack of instructions for floating point arithmetic and reading of data from physical sensors with specific physical properties. To solve the first problem, 32-bit variables was used to represent fixed point values, where 16 bits are the fractional part and 16 are the integer part. The encoders capture the signal in units of pulse per revolution that was converted to the fixed point representation (in rad/s) by means of a simple integer multiplication.

The calculation of speed is obtained deriving the position read by the encoders. It is known that is possible to do all conversions possible to calculate the control law.

5. Memory management

The XUP V2P board has a memory DIMM DDR SDRAM up to 2GB capacity, which is directly connected to the bus Processor Local Bus (PLB). In this way, the PowerPC processor has a range of addresses to access the memory.

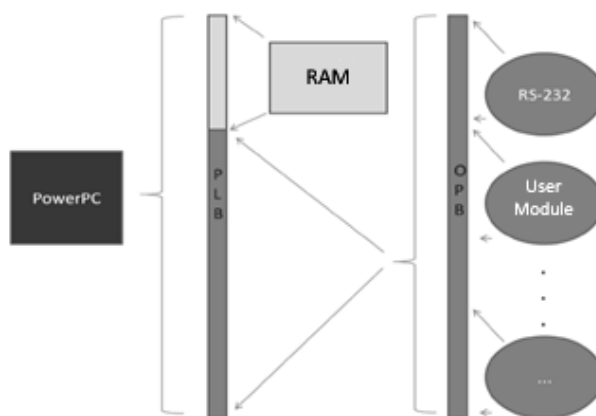


Fig. 3 – The processor and the devices in its area of addressing.

The PowerPC has two types of bus, the PLB and the On-chip Peripheral Bus (OPB). Devices of lower speed are placed in OPB and the highest speed in PLB. Devices that are in the OPB are mapped to the address space of the PLB through the module plb2opb, so the PowerPC has only a space for addressing.

To access the memory is necessary to instantiate a pointer of type integer, initialized with a value belonging to the address space of RAM. Thus, we can access the memory as one dimensional vector where each position can store data of 32 bits.

In this project, the memory was divided into equal parts, and each one stores a specific type of data (speed, position). The size of each part and other parameters, as the variables of status, are stored in the first stretch of the memory.

6. Results

The system was implemented successfully and is now in final phase of testing. It has proved to be extremely robust in what concerns communication with the computer and the FPGA and presents a very consistent memory management. Several security routines have also been implemented successfully. The control was also very efficient, with an error in position lesser of 1 mm.

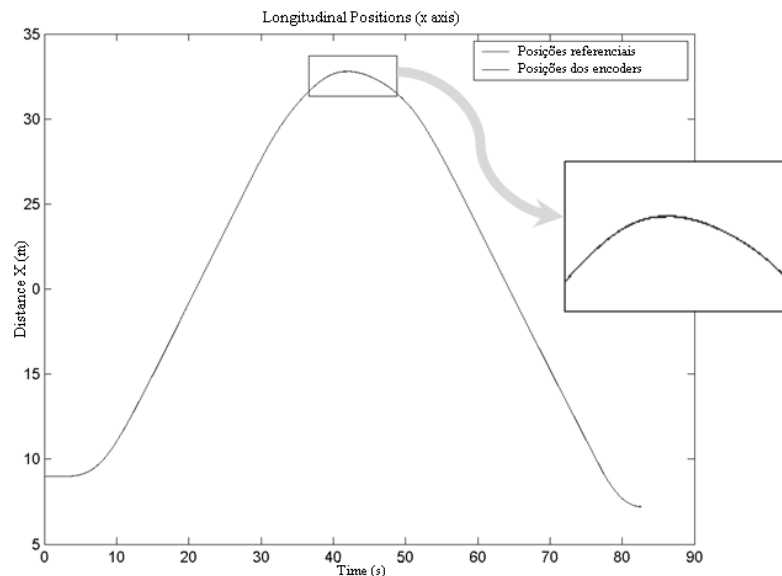


Fig. 4 – Control with less than 1mm error.

7. Conclusions

This work has presented the implementation of a hardware/software system to control robotic actuators. The project was very effective controlling robotic actuators and is now being studied to implement the same solution in the control of other structures, as flexible ones. The implementation of a FPGA device for the control of actuators was very efficient and easy to maintain, since it is much easier to make a change in a block of VHDL code than change the circuit board. This enables changes in both hardware and software as the specification is changed.

Future works includes the implementation of faster communication link to the host computer (such as Ethernet) and a local storage file system for the recording of trajectories in memory cards CompactFlash.

8. References

- [1] C. Hackney, "FPGA Motor Control Reference Design", Xilinx Application Note: Spartan and Virtex FPGA Families (XAP 808 v1.0), 2005.
- [2] X. Shao, and D. Sun, "A FPGA-based Motion Control IC Design", *Proceedings of IEEE International Conference on Industrial Technology (ICIT 2005)*, Hong Kong, December, 2005.
- [3] Inc. Digilent, "Virtex II Pro Development System", Mar. 2009; <http://www.digilentinc.com/Products/Detail.cfm?Nav1=Products&Nav2=Programmable&Prod=XUPV2P>.
- [4] Inc. Digilent, Mar. 2009; <http://www.digilentinc.com>.
- [5] Inc. Xilinx, "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Datasheet", Mar. 2009; <http://www.xilinx.com>.
- [6] Inc. Xilinx, "The Programmable Logic Company", Mar. 2009; <http://www.xilinx.com>.
- [7] Inc. Xilinx, "Xilinx ISE 8.2i software manual and help", Mar. 2009; http://www.xilinx.com/support/sw_manuals/xilinx82/index.htm.
- [8] Inc. Xilinx, "EDK 8.2i Documentation", Mar. 2009; http://www.xilinx.com/ise/embedded/edk_docs.htm.
- [9] D. Rosa, "Desenvolvimento de um Sistema de Hardware/Software para o Controle em Tempo Real de uma Plataforma de Manobras de Embarcações", graduation's thesis, FURG, Rio Grande, 2007.

HeMPS Station: an environment to evaluate distributed applications in NoC-based MPSoCs

Cezar R. W. Reinbrecht, Gerson Scartezzini, Thiago R. da Rosa, Fernando G. Moraes
{cezar.rwr, gersonscar, thiagoraupp00}@gmail.com, fernando.moraes@pucrs.br

PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul

Abstract

Multi-Processor Systems-on-Chip (MPSoCs) are increasingly popular in embedded systems. Due to their complexity and huge design space to explore for such systems, CAD tools and frameworks to customize MPSoCs are mandatory. The main goal of this paper is to present a conceptual platform for MPSoC development, named HeMPS Station. HeMPS Station is derived from the MPSoC HeMPS (Hermes Multiprocessor System). HeMPS, in its present state, includes the platform (NoC, processors, DMA, NI), embedded software (microkernel and applications) and a dedicated CAD tool to generate the required binaries and perform basic debugging. Experiments show the execution of a real application running in HeMPS.

1. Introduction

The increasing demand for parallel processing in embedded systems and the submicron technologies progress make technically feasible the integration of a complete system on chip, creating the concept of SoCs (System-on-chip). In a near future, SoCs will integrate dozens of processing elements, achieving a great computational power in a single chip. This computational power is commonly used by complex multimedia algorithms. Thus, systems based on a single processor are not suitable for this technological trend and the development of SoCs with more complexity will be multi processed, known as MPSoCs (Multi Processors System-on-chip).

Several MPSoC examples are available in academic and industrial areas. Examples of academic projects are HeMPS [CRIS07][CAR09], MPSoC-H [CAR05], [LIN05] and the HS-Scale MPSoC [SAI07]. Examples of industrial projects are Tile64 [TIL07] and [KIS06] from IBM.

The improvement of the MPSoC architectures makes possible the execution of high demands applications in a single chip. These applications exist in different areas, mainly in science, for example, genetic, organic chemical or even nuclear physics. Therefore, MPSoCs could solve them efficiently and with low cost. The cost issue refers that nowadays these applications are solved with efficiency just by super computers or clusters. In addition, the fact this technology has a great potential for parallel applications; it seems a better investment for industrial purposes.

MPSoCs are mostly used in embedded systems, usually without a user interface. However, designers should use dedicated environments to evaluate the performance of the distributed applications running in the target MPSoC. Such environments include: (i) dedicated CAD tools running in a given host; (ii) fast communication link between host and MPSoC to enable real time debugging and evaluation; (iii) monitoring schemes inserted into the MPSoC architecture enabling performance data (throughput, latency, energy) collection. The goal of this work is to present the proposal of an environment to help the development and evaluation of embedded applications targeting a NoC-based MPSoC. Such environment is named HeMPS Station.

2. HeMPS Station

HeMPS Station is a dedicated environment for MPSoC, basically a concept for a system structure to evaluate the performance of distributed applications in a given architecture running on an FPGA. This environment aims dedicated CAD tools running in a given host, fast communication link between host and MPSoC to enable evaluation during the execution and monitoring schemes inserted into the MPSoC architecture, enabling performance data collection. Therefore, HeMPS Station is composed by an MPSoC, an external memory and a Host/MPSoC Interface System, as shown in Fig. 1.

The MPSoC is a master-slave architecture and it contains three elements: (i) interconnection element; (ii) processing element; (iii) monitoring element. The interconnection element is a Network-on-Chip [BEN02] with mesh topology. The processing element comprises a RISC processor and private memory. This element also needs a network interface and a microkernel. The monitoring element can be inserted in the interconnection element, in the processing element or in both elements. The interconnection monitors are inside the routers or attached at the wire and their functions are to collect information of the network like throughput, latency or energy. On the other hand, the processing monitors are systems calls inserted in the tasks codes. With the systems calls, each processor informs a report about the task state.

The external memory is a repository attached to the master processor, storing all tasks of the application to be executed in the system. The master access this memory to allocate the tasks based on demand and availability of the system.

The Host/MPSoC Interface System manages three elements, the host, the master and the external memory. This system is responsible for a fast communication link between host and MPSoC, for loading code in the external memory, for the messages of debug and control with master and for the host's software that presents the performance data and translate the applications in code for the platform. The communication link is the bottleneck of the environment, so it needs to be practical and fast. Aiming these features the communication link uses Ethernet technology and Internet protocols. This make possible to reach high transmission rates since 10Mbps till 1000Mbps, and the internet protocols allows remote access.

The performance analysis is executed at the host, displaying to the user the performance parameters and the placement and state of each task during the execution. All collected data is stored in the host for further analysis. The main idea is to use the HeMPS Station through the internet, enabling the user to upload distributed applications into the MPSoC, and to receive after application(s) execution performance data.

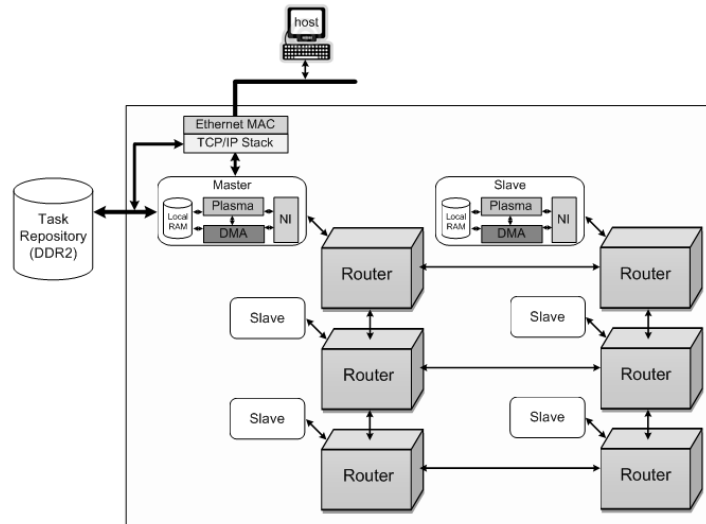


Fig. 1 – Hempt Station Environment

3. Present HeMPS Environment

The HeMPS architecture is a homogeneous NoC-based MPSoC platform. Fig. 2 presents a HeMPS instance using a 2x3 mesh NoC. The main hardware components are the HERMES NoC [MOR04] and the mostly-MIPS processor Plasma [PLA06]. A processing element, called Plasma-IP, wraps each Plasma and attaches it to the NoC. This IP also contains a private memory, a network interface and a DMA module.

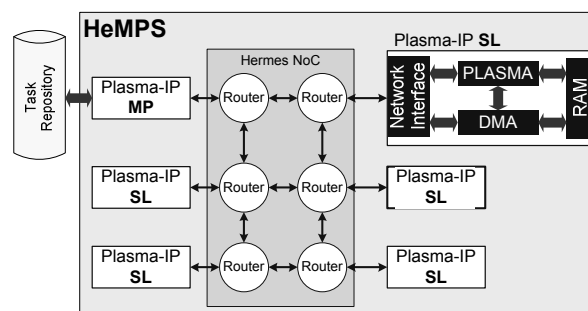


Fig. 2: HeMPS instance using a 2x3 mesh NoC.

Applications running in HeMPS are modeled using task graphs, and each application must have at least one *initial task*. The *task repository* is an external MPSoC memory, keeping all task codes necessary to the applications execution.

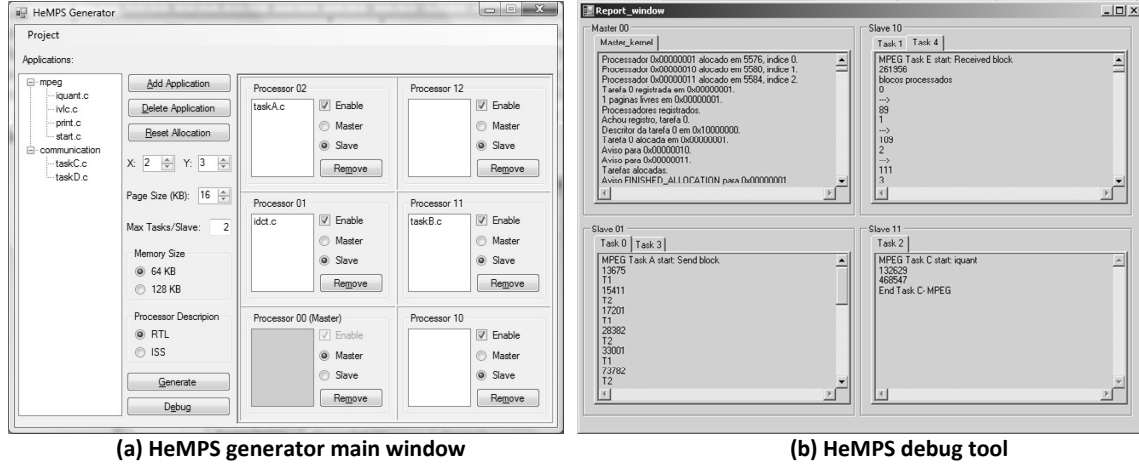
The system contains a *master processor* (Plasma-IP MP), responsible for managing system resources. This is the only processor having access to the task repository. When HeMPS starts execution, the master processor allocates initial tasks to the *slave processors* (Plasma-IP SL). During execution, each slave processor is enabled to ask to the master, on demand, dynamic allocation of the tasks from the task repository. Also, resources may

become available when a given task finishes execution. Such dynamic behavior enables smaller systems, since only those tasks effectively required are loaded into the system at any given moment.

Each slave processor runs a microkernel, which supports *multitasking* and *task communication*. The microkernel segments memory in pages, which it allocates for itself (first page) and tasks (subsequent pages). Each Plasma-IP has a *task table*, with the location of local and remote tasks. A simple preemptive scheduling, implemented as a *Round Robin Algorithm*, provides support to multitasking.

To achieve high performance in the processing elements, the Plasma-IP architecture targets the separation between communication and computation. The network interface and DMA modules are responsible for sending and receiving packets, while the Plasma processor performs task computation and wrapper management. The local RAM is a true dual port memory allowing simultaneous processor and DMA accesses, which avoids extra hardware for elements like *mutex* or cycle stealing techniques.

The initial environment to evaluate distributed applications running in HeMPS is named *HeMPS Generator* (Fig. 3).



(a) HeMPS generator main window

(b) HeMPS debug tool

Fig. 3: HeMPS environment.

The *HeMPS Generator* environment allows:

- 1) Platform Configuration: number of processor connected to the NoC, through parameters *X* and *Y*; the maximum number of simultaneous running tasks per slave; the page size; and the abstraction description level. User may choose among a processor ISS or VHDL description. Both are functionally equivalent, with ISS being faster and VHDL giving more detailed debug data.
- 2) Insertion of Applications into the System: The left panel of Fig. 3 (a) shows applications *mpeg* and *communication* inserted into the system, each one with a set of tasks.
- 3) Defining the initial task mapping: Note in Fig. 3 (a) *idtc* task (initial task of *mpeg*) allocated in processor 01, and tasks *taskA* and *taskB* (initial tasks of *communication*) allocated to processors 11 and 02 respectively.
- 4) Execution of the hardware-software integration: Through the *Generate* button, it fills the memories (microkernel, API and drivers) and the task repository with all task codes.
- 5) Debug the system: Through the *Debug* button, system evaluation takes place using a commercial RTL simulator, such as ModelSim. The *Debug HeMPS* button calls a graphic debug tool (Fig. 3 (b)). This tool contains one panel for each processor. Each panel has tabs, one for each task executing in the corresponding processor (in fig. 3 (b) processors 10 and 01 execute 2 tasks each one). In this way, messages are separated, allowing to the user to visualize the execution results for each task.

4. Experimental Setup and Results

This section presents an application executing in the present HeMPS platform. As a result, features like task allocations and execution time can be observed. The application used as benchmark is a partial MPEG codification. There are five tasks, named as A, B, C, D and E, with a sequential dependence. It means that task B only can initiate after task A request B. The HeMPS used for this experimental has size 2x2, where each slave processor has a memory size of 64KB and a page size of 16KB. So, they can allocate at maximum two tasks per slave processor (two pages are reserved for the kernel). It is important to mention that the master do not allocate for himself tasks. Its function is to dynamically allocate tasks and control the communication with the host.

Initially, the application code is written, using C programming language, and five files are created, each one represents a task. Then, using the HeMPS Generator, the HeMPS configuration is set and the codes of the tasks are opened and mapped. The mapping is depicted in the Fig. 4, and it shows that this experiment executes

dynamical allocation of the tasks. After that, the software generates all files for the simulation. After simulation, the software uses the debug tool to show results, shown in Tab. 2.

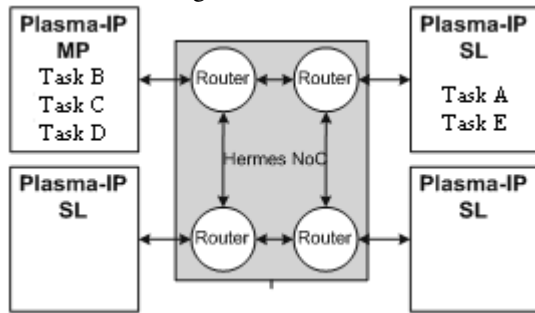


Fig. 4: Experiment Mapping

Tasks	Tick_Counter	
	Start	End
A	3.791	39.608
B	10.762	426.678
C	58.214	435.848
D	79.660	442.303
E	41.939	532.715

Tab. 2: Time, in clock cycles, of the start and end of MPEG application tasks.

The results show that tasks B/C/D are correctly allocated by the master. It is important to mention that dynamic task allocation overhead is minimal [CAR09], enabling the use of such strategy (dynamic task allocation) in larger MPSoCs.

5. Conclusion and Future Works

This paper presented the features of the HeMPS Station framework, composed by software executing in a given host, and an MPSoC executing in FPGA, both interconnect through an internet connection. The basis for the HeMPS Station framework is the HeMPS MPSoC, which is open source, available at the GAPH web site (www.inf.pucrs.br/~gaph). Presented results shown the efficiency of dynamic task allocation, a key feature adopted in the HeMPS MPSoC. Present work includes: (i) development of the IPs responsible to effectively implement the framework (Ethernet communication and memory controller); (ii) integration of such IPs to the HeMPS MPSoC; (iii) adapt the present software environment to communicate with the FPGA platform through the internet protocol. Future works include the addition of hardware monitors to collect data related to power, latency and throughput.

6. References

- [BEN02] Benini, L.; De Micheli, G. "Networks On Chip: A New SoC Paradigm". Computer, v. 35(1), 2002, pp. 70-78.
- [TIL07] Tiler Corporation. "TILE64™ Processor". Breve descrição do produto. Santa Clara, CA, EUA. Agosto 2007. 2p. Disponível em http://www.tiler.com/pdf/ProBrief_Tile64_Web.pdf
- [KIS06] Kistler, M.; Perrone, M.; Petrini, F. "Cell Multiprocessor Communication Network: Built for Speed". IEEE Micro, Vol 26(3), 2006. pp. 10-23.
- [CRIS07] Woszezenki R. C.; "Alocação De Tarefas E Comunicação Entre Tarefas Em MPSoCS", Dissertação de Mestrado. Março 2007.
- [CAR05] Carara, E.; Moraes, F. "MPSoC-H – Implementação e Avaliação de Sistema MPSoC Utilizando a Rede Hermes". Technical Report Series. FACIN, PUCRS. Dezembro 2005, 43 p.
- [CAR09] Carara, Everton Alceu, Oliveira, R. P., Calazans, N. L. V., Moraes, F. G. "HeMPS - A Framework for Noc-Based MPSoC Generation". In: ISCAS, 2009.
- [LIN05] Lin, L.; Wang, C.; Huang, P.; Chou, C.; Jou, J. "Communication-driven task binding for multiprocessor with latency insensitive network-on-chip". In: ASP-DAC, 2005. pp. 39-44.
- [MOR04] Moraes, F.; Calazans, N.; Mello, A.; Mello, A.; Moller, L.; Ost, L. "HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip". Integration the VLSI Journal, Amsterdam, v. 38(1), 2004, p. 69-93.
- [PLA06] PLASMA Processor. Captured on July 2006 at <http://www.opencores.org/?do=project&who=mips>.
- [SAI07] Saint-Jean, N.; Sassatelli, G.; Benoit, P.; Torres, L.; Robert, M. "HS-Scale: a Hardware-Software Scalable MP-SOC Architecture for embedded Systems". In: ISVLSI, 2007, pp 21-28.

Analysis of the Cost of Implementation Techniques for QoS on a Network-on-Chip

^{1,2}Marcelo Daniel Berejuck, ²Cesar Albenes Zeferino
marcelo.berejuck@unisul.br, zeferino@univali.br

¹Universidade do Sul de Santa Catarina

²Universidade do Vale do Itajaí

Abstract

The architecture of network-on-chip called SoCIN (System-on-Chip, Interconnection Network) can be tailored to meet the requirements of cost and performance of the target application, offering a delivery service of type best effort. The objective of this work was to deploy three different mechanisms for providing quality services to the network SoCIN, assessing the impact of these mechanisms in the cost of implementation in programmable logic devices, FPGA type.

1. Introduction

The increasing density on integrated circuits has allowed designers to implement multiple processors of different types in a single chip, also including memories, peripherals and other circuits. They are complete systems integrated on a single chip of silicon, usually known as Systems-on-a-Chip (SoCs). The interconnection of components in future SoCs will require reusable communication architectures with scalable performance, feature not found in the architectures used in the currently SoCs [1].

A consensus solution for this problem is the use integrated switching network, largely known as Networks-on-Chip – NoCs [1]. Among the main advantages of this approach, it can be highlighted the fact that these networks provide parallelism in communication, scalable bandwidth and modular design [2], [3].

An example of NoC is the network called SoCIN (System-on-Chip Interconnection Network), presented in [3]. This network was originally conceived as a packet switched NoC with best effort (BE) services for the treatment of communication flows, ensuring the delivery of packets, but without any guarantees of bandwidth or latency. In this type of service, all communication flows are treated equally and the packets transferred over the network may suffer an arbitrary delay. The term Quality of Service (QoS) refers to the ability of a network to distinguish different data flows and provide different levels of service for these flows. Consequently, BE services are inadequate to meet the QoS requirements of certain applications, such as, for example, the multimedia streams.

This paper presents the results of the implementation of three techniques for QoS in SoCIN network, by making changes in the design of its router, named ParIS (Parameterizable Interconnect Switch) [4]. These three techniques are recognized in the literature as mechanisms for providing some level of QoS. The following text is organized into four sections. Section 2 presents some related works regarding QoS in NoCs. Section 3 presents an overview about the original architectures of SoCIN network and ParIS router. Following, Section 4 describes the implementation of three techniques to provide QoS in SoCIN: circuit switching, virtual channels and aging. Afterwards, Section 4 presents an analysis about the silicon overhead of the implemented techniques. Finally, in Section 5, they are presented the conclusions.

2. Related works

According Benini and De Micheli [1], “quality of services is a common networking term that applies to the specification of network services that required and/or provided for certain traffic classes (*i.e.* traffic offered to the NoC by modules)”.

Each designer can identify classes of services for a particular implementation of NoC. Several studies have noted the need to classify traffic and differentiate the service classes according to pre-established services. For example, *Æthereal* network [5] separates services into two distinct classes: Guaranteed Throughput (GT) and Best Effort (BE). In QNoC architecture [6], its designers defined four service classes: Signaling, Real-Time, Read/Write and Block Transfer. In both architectures, a virtual channel is allocated for each class, allowing providing differentiated services for the different classes. In *Hermes* network, several implementations were done in order to add QoS supporting to the NoC [6][7], including circuit switching (*Hermes_CS*) and virtual channels (*Hermes_VC*). The circuit switching technique is also used in *Æthereal* network. However, in this case, time slots are dynamically allocated to the flows, which share the channels bandwidth by means of a time-shared multiplexing.

In a different approach, Correa et al. [9] applied a combination of aging technique with disposal of packages in order to prioritize those packets with requirements for time delivery in a NoC. The goal was to reduce the missed deadlines rate for soft real-time flows. In the proposed approach, the oldest packets in the network gets more opportunities when competing with packets injected more recently, reducing the latency to reach its destination. For soft-real time packets, when a target deadline is missed, the packet is discarded. This proposal was applied to SoCIN network and was validated and evaluated by using a C-based simulator. However, there was no synthesis in silicon to assess the feasibility of physical implementation of these techniques, in particular the aging one. The analysis was limited to assessing the ability of reducing losses of deadlines.

3. Overview on SoCIN network

SoCIN network was developed at the Federal University of Rio Grande do Sul [3]. It uses 2-D mesh topology and its links are implemented as two unidirectional channels in opposition. Each channel consists of n bits of data and two sideband bits used as markers of the beginning and the ending of a packet (BOP: Begin-of-Packet; and EOP: End-of-Packet). Therefore, the *phit* (physical unit) is $n+2$ wide.

All packets are composed by *flits* (flow control unit), and each *flit* has the size of a *phit*. The first *flit* is the packet header and includes the information necessary to establish a path along the network. The other *flits* are the payload and contain the information to be transferred. They follow the header in a pipeline way and the last *flit* of the payload is the package trailer (*i.e.* the tail that ends the packet).

Fig. 1 shows the header of SoCIN packet. It includes the framing bits (EOP and BOP) and two fields: RIB and HLP. The RIB (Routing Information Bits) includes the address of the source router (Xsrc, Ysrc) and the destination router (Xdest, Ydest) of the packet. The HLP (Higher Level Protocol) is composed of the remaining bits of the header that are not handled by the routers. They are reserved to be used by the communication adapters to implement services at higher protocol levels (*e.g.* services of the transport layer).

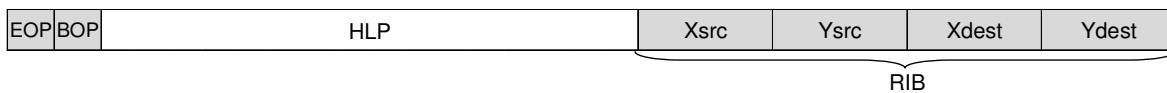


Fig. 1 – SoCIN packet header.

The current building block of SoCIN network is the ParIS router, developed at the University of Vale do Itajaí [4]. It is a packet switched router and has a distributed organization, as illustrated in fig. 2. All the input channels (Lin, Nin, Ein, Sin and Win) are handled by instances of the input module (Xin). This module is responsible for buffering a received packet, routing the packet and sending a request to a selected output module (Xout). Since multiple requests can be simultaneously assigned to the same output channel, the output module uses an arbitration mechanism to schedule the attendance of such requests. When a request is granted, a connection is established between the selected Xin module and the Xout module. This connection is made through data and control crossbars, which are implemented as multiplexers in the Xin and Xout modules.

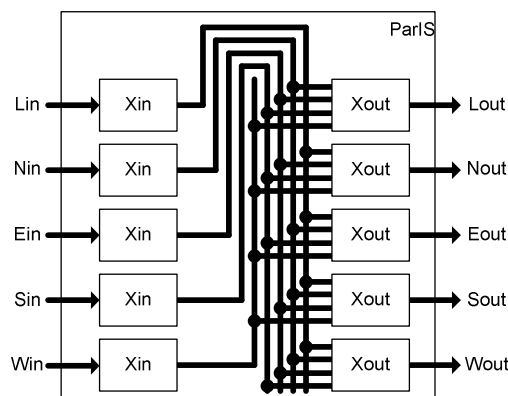


Fig. 2 – Internal organization of ParIS router.

4. Adding QoS support on SoCIN network

In ParIS router, the only way to try to meet bandwidth or latency requirements is by oversizing the network, increasing the operating frequency or adding more wires to the data channels. In this section, three implementations are presented to support some level of QoS in ParIS router: circuit switching, virtual channels and aging.

4.1. Circuit switching

In SoCIN network, when there is no contention, each router adds 3 cycles to the packet latency. If there is contention, the packet latency increases, adding jitter to the flow. To minimize this problem, it was proposed the implementation of the circuit switching technique mapped onto the packet switching. Control packets are used to allocate and release circuits, which are used to transfer data packets with minimal latency (one cycle per router) without the risk of contention. A new 2-bit field, named *CMD*, was added to the packet header in order to identify the packet type, as is shown in tab. 1.

Tab.1 – Description of the bits of the field *CMD*.

CMD	Command	Packet type
00	Normal	Data packet
01	Aloc	Packet for circuit allocation
10	Release	Packet for circuit release
11	Grant	Packet granting circuit allocation

To support the circuit switching, minimal changes were made in the internal architecture of Xin. Within this module, there is a block, called *REQ_REG* (Request Register), which records a request for the output channel selected by the routing algorithm. In the packet switching technique, the request is registered when the header is received and remains active until the packet trailer is delivered, releasing the connection. With the changes made in *REQ_REG*, when an *Aloc* command is received and the required connection is set, the only packet that can reset this connection is the one with the *Release* command. The *Grant* command is reserved for the destination node to notify the source node that the circuit was established. The *Normal* command identifies regular data packets.

4.2. Virtual Channels

The original SoCIN network does not implement any technique to provide differentiated services. As a workaround to this limitation, they were established two general classes: Real-Time (RT) and non-Real-Time (nRT), and in each class were defined two levels of subclass, resulting, effectively, into four classes: RT0, RT1, nRT0 and nRT1. In the packet header, a 2-bit field (called *CLS*) was added to identify the class.

In the ParIS₂ router they were implemented two virtual channels, one for RT flows and another for nRT flows. The implementation of the virtual channels was based on the replication of Xin and Xout modules. As Fig. 3 shows, the *Xin_{2VC}* module is composed by two Xin modules and one selector, while the *Xout_{2VC}* module is composed by two Xout modules and one selector (the new blocks added to ParIS₂ are filled in gray). When a packet is received by the *Xin_{2VC}* module, it is forwarded to the respective Xin module. When there are packets in both *Xout_{RT}* and *Xout_{nRT}* modules, the first one will always preempts the second one.

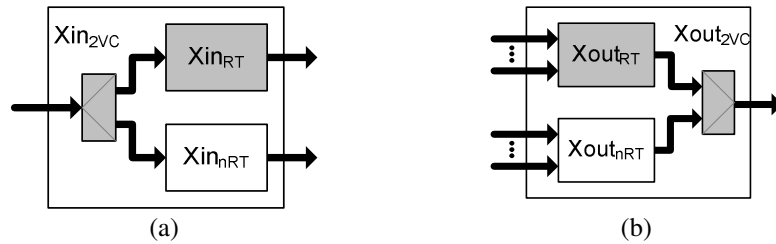


Fig. 3 – New modules with two virtual channels: (a) *Xin_{2VC}*; and (b) *Xout_{2VC}*.

The major difference between these modules and the ones of the original ParIS router relies on the arbiter included in the *Xout_{RT}* and *Xout_{nRT}*. The new arbiter has to give more priority to Class 0 packets when competing with Class 1 packets (*e.g.* priority of RT0 packets is greater than the one of RT1 packets). Therefore, it includes an additional arbitration level to the original arbiter.

4.3. Aging

The differentiation of services based on classes can give higher priority to a given class. However, there is no way to offer a different treatment for packets of a same class based on any criteria. As an alternative to this limitation, it was proposed the use of the aging technique, first applied in [9]. In this technique, the time that the package remains in the network is taken into account at the moment of the arbitration, and the oldest packets receive a higher priority than the newest ones of the same class. Such approach helps to reduce latency and the missed deadlines rate.

To implement this technique, a new 3-bit field was added to the header. This field (named *AGE*) is processed only by the Xin and Xout modules of the RT channels. In the Xin module, the *AGE* field is updated

every certain number of clock cycles when the packet header remains waiting for a grant at the input buffer of the Xin_{RT} module, getting older. In the $Xout$ module, at the time of the arbitration, when two or more RT packets of the same class (RT0 or RT1) compete for the channel, the one with bigger AGE will be selected.

To accomplish with such functionalities, a *stampler* circuit was added to the Xin_{RT} module in order to update the AGE field of the header while it is not forwarded through the selected output channel. Also, in the $Xout_{RT}$ module, it was necessary to add a new block to the arbiter in order to prioritize the oldest packet.

5. Results

The proposed techniques were described in VHDL (the HDL language of ParIS soft-core) and synthesized by using the Xilinx® ISE® tools. For the circuit switching, the silicon overhead was of less than 3% in LUTs and FFs because the only change was made in the REQ_REG block. For the virtual channels, the silicon overhead was about 115% compared to the original version of the ParIS router, which was expected, since the implementation of virtual channels was based on the replication of modules and on the inclusion of selectors. Finally, the overhead of implementing the aging technique was of about 24% in flip-flops and 31% in look-up tables in comparison with the router with two virtual channels. This overhead is due to the *stampler* circuit added to the Xin_{RT} module and to the changes made in the arbiter of the $Xout_{RT}$ modules.

All implementations were evaluated and validated by simulation, which allowed verifying the correctness of the VHDL codes.

6. Conclusion

In this paper, they were presented the implementation of three techniques for providing QoS in SoCIN network, which were modeled in VHDL and synthesized in FPGA. The circuit switching added a negligible overhead, but the other techniques resulted in large overheads, because the virtual channels were based on the replication of resources. It is intended to achieve architectural optimizations to reduce this overhead by sharing some control blocks by the virtual channels, but this can implies in a bigger latency. As future work, these implementations will be added to the SystemC model of ParIS router for assessing the impact of these techniques in meeting QoS requirements in SoCIN-based systems by means of SystemC simulation.

7. References

- [1] BENINI, Luca; DE MICHELI, Giovanni. "Networks on Chip." 1. ed. [S.l.]: Morgan Kaufmann, 2006.
- [2] DALLY, William. J.; TOWLES, Brian. "Principles and Practices of Interconnection Networks." 1 ed. [S.l.]: Morgan Kaufmann, 2004.
- [3] ZEFERINO, C. A.; SUSIN, A. "SoCIN: a parametric and scalable network-on-chip." In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS, 16, 2003, São Paulo. Proceedings... Los Alamitos: IEEE Computer Society, 2003. p. 169-174.
- [4] ZEFERINO, Cesar A.; SANTO, Frederico G. M. E.; SUSIN, Altamiro A. "ParIS: a parameterizable interconnect switch for networks-on-chip." In: 17th symposium on Integrated circuits and system design, SBCCI, 2004. Proceedings... Sept, 2004.
- [5] DIELISSEN, John; GOOSSENS, Kees; RADULESCU, Andrei; RIJPKEMA, Edwin. "Concepts and Implementation of the Philips Network-on-Chip." In: IP-SOC, 2003, Grenoble.
- [6] BOLOTIN, E. et al. "QNoC: QoS architecture and design process for network on chip." Journal of Systems Architecture, 49, 2003. Special issue on Networks on Chip.
- [7] MELLO, A. V. "Qualidade de Serviço em Rede Intra-chip. Implementação e avaliação sobre a Rede Hermes." 2006. 129f. Dissertação (mestrado) – Pontifícia Universidade Católica do Rio Grande do Sul. Programa de Pós Graduação em Computação, Porto Alegre, 2006. (in portuguese)
- [8] MELLO, A.; TEDESCO, L.; CALAZANS, N.; MORAES, F. "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC". In: 18th SBCCI, 2005, pp. 178-183.
- [9] CORREA, E. F., SILVA, L. A. P., CARRO, L., WAGNER, F. R. "Fitting the Router Characteristics in NoCs to Meet QoS Requirements." In: Proceedings of 20th Symposium on Integrated Circuits and Systems Design, 2007. p. 105-110.

Performance Evaluation of a Network-on-Chip by using a SystemC-based Simulator

Magnos Roberto Pizzoni, Cesar Albenes Zeferino
{magnospizzoni, zeferino}@univali.br

Universidade do Vale do Itajaí – UNIVALI
Grupo de Sistemas Embarcados e Distribuídos – GSED

Abstract

In order to explore the design space of Networks-on-Chip and evaluate the impact of different network configurations on the system performance, designers use simulators based on one or more abstraction levels. In this sense, this work applies a SystemC RTL/TL simulator in order to evaluate the performance of a Network-on-Chip with the goal of obtain a better understanding about this issue.

1. Introduction

Network-on-Chip – NoCs are a consensus solution for the problem of providing a scalable and reusable communication infra-structure for Systems-on-Chip – SoCs integrating from several dozens to hundreds of processing cores [1][2].

The design space of NoCs is large, and the designer can configure several parameters in order to try to meet the application requirements, including, the strategies used for flow control, routing, switching, buffering and arbitration, and others, like the sizing of channels and FIFOs. In order to tune the best configuration for a target application, designers use simulators which help to evaluate how a NoC configuration behaves for a given traffic pattern, meeting (or not) the application requirements.

In this context, a NoC simulator was developed at UNIVALI (Universidade do Vale do Itajaí) allowing designers to make performance evaluation experiments in order to characterize the performance of the Network-on-Chip (NoC) named SoCIN (System-on-Chip Interconnection Network) [3] for different network parameters and traffic patterns. This simulator, named X Gsim [4], is based on mixed Register-Transfer Level/Transaction Level (RTL/TL) SystemC-based models and offers a graphical interface that automates the design of experiments and the analysis of results [5].

In this paper, we present a set of experiments performed over X Gsim in order to characterize performance metrics of SoCIN and obtain a deeper knowledge about the impact of the network parameters in its performance. The text is organized in four sections. Section 2 describes the X Gsim simulator, while Section 3 presents the parameters used in the performed experiments. Section 4 discusses the obtained results, and Section 5 presents the final remarks.

2. The X Gsim simulator

The X Gsim Simulator [4][5] was developed by the research Group on Embedded and Distributed Systems (GSED) of UNIVALI. It aims at providing a computational infrastructure for performance evaluation of SoCIN NoC. It is based on SystemC models for the network components: routers (modeled at RTL), traffic generators and traffic meters (modeled as mixed RTL/TL descriptions). The TL approach was used offer flexibility and easiness to the descriptions of the tasks related to traffic generation and traffic measurement for analysis. The X Gsim simulator has a graphical interface implemented by using C language and GTK+ library, and offers resources for graphical analysis supported by GnuPlot and GTKWave.

The X Gsim simulator allows configuring the network parameters for one or more experiments, including: the network size, the channels width, the techniques used for flow control, routing and arbitration, and the FIFOs depth. Also, it allows configuring the traffic pattern to be applied by using the traffic model proposed in [6], in which the communications flows can be defined for different spatial distributions (*e.g.* uniform or complement) and injection rates (*e.g.* constant-bit rate or variable-bit rate). For a given traffic pattern, one can specify the number of packets to be sent by each communication flow, the size of each packet and the interval between two packets. The simulator also allows the configuration of a sequence of experiments where the injection rate is automatically varied in a given range (*e.g.* from 10% to 70% of the channel bandwidth).

3. Using X Gsim to evaluate the performance of SoCIN

A dozen of experiments were performed by using X Gsim. In this paper we describe and discuss some issues related with three of these experiments, by analyzing the behavior of the NoC for different settings related to the number of packages, the size of packets and the routing algorithm.

All the experiments were based on a 4×4 2-D mesh with 32-bit channels. The traffic pattern was configured to use a uniform spatial distribution based on a constant-bit rate injection. In this pattern, each node sends one communication flow for each one of the other nodes in the network. The injection rate was varied from 10% to 90% of the channel bandwidth.

In all the experiments, the following common configuration of parameters was used: wormhole switching; credit-based flow control; round-robin arbitration; and 4-flit FIFO buffers at the input and output channels. In the different experiments, they were varied the number of packets per flow (10, 100 and 200 packets) and the routing algorithm (XY and WF).

4. Results

4.1. Analysis of the simulation time

In this first set of experiments, we varied the number of packets generated by each flow (10, 100 and 200) and measured the time spent to perform nine experiments per configuration, by varying the injection rate from 10% to 90% of the channels bandwidth, with an increment of 10%. As Fig. 1 shows, the simulation time to run 9 experiments (in seconds) increases linearly with the number of packets to be transferred.

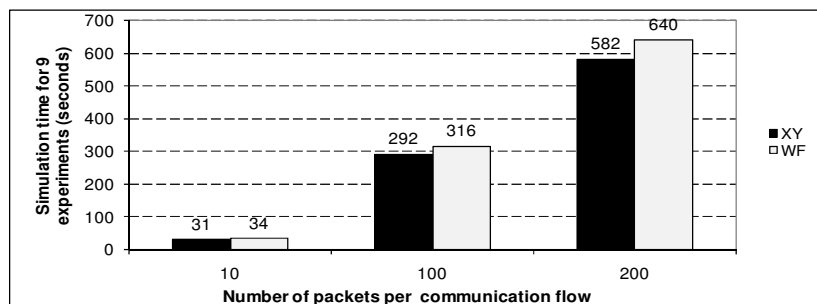


Fig. 1 – Simulation time for different communication flow lengths (4 flits per packet payload)

By analyzing Fig. 1, one can consider that is better to simulate traffic patterns with less packets (ex. 10), but as Fig. 2 shows, results for such configuration are quite different from the one obtained in other experiments when one analyzes the saturation threshold. In fact, for better results, the simulated time has to be as long as possible.

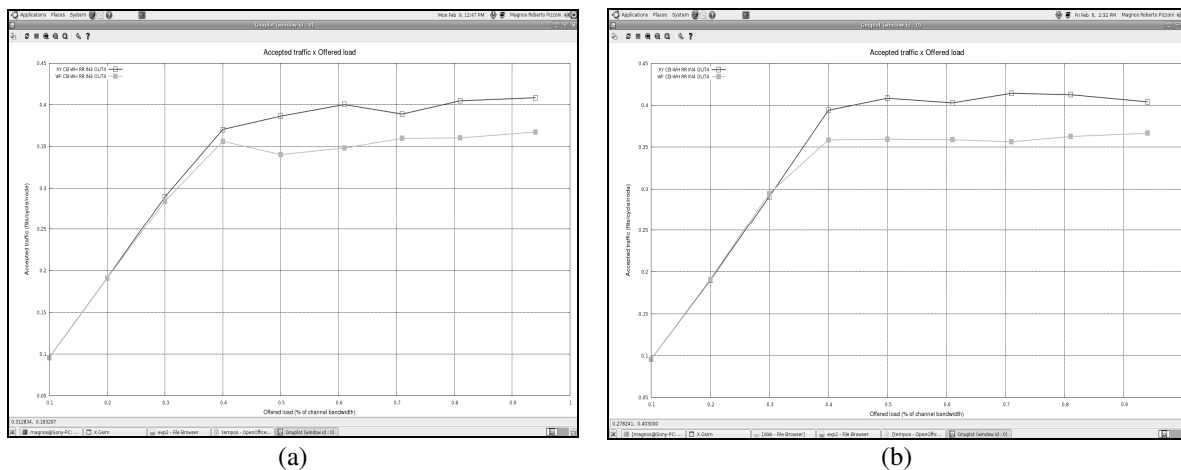


Fig. 2 – Accepted traffic × offered load for XY and WF routing for: (a) 10 packets/flow; (b) 100 packets/flow.

4.2. Analysis of the accepted traffic

Fig. 3 presents the curves for the “Accepted Traffic × Offered Load” for experiments performed for XY routing (the curve with empty bullets) and WF routing (the curve with filled bullets) considering 100 packets per flow. The offered load is expressed as a normalized fraction of the channel bandwidth. For instance, for credit-based flow control, an offered load of 0.5 (50%) means that the nodes are trying to inject one flit at each 2 cycles on the network channels. The accepted traffic expresses how many flits per cycle the network is able to deliver to each node. As one can observe in Fig. 3, for both configurations, the network accepts almost all the load offered by the nodes when the offered load is smaller than 0.4 (*i.e.* 40%). When the offered load equals

0.4, both configurations are already saturated and even if the nodes inject more communication load, the network will be not able to accept more traffic. However, it is important to note that the XY routing accepts more traffic (about 40%) than the WF routing (about 36%). This means that this routing algorithm supports application with more bandwidth requirement than the WF.

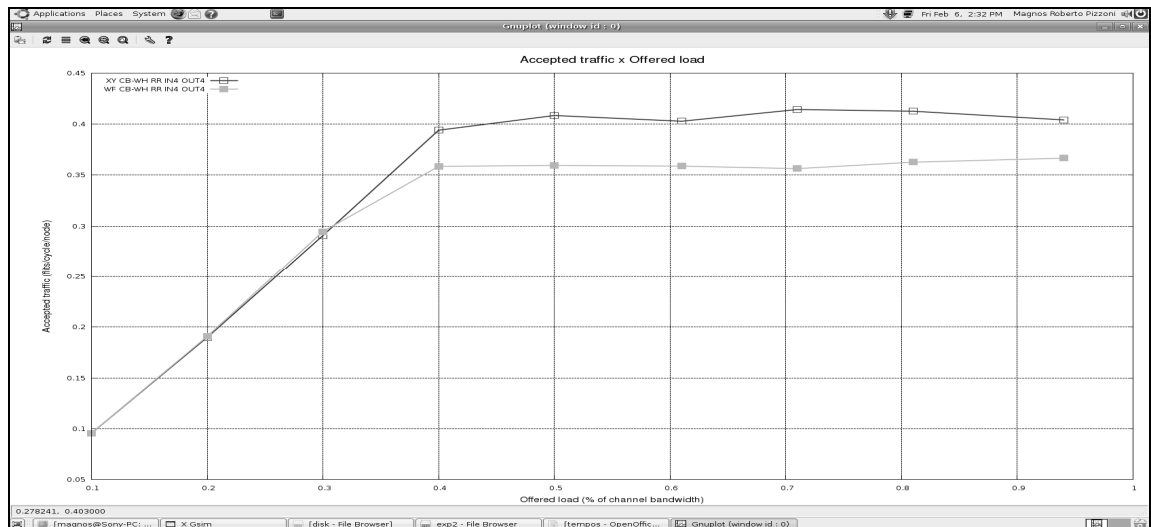


Fig. 3 – Accepted Traffic $\times$ Offered Load for XY and WF routing for:
(a) 10 packets/flow; (b) 100 packets/flow.

4.3. Analysis of the average latency

Fig 4. presents the curves for the “Average Latency $\times$ Offered Load” for the same experiments. The latency expresses the number of cycles spent to deliver a packet, since its creation until its last flit is ejected from the network. The latency takes into account the time that packets stay in the buffers and the time for routing and arbitration. It is a good metric to analyze the packet contention in the NoC [6]. As Fig. 4 shows, below the saturation point (offered load = 0.4), the average latency remains stable at a given value. At the saturation point (and above) the average latency increases almost exponentially and reaches values that are not acceptable by a real application.

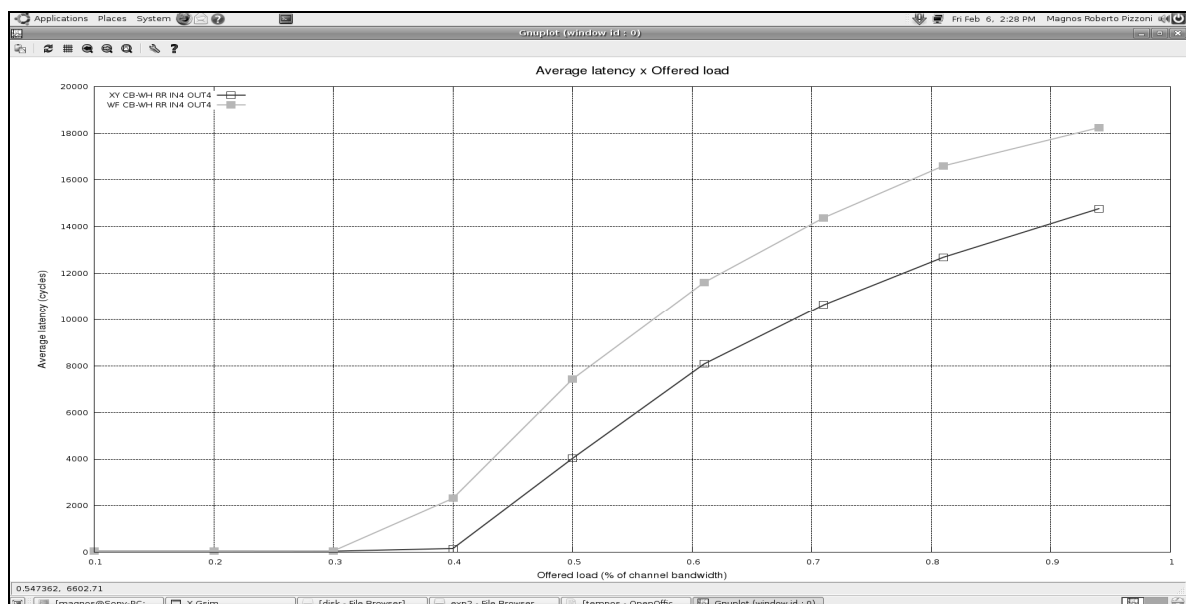


Fig. 4 – Accepted Traffic $\times$ Offered Load for XY and WF routing for:
(a) 10 packets/flow; (b) 100 packets/flow.

5. Final remarks

This paper presented the results of a study on the performance analysis of a Network-on-Chip based on simulation. The study was supported by a SystemC-based simulator integrating a set of tools which automates the design of experiments and the analysis of results. The performed experiments were useful to obtain a deeper knowledge about the performance evaluation of NoCs.

6. References

- [1] A. Hemani, et al. "Network on a Chip: An Architecture for Billion Transistor Era", *Proceedings of the IEEE NorChip Conference*, 2000, pp. 166-173.
- [2] L. Benini, G. De Micheli, "Networks on Chips: a New SoC Paradigm", *Computer*, v.35, n.1, 2002.
- [3] C. A. Zeferino, A. A. Susin, "SoCIN: A Parametric and Scalable Network-on-Chip", *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2003. p. 169-174.
- [4] C. A. Zeferino, J. V. Bruch, T. F. Pereira, M. E. Kreutz, A. A. Susin, "Avaliação de desempenho de Rede-em-Chip modelada em SystemC", *Anais do Congresso da SBC: Workshop de Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*, 2007. p. 559-578.
- [5] J. V. Bruch, R. L. Cancian, C. A. Zeferino, "A SystemC-based environment for performance evaluation of Networks-on-Chip", *Proceedings of the South Symposium on Microelectronics*, 2008. p. 41-44.
- [6] L. P. Tedesco, "Uma Proposta Para Geração de Tráfego e Avaliação de Desempenho Para Nocs," *Dissertação de Mestrado, PUCRS-PPGI, Porto Alegre, RS*, 2005.

Traffic Generator Core for Network-on-Chip Performance Evaluation

¹Miklécio Costa, ²Ivan Silva
miklecio.costa@inf.ufrgs.br, ivan@dimap.ufrn.br

¹PPGC – Instituto de Informática – UFRGS
²Departamento de Informática e Matemática Aplicada – UFRN

Abstract

The increase of transistor integration on a chip capability has allowed the development of MP-SoCs (Multi-Processor System-on-Chip) to solve the problem of the growing complexity on applications. Based on that, the concept of NoC (Network-on-Chip) came up, being considered more adequate to the purposes of MP-SoC. Many NoC projects have been developed in the last years, what has created the need of evaluating the performance of these mechanisms. This paper presents systems based on traffic generator cores for NoC performance evaluation. The systems are based on the measures latency and throughput, calculated by cores coupled on NoC routers. The cores were implemented in the hardware description language VHDL, integrated with the NoC SoCIN and with some Altera's components, like Avalon bus and Nios II/e processor, synthesized and prototyped on a FPGA board. Applications implemented in the programming language C configured the cores in order to obey some traffic patterns. These applications collected latency and throughput results measured by the cores on the FPGA and generated SoCIN evaluations. The results showed that these systems compose a consistent and flexible methodology for Network-on-Chip performance evaluation.

1. Introduction

The increase of transistor integration on a chip capability has allowed the development of MP-SoCs (Multi-Processor System-on-Chip). MP-SoCs have been used to solve the problem of the growing complexity on applications, because the traditional mono-processor systems present a theoretical physical limit in their processing capability [1].

Based on that, the concept of NoC (Network-on-Chip) came up, a mechanism of module interconnection on MP-SoC. NoC is considered more adequate to the purposes of a MP-SoC when compared to mechanisms like dedicated point-to-point hires, shared bus or bus hierarchy, because of characteristics like high scalability and reusability [2].

Many NoC projects have been developed, trying to attend this growing demand. These projects differ from each other because of the large number of possible NoC implementations. For that reason, some performance measures are used to evaluate the quality of those NoC projects in order to decide which one attends the requisites of a MP-SoC. The most important measures to evaluate an interconnection network, like a NoC, are latency and throughput [3].

This paper proposes two systems based on traffic generator cores for Network-on-Chip performance evaluation: one uses the measure latency and the other uses the throughput. In both systems, the cores are coupled on NoC routers and communicate with a software system, composed by one Nios II/e processor and one memory. The developed methodology includes the system synthesis and prototyping on a FPGA board.

This paper is divided in the following sections: section 2 presents a general view about NoC; section 3 presents some concepts involved in NoC performance evaluation that are used in this project; section 4 describes both systems proposed in this paper; section 5 presents the results obtained by the use of these systems on the NoC SoCIN; and finally, section 6 presents the conclusion.

2. Network-on-Chip

A Network-on-Chip is formed by a set of routers and point-to-point links that interconnect the cores of an integrated system [4], as illustrated in fig. 1. It's a concept inherited from the traditional computer interconnection networks.

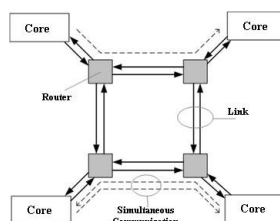


Fig. 1 – A Network-on-Chip model

The messages transmitted among the system cores are divided into packets. The packets may yet be divided into flits (flow control unit). The routers are responsible for delivering the packets of core communication, forwarding them to each router toward the destination.

A NoC project is defined by how it implements a set of characteristics: topology, routing, arbitration, switching, memorization and flow control. This diversity of characteristics has allowed the development of several NoC designs, what intensifies the relevance of evaluating such projects performance.

3. NoC Performance Evaluation

In the recent years, some works have been developed to evaluate NoCs performance [5], [6]. They combine bit level accuracy and flexibility, implementing on FPGA (Field Programmable Gate Arrays) many kinds of NoCs, traffics and performance measures.

The performance evaluation of an interconnection network, such as a NoC, is based on two main measures: latency and throughput [3]. In order to collect these values, the literature describes some traffic patterns which define pairs source-target in the network for generation of packets, for example: Complement, Bit Reversal, Butterfly, Matrix Transpose and Perfect Shuffle, as seen in [7]. These patterns consider the data exchanges that parallel applications usually perform.

The latency is the interval of time between the start of the message transmission and the reception of it on the target node [7]. However latency is a vague concept [3]. It may be the time between the injection of the packet's header into the router and the reception of its last information or it may count the time just on the routers, not on the nodes. The latency is measured in cycles or another time unit.

The throughput is the quantity of information transmitted on the network per time unit [8]. It may be measured in number of bits transmitted on each pair source-target or in percentage of this link utilization. In this last case, a communication link works on its maximum capability if it is used 100% of the time [3].

4. Core Architecture

This paper proposes two systems based on cores for NoC performance evaluation: one measures the latency and the other measures the throughput. When coupled on the NoC routers, these cores return values of network performance. The systems were constructed to integrate hardware and software on a FPGA device, which allows fast and accurate evaluations. Initially, both systems were implemented together, but the silicon costs of the full system were too high for the FPGA device used in this paper. Therefore, the systems were separated in order to ensure low silicon costs.

4.1. Latency-based System

The system that measures latency is illustrated in fig. 2, in which C represents the core. The hardware of the system communicates with a software application executed on the Nios II/e processor, through the Avalon bus. A memory module coupled on this bus and controlled by the Nios II/e processor completes the system. The software Altera Nios II IDE provides a graphic interface for the user to run the evaluation and analyze the results.

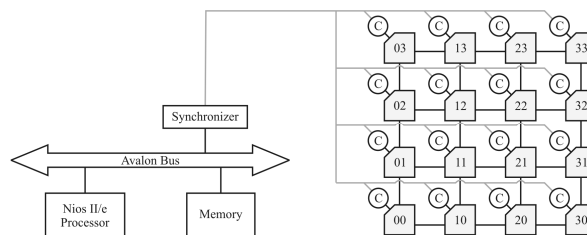


Fig. 2 – Latency-based system architecture

Each core has a sender, a receiver and a timer module in order to count the number of cycles spent between the start and the end of a packet transmission. The application communicates with the cores through a synchronizer, which warranties a parallel configuration of the cores and doesn't make use of the NoC links for that. The application executed on the Nios II/e processor sends configuration parameters to the cores, such as pairs source-target, size of the packets, injection rate and number of packets collected per result. Then the configured cores generate traffic on the NoC. The latency results measured by the cores are sent back to the Nios II/e and collected by the software application. These data transferences must be adapted to NoC's packet format and addresses, such as shown in fig. 3, where the packets were adapted to the NoC used in the results section of this paper. In this figure, the 4 types of packets used in the system are detailed: a) receiver configuration, b) sender configuration, c) generated traffic and d) latency result.

The receiver configuration packet must be sent by the application to define the number of packets (NoP) that the cores must receive before returning the latency result. Then, the application must transmit one sender configuration packet to each core sequentially to define the target receiver address (RA), the size of the packet

(SoP) generated and the delay between each packet injection. The generated traffic packets are automatically sent to the receiver address, carrying the sender time (ST) to calculate the latency. Finally, when the core receives NoP packets, the latency result packet is sent to the processor to return the sum of latencies (SoL) accumulated.



Fig. 3 – Packets' format in the latency-based system

4.2. Throughput-based System

Structurally the system that measures throughput is formed by the same components as the latency-based system. The adaptations were restricted to the core, the synchronizer and the interface between core and router. It's a different type of measure, so the core and the synchronizer behave in a different way. In the core-router interface four ports were inserted to count the number of packets transmitted on the four router links, as illustrated in fig. 4.

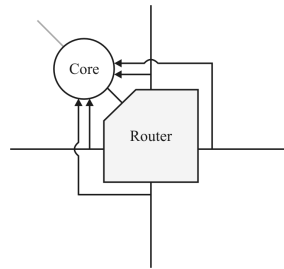


Fig. 4 – Alteration for the throughput-based system architecture

A counter module was inserted into the core to count the throughput on the links at the same time that the packets are sent and received. The system packets for the throughput evaluation of the NoC used in the results section of this paper are illustrated in fig. 5: a) counter configuration, b) sender configuration, c) generated traffic and d) throughput result. In the counter configuration packet, CT (counter time) defines the interval of time spent during the measurement. The throughput result packet returns to the processor the number of packets counted on north (NNoP), east (ENoP), south (SNoP) and west (WNoP) link.

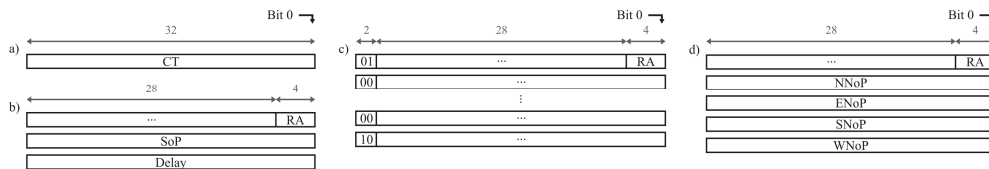


Fig. 5 – Packets' format in the throughput-based system

5. Results

In order to obtain experimental results and validate the systems developed, both of them were applied to the NoC SoCIN, a very well recognized and documented interconnection network that implements the router ParIS [9]. The NoC SoCIN has 2-D grid (or mesh) topology and wormhole packet-based switching. Other aspects are parameterizable. In this project, a NoC of dimensions 4x4 was configured to use deterministic (XY) routing, handshake flow control, dynamic (round robin) arbitration and FIFO memorization on the input ports.

The implementation in the hardware description language VHDL of each system was integrated to the SoCIN implementation, also described in VHDL. Then the complete systems were synthesized and prototyped on the Altera DE2 FPGA board, specifically the device EP2C35F672C6 of the family Cyclone II, obtaining fast and accurate NoC evaluations. Tab. 1 shows the silicon costs and the maximum operational frequency of each system after the FPGA prototyping.

Tab. 1 – Silicon costs and maximum operational frequency of each system

System	Logic Elements	F _{max} (MHz)
Latency-based	26130 (79%)	71.43
Throughput-based	20212 (61%)	91.63

With the systems prototyped on FPGA, software applications implemented in the programming language C and executed on the Nios II/e processor injected configuration packets into the systems, defining parameters for measuring latency and throughput and selecting one of a set of programmed traffic patterns: Complement, Bit Reversal, Butterfly, Matrix Transpose and Perfect Shuffle. The results were collected by the same application.

In fig. 6, a graphic of average latency per core and the throughput on each SoCIN link as a percentage of utilization are illustrated for the traffic pattern Complement. On the left, as expected from this pattern, the cores located in the center of the network, that is, where the source core is near the target core, present lower latency values when compared to the peripheral ones. And on the right, the highest throughput values are highlighted and located exactly on the links with the biggest access disputes, according to the static paths of the packets generated by the pattern Complement. For the other patterns, the latency results were also related to the distance between the pairs source-target and the throughput results were also related to the network congestion caused by the static paths of the generated packets.

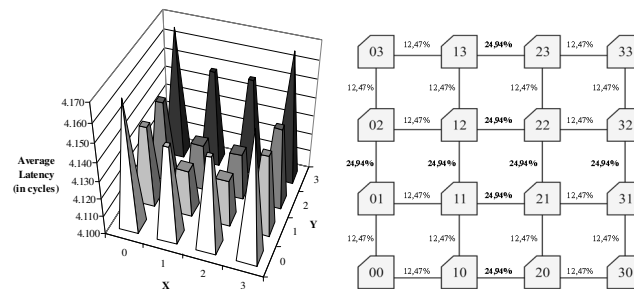


Fig. 6 – Average latency per core (on the left) and throughput per link (on the right) for the Complement

6. Conclusion

This paper presented two proposals of systems based on traffic generator cores for Network-on-Chip performance evaluation: one measures latency and the other measures throughput. The experiments were performed using a FPGA device, which provided fast and accurate evaluations. Both systems were applied to the NoC SoCIN in order to verify the consistence of the results. It was concluded that the systems are consistent, because the results revealed the traffic patterns' tendency of behavior. Besides, the utilization of both systems allows an important type of NoC evaluation, because the measures latency and throughput are the most important for interconnection mechanisms performance evaluation.

The methodology presented in this paper may be applied to any type of NoC that implements the handshake flow control, adapting only the format of system packets to the NoC packet format. Future works include improvements on silicon costs of the systems and the implementation of others flow control mechanisms, allowing the evaluation of more NoC designs, including larger ones.

7. References

- [1] Loghi, M. et al. "Analyzing On-Chip Communication in a MPSoC Environment". In: Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings... [S.l.:s.n.], 2004.
- [2] Benini, L.; Micheli, G. D. "Networks on Chips: A New SoC Paradigm". IEEE Computer Society Press: 2002.
- [3] Duato, J.; Yalamanchili, S.; Ni, L. "Interconnection Networks". Elsevier Science, 2002.
- [4] Zeferino, C. A. "Redes-em-Chip: Arquitetura e Modelos para Avaliação de Área e Desempenho". Tese (Doutorado em Ciências da Computação) – Instituto de Informática - PPGC, UFRGS, Porto Alegre, 2003.
- [5] Wolkotte, P. T.; Hölzenspies, K. F.; Smit, J. M. "Fast, accurate and detailed NoC simulations". In: International Symposium on Networks-on-Chip (NoCS), 2007.
- [6] Genko, N.; Atienza, D.; De Micheli, G.; Mendias, J. M.; Hermida, R.; Catthoor, F. "A Complete Network-On-Chip Emulation Framework". Design, Automation and Test in Europe (DATE), 2005.
- [7] Mello, A. V. "Canais Virtuais em Redes Intra-Chip: um Estudo de Caso". Trabalho Individual I, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2005.
- [8] Dally, W.; Towles, B. "Principles and Practices of Interconnection Networks". Morgan Kaufmann Publishers Inc.: 2003.
- [9] Zeferino, C. A.; Santo, F. G. M. E.; Susin, A. A. "ParIS: A Parameterizable Interconnect Switch for Networks-on-Chip". In: 17th Symposium on Integrated Circuits and Systems Design (SBCCI), 2004, Porto de Galinhas. Proceedings. New York: ACM Press, 2004.



Test and Fault Tolerance

A Design for Test Methodology for Embedded Software

Guilherme J. A. Fachini, Humberto V. Gomes, Érika Cota, Luigi Carro
{gjafachini, hvgomes, erika, carro}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul

Abstract

This work discusses appropriate model and coding techniques that allow the software tester plan and execute most tests, including hardware and software integration validation, early in design, without hardware platform. The software development and debugging can be performed with high level development tools and, through hardware devices modeling, defined test cases can be re-applied to the system when the hardware becomes available. This proposal can efficiently reduce design and debug time for embedded systems while ensuring the system quality.

1. Introduction

Embedded systems are special propose computer devices that have some limitations like low power consumption, little memory availability and much software and hardware coupling. Despite these restrictions, current embedded devices have become extremely complex, both software as hardware [1] [2].

This way, allied to small time to market demand, is necessary development and test methods capable to, at same time, guarantee on time product release, ensuring its quality. Due to strong dependence between hardware and software is very difficult test embedded software with traditional software tools, besides the software must be developed on very specific design tools, which, often, do not have debugging and simulation resources. Finally, embedded software testing involves execute it on a fully functional hardware device, that is normally available latter on the design process.

Embedded software development and test have been studied by many authors as [1][2][3][4]. However, problems concerning structure, debugging and simulation strategies, test cases application and the use of package of software test tools have not been sufficiently addressed.

To ensure market requirements this work presents a methodology for testing embedded software where project development is though, since first time, aiming test. By the application of sufficiently accurate hardware device models and generation of test cases compatible with models and target platform, this methodology aims to reduce the overall testing and development time. Furthermore, the embedded software development and test process can be made with high level tools without hardware or a debugger platform, and, due to hardware devices similarities, test cases and hardware devices can be easily reused on others embedded projects.

This paper is structured as follow: Section 2 presents some embedded software test challenges, Section 3 discusses some relevant works and Section 4 introduces the proposed methodology. Conclusion and future work are presented on Section 5.

2. The challenges of embedded software testing

One of the reasons that make research of embedded software development methodologies so important is that, nowadays, the final user expects devices with performance and quality of supercomputers with price and power dissipation of portable FM radios.

Performing as much as possible of the development cycle of embedded software using high level design tools, without the a physical prototype is one of the objectives of the proposed development and test methodology. Due to the strong dependence with the hardware behavior, the hardware-dependent software and its interfaces are the main responsible by the difficulties found in using a standard software testing tools.

Analyzing these softwares, standard complexity metrics simply cannot be used. Fig. 1 shows a code fragment of an analog to digital data conversion routine. When applying to this routine the McCabe cyclomatic metric, which measures the number of linearly independent paths of the code [6], the results show that this is a very simple code. However, this is a critical and key routine that hides several bugs. This is easily explained because several pins of the AD converter interface are mapped to memory positions. Hence, all complex AD configurations is seen by the test tool as simple writes to a memory position, so deemed simple.

```

#pragma vector = TIMERB0_VECTOR
__interrupt void energy_metering(void)
{
    TBCCR0 += TB0_SAMPLING;
    int16 index = 0;
    do
    {
        ADC10CTL0 &= ~ENC;
        ADC10CTL1 = sampling_Seq[(IA + index)];
        ADC10CTL0 |= ENC + ADC10SC;
        self.mtrEngine.bufferDelay[(IA + index)] =
            self.mtrEngine.buffer[(IA + index)];
        while (ADC10CTL1 & ADC10BUSY);
        self.mtrEngine.buffer[(IA + index)] = (ADC10MEM << 4);

        ADC10CTL0 &= ~ENC;
        ADC10CTL1 = sampling_Seq[(VA + index)];
        ADC10CTL0 |= ENC + ADC10SC;
        self.mtrEngine.bufferDelay[(VA + index)]
            = self.mtrEngine.buffer[(VA + index)];
        while (ADC10CTL1 & ADC10BUSY);
        self.mtrEngine.buffer[(VA + index)] = (ADC10MEM << 5);

        ADC10CTL0 &= ~ENC;
        ADC10CTL1 = sampling_Seq[(IA + index)];
        ADC10CTL0 |= ENC + ADC10SC;
        while (ADC10CTL1 & ADC10BUSY);
        self.mtrEngine.buffer[(IA + index)] += (ADC10MEM << 4);
        self.mtrEngine.buffer[(IA + index)] -=
            ((int16)(self.mtrEngine.offset[(IA + index)] >> 16));
    } while (++index < NUM_PHASES);

    __bic_SR_register_on_exit(CPUOFF);
}

```

Fig. 1 – Strong hardware-dependent software routine

The main concept of the proposed methodology is having hardware-devices models that simulate such devices behavior. The models are programmed on the same programming language of the embedded software and stored into libraries. The hardware models are not intended to be capable of simulating the hard real time interface devices behavior, since such characteristics can be debugged during last project steps by the execution of the hardware-dependent code on the real device, supposed to be available at this time. The idea is to have sufficiently accurate models that simulate the real device behavior at instruction level, leading to large test cases code coverage.

The application of models to the design of embedded systems can be found in many works and development strategies [4][7]. For instance, the multiple-V model suggests the division of the embedded system design into three steps, the model design, the prototype design and the final product design, not necessarily executed as a straightforward sequential process [7].

The proposed methodology does not assume the fully absence of the physical hardware devices during project, but it has the objective of transforming the hardware-dependent software development into a purely software environment. Furthermore, the concept of developing only one software that can be executed in the embedded system model and in the final product, making possible the application of exactly the same test cases, is very attractive. By the methodology application, at the project end, when software must be executed into physical devices, is expected that the software will be sufficiently mature and tested, reducing, in this way, the possibilities of software failures during the field tests.

3. Related work

Embedded systems development has been the target of many works aiming at meeting the design and time to market requirements [1][2][3][4][8]. In [1], a design methodology and an architecture style for embedded systems are presented. The proposed design methodology consists of the requirements specification, hardware and software partitioning and the software architecture design, where the overall system structure is described by the components and the interconnection among them. The final steps are the functional test, composed by the unit and integration tests, host static test and system test, where the whole system is executed in the target device. For the system test, the authors propose a method for tracing the execution states during running the target system by the inclusion of a test driver into the source code.

The referenced works confirm the importance of the hardware-software interface the embedded software, the necessity of the generated code organization and the benefits of developing with high level software and simulation tools.

4. The design and test methodology

Due to embedded systems characteristics, it is very difficult to perform the embedded software test without thinking in a design methodology focused into test since the initial development steps. In this paper, the

proposed methodology divides the design of the embedded software into the following steps: hardware analysis, hardware devices modeling, hardware dependent software development and packaging and application software design. The test cases generation and validation are performed during the developed modules construction. Fig. 1 shows an overview of the proposed methodology.

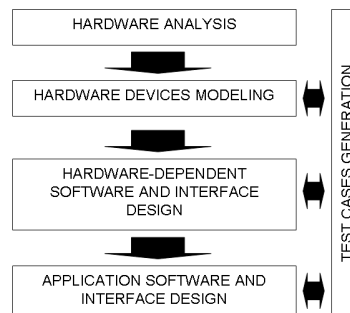


Fig. 1 - the design and test methodology overview

4.1. The hardware analysis

At this point, the design of the embedded system is analyzed and, from the initial requirements, all the necessary hardware devices are mapped. Examples of hardware devices are liquid crystal displays, external memories, input and output pins, analog-to digital and digital-to analog converters, communication devices and keyboards.

4.2. Hardware devices modeling

Before the construction of new hardware models of the devices mentioned above, it is very important to search the model libraries for similar ones. These models do not intend to be hard-real time models, but only the functional behavior of the hardware devices will be modeled, at the instruction level. For example, an internal microprocessor hardware device, like a flash memory, is usually interfaced by writing and reading specific internal memory-mapped registers. The registers content changes over the time according to the specific hardware device protocol. The created model behavior will respect this hardware protocol.

The models are created with the same programming language of the application software, and are stored into source code libraries, allowing the application of high level test tools like Cantata [9], which usually compiles the embedded software code. Furthermore, the models source code can be easily compiled with high level design tools [10]. The external interface and internal structure of the models are similar, being composed by an execution function and an internal state machine, respectively, as showed in Fig. 2.

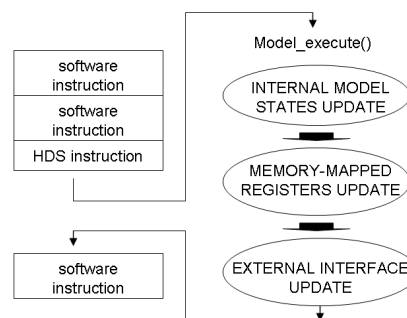


Fig. 2 - internal model execution flow and external interface

4.3. Hardware-dependent software and interface design

The design of the hardware-dependent software is performed exactly as specified by the hardware device manufacturing, but adding, after each instruction that access a hardware-specific memory-mapped register, a call to the *model_execute()* function. By calling this function, the internal state variables of the model are updated, together with the external memory-mapped registers and interfaces of the hardware device model as showed in Fig. 2.

Fig. 3 shows a fragment of code that exemplifies the model interface to a 10 bit analog-to digital converter model.

```

//Access to hardware memory-mapped register
ADC10CTL0 &= ~ENC;
//Execution of the device model
ADC10_model_execute();
ADC10CTL1 = sampling_Seq[(IA + index)];
//Execution of the device model
ADC10_model_execute();
ADC10CTL0 |= ENC + ADC10SC;
//Execution of the device model
ADC10_model_execute();

```

Fig. 3 - the hardware device model interface

4.4. Application software and interface design

Designing the application software starts with the definition of all the necessary software components and their interfaces. The application software interfaces are the interconnections between the individual software modules, composed by the necessary sets of data and messages, which can be derived from the embedded software specifications [1]. Just like in the hardware-dependent software design, for each application software module, test cases must be constructed to test the application functionality and the interface usage.

5. Conclusion and future work

The application of this development and test methodology in a complex commercial embedded system produced in high volumes and subjected to a broad quality assurance standards, intends to help the company and the design engineers on meeting the time to market and quality requirements. Furthermore, at the same time that a library of hardware models, test cases and application software, fully reusable into new designs is generated, the development and test team are creating a culture of test since the beginning of the project specification.

This work will continue by detailing the proposed hardware models, software interfaces and used test cases that compose the application of this methodology.

6. References

- [1] B. Kang, Y. Kwon, "A Design and Test Technique for Embedded Software", Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA), 2005.
- [2] J. Seo, Y. Ki, B. Choi, K. La, "Which Spot Should I Test for Effective Embedded Software Testing?", Proceedings of the Second International Conference on Secure System Integration and Reliability Improvement, 2008.
- [3] T. Kanstrén, "A Study on Design for Testability in Component-Based Embedded Software", Proceedings of the Sixty International Conference on Software Engineering Research, Management and Applications (SERA), 2008.
- [4] J. Engblom, G. Girard, B. Werner, "Testing Embedded Software using Simulated Hardware", Proceedings in Embedded Real Time Systems (ERTS), 2006.
- [5] J.L. Anderson, "How To Produce Better Quality Test Software", IEEE Instrumentation & Measurement Magazine, 2005.
- [6] T. McCabe, "Structured testing: A testing methodology using cyclomatic complexity metric", National Institute of Standards and Technology, 1996.
- [7] Broekman, B. and Notenboom, E., "Testing Embedded Software", Addison-Wesley, 2003.
- [8] A. Sung, B. Choi, S. Shin, "An interface test model for hardware-dependent software and embedded OS API of the embedded system", Computer Standard & Interface, Vol. 29, 2007, pp 430-443.
- [9] Cantata ++ software, available in <http://www.ipl.com/products/tools/pt400.uk.php>
- [10] Borland C++ Builder 2006, available in <http://www.borland.com/br/products/cbuilder/index.html>.

Testing Requirements of an Embedded Operating System: The Exception Handling Case Study

Thiago Dai Pra, Luciéli Tolfo Beque, Érika Cota
thiagodaipra@gmail.com, ltbeque@inf.ufrgs.br, erika@inf.ufrgs.br

Universidade Federal do Rio Grande Do Sul

Abstract

Real-time operating systems (RTOS) are becoming quite common in embedded applications, and to ensure system quality and reliability, its correct operation is crucial. However, there hasn't been much embedded software testing literature that deals with this specialized software testing. This paper presents a preliminary study on the testing of the exception handling routines of an embedded operational system (EOS). We start analyzing the difficulties of testing a RTOS and then, we present some experiments that will help to devise a test methodology for this specific software.

1. Introduction

An embedded system is a combination of software and hardware designed to perform a specific task, a micro processed system that supports a particular application [1, 2]. In case of systems containing programmable components, the software can be composed of multiple processes, distributed among different processors and communicating by means of various mechanisms. In current developed applications, software is advancing to constitute the main part of the system, and in many cases software solutions are preferred to hardware ones, because they are more flexible, easier to update and can be reused [3].

For these systems, a RTOS may be necessary to offer supporting services for process communication and scheduling [1]. As the complexity of the application increases, the OS becomes more important to manage hardware-software interactions.

Embedded systems have well known limitations regarding memory availability, performance, and power consumption, among others. They are often used in life-critical situations, where reliability and safety are more important than performance [4]. Consequently, a carefully planned test has become very important to ensure good quality of the product without deeply affecting design time, constraints and cost.

We start discussing the constraints involving the RTOS testing in Section 2. Section 3 presents a brief overview in the related work. Section 4 describes eCos, our case study. Section 5 presents the fault model defined for the exception handling flow of eCos and the fault injection campaign implemented. Section 6 describes the definition of the test cases. Section 7 concludes the paper.

2. Test Requirements for a RTOS

The routines that compose an RTOS may be programmed in different abstraction levels: from higher-level routines that can be tested as embedded software, to low-level routines requiring specifics test procedures. For example, the interrupt handling routines has part of the code written in assembly to implement the interaction with the microprocessor. For this part, traditional test methods may not apply, because they are usually developed for high level programming languages, and an analysis disconnected from the hardware platform, although the code simplicity, has no meaning. Also, a C-code can access a memory position that represents a memory-mapped register, and a standard testing tool may consider the code fault-free during simulation, even if there is an error in the index of the accessed memory. Finally, the execution of those low-level routines without the hardware or a detailed hardware model (i.e. a simulator) is not accurate and may lead to lower coverage.

It is also important to consider the OS test and the embedded application test together to reduce design time. Thus, during the application test, the EOS must be present. If there is error detection, the designer needs all possible information to locate the fault. Fault in the EOS can interfere in the execution of the test planed for the application, causing misdiagnosis and increasing debug time.

Thus, we believe that a structured test method for the EOS is crucial in the design flow of an embedded application.

3. Related Work

Most works on embedded software testing present test solutions developed for specific applications. There is not a general methodology to test an embedded system. The most used technique seems to be the model-based testing, conducted to determine whether a system satisfy its criteria for acceptance.

Concerning testing of operating systems Yunsu et. al. [5] and Tan et. al. [6] use the embedded Linux OS, and have, respectively, as their main objective, to minimize the energy consumed in the execution of OS functions and services, and analyze the energy consumption characteristics of the OS.

Few studies focus on the interaction of the hardware and the software when testing an embedded system. A definition of embedded software interface and a proposition that it should be considered as a critical analysis point for testing embedded software are presented in [9]. This model takes into account a hierarchical structure: hardware-dependent software layer, operating system layer, and applications layer and defines a check list to improve the test coverage.

4. The Embedded Configuration System (eCos)

The eCos is a free RTOS which main characteristic is the configurability that allows the designer to choose the features that will be used by the application, making it a system simpler to manage and smaller in terms of memory [7, 8].

The main components of eCos are described below [8]:

- **Hardware Abstraction Layer (HAL)** – provides a software layer that gives general access to the hardware.
- **Kernel** – includes interrupt and exception handling, thread and synchronization support, scheduler, timers, counters and alarms.
- **ISO C and math libraries** – standard compatibility with function calls.
- **Device drives** – includes standard serial, Ethernet, Flash ROM and others.

For our experiments, we focused in the exception handling execution flow of eCos, because of its interaction between hardware and OS. This execution flow will be presented next.

4.1. eCos Exception Handling

An exception is a synchronous event that changes that occurs during a thread execution and changes the normal program instruction flow. If the exception isn't processed during the program execution, several consequences, ranging from a simple inconsistency to a complete system failure, may occur.

On eCos there are two kinds of exception handling: one is a combination of HAL and Kernel's Exception Handling, and the other is the application itself that handles the exception. For our experiment we configured the system to use the HAL and the Kernel's exception handling logic, which is shown in fig. 1 flow diagram.

4.2. Embedded Application

Two basic applications were implemented to be executed on top of the RTOS: the Loop and the Bubble Sort. The loop application consists in a simple program, which generates random values within a loop. The user input is the amount of random values that the program will generate and the program generates values in the range [0, 5]. The Bubble Sort sorts the values (5, 4, 3, 2, 1, -5, -4, -3, -2, -1). Both applications force divide by zero exceptions during the execution.

5. Fault Model Definition

Four types of faults were defined to consider different fault locations (system layers), target code level (assembly or high-level code), interaction faults (HW-SW and SW-SW), and also HW faults (stuck-at) are taken into account. The four fault types are:

- 1) *Code faults*: changes in the program code (OS or application, assembly or high-level code). Include, for instance, a fault in the code (change of operation or operator) or code omission.
- 2) *PC faults*: faults in the program counter. Note that this fault can be a hardware fault (stuck-at in one bit of the register) or a software fault (wrong assignment, for example)
- 3) *Registers faults*: changes in the registers. Again, can be a hardware or software fault.
- 4) *Parameters faults*: change or deletion of some argument when calling a routine.

For each type of fault, a few faults were injected. In fig. 1, the boxes identify the location, type, and number of injected faults. Faults were manually included in the application, the HAL and the Kernel.

6. Experimental Setup

For the experiments, we chose SIMICS simulator, because it allows arbitrary fault injection in the system through a powerful mechanism, and the simulation is performed at instruction level. The simulated platform processor used is the i386 (lower-cost version, specific for the embedded market).

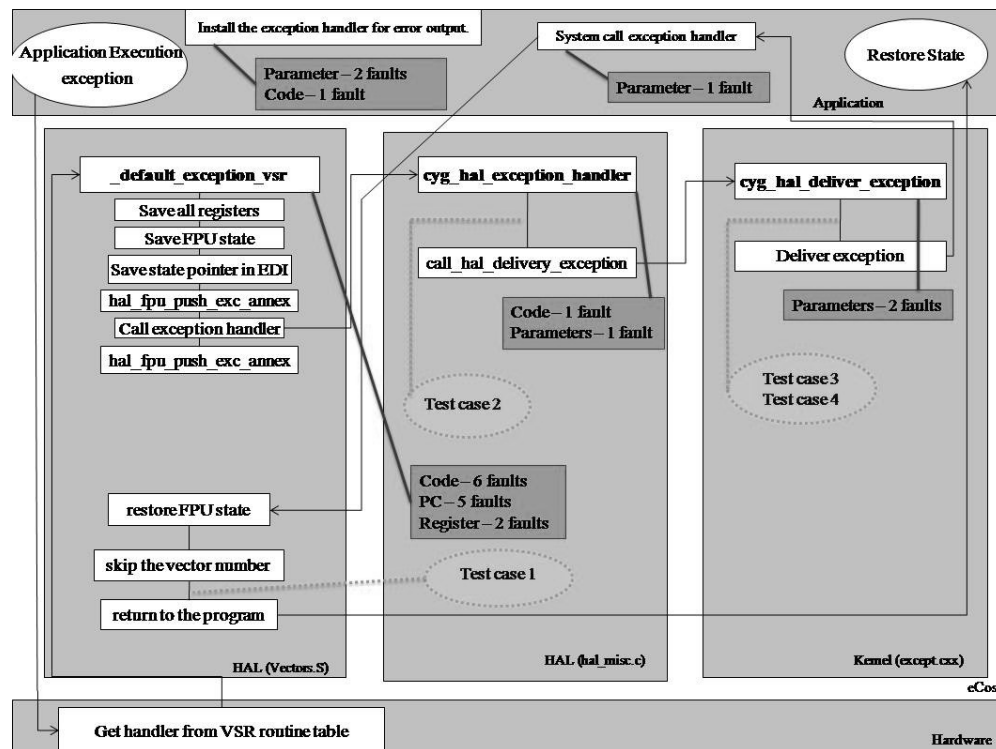


Fig. 1 – Implemented exception handling execution flow.

6.1. Fault Injections Results

Fig. 2 shows the results of the fault injection campaign which consists of 22 faults injected. We observed two possible outcomes for a fault: a catastrophic effect (where the program stops its execution after the exception handling) and a non-catastrophic behavior (where the exception was handled and the normal flow of the program is resumed).

From fig. 2, we can see that 9 out of 22 (41%) of the injected faults led to catastrophic effects. Those faults involve or affect special registers that store the system status, and are often related to violations of memory.

Among the non-catastrophic faults, only two PC faults led to an incorrect system behavior. The PC faulty value for those cases was set to the beginning of the application or the first instruction executed after a reset. For those cases, the system remains operational but the application executes for more cycles and outputs extra results.

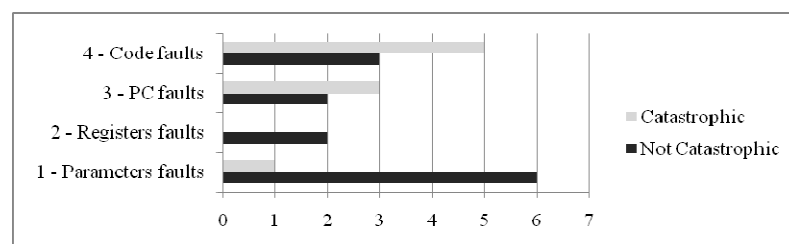


Fig. 2 – Fault injection results

6.2. Test Cases Definitions and Results

From the fault injection campaign four test cases were inserted in the exception handling flow of eCos, to help detection and diagnosis. In fig. 1, the circles show the location of the test cases on the code.

Test Case 1 is implemented in assembly, and is inserted to detect the catastrophic faults at the HAL level, which causes a GPF. This test case verifies, immediately before the end of the exception handling routine, if the PC is valid, i.e., if its value is between the first and the last address of the application. The test code is located in the starting point for all packages of HAL.

Test Cases 2 and 3 are added to detect faults in the parameters passed to the functions of exception handling. Test Case 2 is inserted in the HAL and Test Case 3 in the Kernel. These tests check whether the HAL_SavedRegisters structure has a valid value.

Finally, Test Case 4 aims the faults that might possibly occur in the number of the exception. For this case study, the exception number is 0 (indicating a division by zero). This parameter is at the kernel level and the test case checks it along the execution flow.

Tab. 1 shows the relationship between each Test Case and the kind of fault it detects. We note that Test Case 1 can be reused for other exceptions, and also for the flow of the interrupt handler.

The main targets of the test cases are the faults in the OS code. Faults in the application or in the exception handling routine in the application level will be detected by application-level test cases. Thus, to evaluate the efficacy of the test cases, only OS-specific faults were injected in this second fault injection campaign. The result is that 12 out of the 17 injected faults were detected by at least one test case. The 5 undetected faults include 3 faults in the PC that preclude the test case execution and cause a catastrophic effect, and 2 faults in specific registers, which do not affect the program execution and the exception handling.

Tab. 1 – Test cases target faults

	Code Faults	PC Faults	Parameter Faults
Test Case 1	X	X	
Test Case 2			X
Test Case 3			X
Test Case 4			X

7. Final Remarks

This paper presented a preliminary study on the requirements and constraints for the test of a RTOS. From this case study we can conclude that structural testing can be of great value to this specific software, although the standard test strategy for operating systems is the use of pure functional testing.

Current work includes the definition of a more complete fault model and its correspondent test cases. Then, the defined fault model and test strategy will be applied to another RTOS to support the definition of a more generic methodology.

8. References

- [1] Burns, A. and Wellings, A. (1997). "Real-Time Systems and Programming Languages." Addison-Wesley, 1997.
- [2] Barr, G.O. "Programmin Embedded Systems in C and C++". 1st ed. Sebastopol: O'Reailly & Associates Inc. 1999.
- [3] Bas Graaf, Marco Lorman, Has Toetenel, "Embedded Software Engineering: The State of the Practice", IEEE Software, vol. 20, no. 6, pp. 61-69, Nov/Dec, 2003;
- [4] Edwards, S. Lavagno, L. Lee, E.A. Sangiovanni-Vincentelli, A. "Design of embedded systems: formal models, validation, and synthesis." Proceedings of the IEEE. On page(s): 366-390 Volume: 85, Mar 1997.
- [5] Fei, Yungsi; Ravi, Srivaths; Raghunathan Anand and Jha, Niraj K. "Energy-Optimizing Source Code Transformations for OS-driven Embedded Software." Proceedings of the 17th International Conference on VLSI Design (VLSID'04).
- [6] Tan, T. K.; Raghunathan, A. and Jha, N. K. "A Simulation Framework for Energy-Consumption Analysis of OS-Driven Embedded Applications". IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems, Vol. 22, n°. 9, September 2003.
- [7] eCos (2008). <http://ecos.sourceforge.org/> April.
- [8] Massa, Anthony J. (2002) "Embedded *Software* Development with eCos." Upper Sadle River: Prentice Hall PTR.
- [9] Sung, Ahyoung; Choi, Byoungiu and Shin, Seokkyoo. "An Interface test model for hardware-dependent software and embedded OS API of the embedded system." Computer Standards & Interfaces 29 (2007) 430-443.

Testing Quaternary Look-up Tables

¹Felipe Pinto, ¹Érica Cota, ¹Luigi Carro, ¹Ricardo Reis
{fapinto,erika,carro,reis}@inf.ufrgs.br

¹ Universidade Federal do Rio Grande do Sul

Abstract

The use of quaternary logic has been presented as a cost-effective solution to tackle the increasing area and power consumption of million-gate systems. This paper addresses the problem of testing a look-up table based on quaternary logic. Two possible test strategies are discussed in terms of number of test vectors and total test time for a programmable and regular architecture based on the quaternary logic blocks. We show that the test cost for this new design paradigm is also very competitive and comparable to the test cost of a binary-based look-up table used in standard FPGAs. With this result, we conclude that the configurable quaternary look-up table is an interesting solution not only in terms of design capabilities, but also in terms of test costs.

1. Introduction

With the advent of deep submicron technologies, interconnections are becoming the dominant aspect of the circuit delay for state-of-the-art circuits, and this fact is becoming even more significant to each new technology generation [1]. This is due to the fact that gate speed, density, and power scale according to Moore's law, whereas the interconnection resistance-capacitance product increases with each technology node, with a consequent increase in the network delay. Interconnections play an even more crucial role in Field Programmable Gate Arrays (FPGAs) because they not only dominate the delay [4], but they also severely impact power consumption [5] and area [6]. For million-gates FPGAs, as much as 90% of chip area is devoted to interconnections.

As an alternative to the binary logic, multiple-valued logic can compact information, since a single wire can transmit more than two distinct signal levels. This can lead to a reduction on the total interconnection costs, hence reducing area, power and delay, which is especially important in FPGA designs. Area reduction in FPGAs by means of Multiple-valued logic (MVL) has been shown in [7-9], but their excessive static power consumption and complex realization have precluded their acceptance as a viable alternative to standard CMOS designs. Regarding combinational circuits, several kinds of MVL circuits have been developed in the past decades in several different technologies from earlier works on bipolar technologies to novel solutions presented recently using floating gates [6], capacitive logic [7] and quantum devices. Some circuits already developed a quaternary general-purpose circuit in voltage mode technology that is the realization of multiplexer with high performance, negligible static and low dynamic consumption using fewer transistors than the equivalent binary circuit.

Using quaternary circuits, the total number of interconnections can be reduced, as shown in [7], thus reducing the total area and power consumption of the so-called quaternary FPGA. Indeed, results have shown [3] that a quaternary-LUT-based FPGA can implement the same logic of a binary FPGA using 50% less area and the power consumption can be as low as 3% of the power consumption of the correspondent binary circuit. This improvement in power consumption is due to the reduction in the number of transistors and lower average voltage, since some signals are fractions of the binary VDD voltage. Multiple-valued logic has more representational power since a single wire can transmit more than two distinct signal levels.

However, to become an industrial reality, this new design paradigm also depends on the definition of cost-effective test mechanisms. It is important to evaluate whether existent fault models and test methods can be used or easily adapted to ensure a high quality and low cost production test for quaternary FPGA circuits.

In this paper we evaluate the test requirements of a quaternary look-up table and devise two test strategies for this block based on a catastrophic fault model. We start by evaluating the effect of some defects in the LUT behavior and the validity of the stuck-at fault model for the multiple-valued logic. In the sequence, a resource-aware test method is devised and analyzed. Then, a second test strategy is devised to optimize the test application time-aware method. Our results show that the test of the QLUT can be as efficient as the test of its binary counterpart. We conclude that also from the test point of view, this new design paradigm can represent an interesting solution to tackle the current CMOS design challenges.

The paper is organized as follows: Section 2 presents the structure of the QLUT. Section 3 presents the fault model used in this work and an evaluation of the QLUT faulty behavior for three possible defects in the LUT structure. In Sections 4 and 5 the two test strategies are presented while Section 6 concludes the paper. In Section 7 the references.

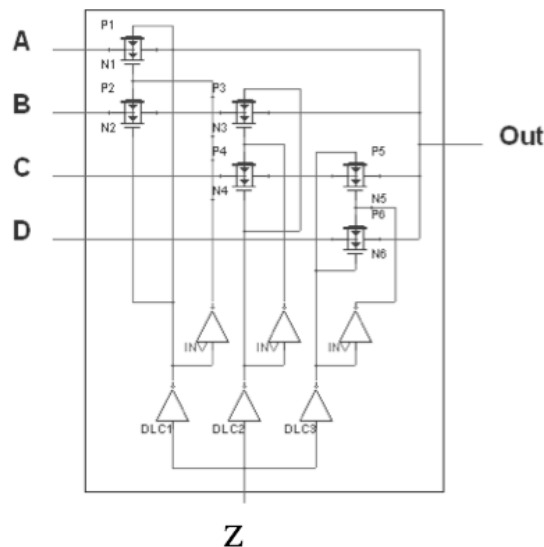


Fig. 3 – Quaternary look-up table schematic

The output of DLC1 and its negative are applied to the NMOS pass transistor gate N1 and PMOS pass transistor P1, respectively, setting the output to the A value. In the same time these two signals are also applied respectively to the PMOS P2 and to NMOS N2, turning them off and therefore opening the path between B and the output. The DLC2 and DLC3 output and their complement are used to turn off P4, N4, P6 and N6, opening the paths between the output and C and D. If Z is set to 0.6V, the output of DLC1 is 0 and P1 and N1 are both off, while P2 and N2 are on, and the output is connected to B while the path between output and 0(A) input is opened. In the same way values C and D can be propagated to the QLUT output.

The quaternary logic has a few sensitive points, including the necessity of additional implants, reduced noise margins and the need for three different voltage sources. However, due to the fully compatibility of this quaternary design with present state-of-the-art CMOS technology, these issues can be tackled during design time. For example, by using Monte Carlo simulations we have verified that even for a 32nm process the QLUT still shows correct performance for all process corner cases.

3. Fault Model for the Quaternary LUT

For the QLUT depicted in fig. 3 we consider a catastrophic fault model that is consistent with the analog behavior of the DLCs and the other supporting transistors. For every transistor in the QLUT, a short and an open fault are assumed. Furthermore, for each DLC, a catastrophic deviation of V_t (threshold voltage) is also assumed. This deviation corresponds to the worst case, when the V_t deviation makes one DLC change its correct output value. For example, a faulty DLC1 would behave like a DLC2.

A total of 54 faults were injected in the QLUT, one at a time, to evaluate the faulty behavior of the block depicted in fig. 3. Tables 4, 5, and 6 detail the results obtained through Spice simulation for opens, shorts, and V_t deviations, respectively. Tab. 4(a) presents the results for the open faults in the twelve transistors that compose the set of transmission gates; Tables 4(b) and 4(c) show the results for the open faults affecting the transistors of each DLC and each inverter, respectively. Similarly, Tables 5(a), 5(b) and 5(c) correspond to the short faults in the transmission gates, DLCs and inverters, respectively. In all tables, each column indicates the faulty transistor while each line corresponds to a QLUT input value. During simulation, the QLUT block is configured in combinational mode, i.e., the QLUT output is not registered. Furthermore, we implemented an exhaustive simulation for all faults, i.e., for each injected fault, all possible combination of values in the SRAM cells (256 combinations) are tried and, for each combination, all possible values for the QLUT input (4 values) are also applied. Thus, for each fault, a total of 1024 simulations are run. An entry marked as “Ok” in the table means that the QLUT behaves correctly for all 1024 simulations, i.e., for all possible values of Z and SRAM combinations the output is the correct one. Otherwise, an incorrect behavior is detected.

Based on the results of Tables 3, 4, and 5, incorrect behaviors can be classified in three groups: wrong value, wrong selection, and multiple drive error. A “wrong value” faulty behavior means that the correct SRAM position is selected according to Z, but a specific value in that position is not correctly propagated to the QLUT output. This is the case, for instance, of an open fault in transistor p1 of tab. 4(a) and transistor t9 of tab. 4(c) (INV3). For both faults, the expected value is logic value 3, while the actual value obtained in the output is given in parenthesis in the tables (2.43 and 2.21, respectively). A “wrong selection” faulty behavior means that the incorrect SRAM position is selected according to Z. The position actually selected (faulty one) is informed in the specific cell in Tables 3 to 5. Note that this faulty behavior appears for a subset of configurations and is

similar to a stuck-at fault for some values of Z. For instance, an open in transistor t5 of tab. 3(b) (DLC2) or a short fault in transistor t1 of tab. 4(b) (DLC1) lead to a “Z stuck-at 1” behavior. This means that the QLUT output corresponds to the value stored in the second SRAM cell even though the value of Z is different from ‘1’. Finally, a “multiple drive” faulty behavior indicates that the injected fault leads to a situation where multiple transistors are conducting and affecting the output. In this case, signalized as ERROR in Tables 3, 4, and 5, the output provides a voltage level, which has no correspondence with a valid logic value, and hence any logic value can be read.

From the fault injection results, one can conclude that a functional fault model is adequate for the QLUT. Indeed, the “wrong selection” and “multiple drive” faulty behaviors can be considered as types of address decoder faults (one cell accessed from multiple addresses and multiple cells addressed simultaneously, respectively). The “wrong value” faulty behavior can be seen as a variant of the stuck-at fault, since it is related to a specific value in the SRAM memory.

Tab. 3 - Results for open faults

Z	p1	n1	p2	n2	p3	n3	p4	n4	p5	n5	p6	n6
0	3 (2.43)	0(0.491)	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok
1	Ok	Ok	3(2.23)	0(0.85)	3(2.23)	0(0.85)	Ok	Ok	Ok	Ok	Ok	Ok
2	Ok	Ok	Ok	Ok	Ok	Ok	3(2.22)	0(0.86)	3(2.24)	0 (0.85)	Ok	Ok
3	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	3 (2.21)	0 (0.9)

a) Results for open faults injected in the transistors of the transmission gates

	Z	t1	t2		t3	t4		t5	t6
DLC1	0	Ok	Z sta-1	DLC2	Ok	ERROR	DLC3	Ok	ERROR
	1	Z sta-0	Ok		Ok	Z sta-2		Ok	ERROR
	2	ERROR	Ok		Z sta-1	Ok		Ok	Z sta-3
	3	ERROR	Ok		ERROR	Ok		Z sta-2	Ok

b) Results for open faults injected in the transistors of the DLCs

	Z	t1	t2		t3	t4		t5	t6
INV1	0	ERROR	Ok	INV2	ERROR	Ok	INV3	ERROR	Ok
	1	Ok	ERROR		ERROR	Ok		ERROR	Ok
	2	Ok	ERROR		Ok	ERROR		3(2.21)	Ok
	3	Ok	ERROR		Ok	ERROR		Ok	0(0.8)

c) Results for open faults injected in the transistors of the inverters

Tab. 4 - Results for short faults

Z	p1	n1	p2	n2	p3	n3	p4	n4	p5	n5	p6	n6
0	Z sta-0	Z sta-0	ERROR	ERROR	Ok	Ok	ERROR	ERROR	Ok	Ok	Z sta-4	Z sta-4
1	Z sta-0	Z sta-0	Ok	Ok	Ok	Ok	ERROR	ERROR	Ok	Ok	Z sta-4	Z sta-4
2	Z sta-0	Z sta-0	Ok	Ok	ERROR	ERROR	Ok	Ok	Ok	Ok	Z sta-4	Z sta-4
3	Z sta-0	Z sta-0	Ok	Ok	ERROR	ERROR	Ok	Ok	ERROR	ERROR	Z sta-4	Z sta-4

a) Faults injected in the transistors of the transmission gates

	Z	t1	t2		t3	t4		t5	t6
DLC1	0	Z sta-1	Ok	DLC2	ERROR	Ok	DLC3	ERROR	Ok
	1	Ok	Z sta-0		ERROR	Ok		ERROR	Ok
	2	Ok	ERROR		Ok	Z sta-1		ERROR	Ok
	3	Ok	ERROR		Ok	ERROR		Ok	Z sta-2

b) Faults injected in the transistors of the DLCs

	Z	t1	t2		t3	t4		t5	t6
INV1	0	Ok	ERROR	INV2	Ok	ERROR	INV3	Ok	ERROR
	1	0 (0.8)	Ok		Ok	ERROR		Ok	ERROR
	2	ERROR	Ok		0(0.8/1.5)	Ok		Ok	ERROR
	3	ERROR	Ok		ERROR	Ok		0 (0.8)	Ok

c) Faults injected in the transistors of the inverters

Tab. 5 - Results for faults in the Vt signals of the DLCs

	Z	1->2	1->3		2->1	2->3		3->1	3->2
DLC1	0	Ok	Ok	DLC2	Ok	Ok	DLC3	Ok	Ok
	1	Z sta-0	Z sta-0		Z sta-2	Ok		ERROR	Ok
	2	Ok	ERROR		Ok	Z sta-1		Z sta-3	Z sta-3
	3	Ok	Ok		Ok	Ok		Ok	Ok

4. Resource-Aware Testing Application

One can think of a quaternary FPGA by replacing the traditional binary configurable logic blocks (CLBs) by its quaternary version QLUT presented in Section 2. Besides the distinct basic block, another important difference between the binary and the quaternary FPGA is the number of interconnection wires required to connect the QLUTs. Indeed, each QLUT can implement any 4-input function, which corresponds to two binary

CLBs. Thus, considering the same number of configurable blocks in each device and k binary CLBs to implement a given application, only $k/2$ QLUTs must be connected to implement the same function in the quaternary FPGA. In other words, a quaternary FPGA requires less interconnection wires and less CLBs to implement the same logic. Thus, the association of the QLUT with a programmable interconnection infrastructure is straightforward and the same regular organization used in a traditional FPGA can be assumed. To test a QLUT within such a structure, one must devise a strategy that ensures the correct test data delivering and recovering.

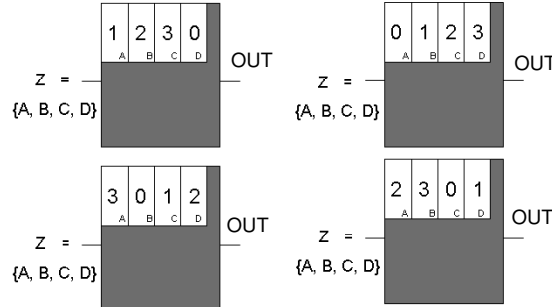


Fig. 4 – Test Configurations for the Complete QLUT Testing

Let us assume, without loss of generality, that the configuration of the QLUTs in the quaternary FPGA is also implemented through a single bitstream that configures both, the sequential (SRAM) and the logic (flip-flop and mux) parts of the QLUT. Similarly to the binary FPGA, this configuration time is the most time-consuming one since a long configuration register must be updated serially. To implement the test of each QLUT in this regular structure, we can adapt a successful FPGA test strategy proposed by Renovell et. al. in [7].

To be implemented the resource-aware technique needs just one input pin, one output pin and one wire connecting each QLUT. Then is necessary configuring every QLUT with one vector, however could be any vector showed in fig. 4. For example we will use the first vector: A=1, B=2, C=3 and D=0. When the value 0(A) is chooses by Z to the first QLUT the value 1 is returned. In this moment we could put this value to an output, but in really, we will put this value in the next QLUT. The next QLUT will return a 2, and then all the values will be linked. We know how many QLUTs we have in this path, at this time the value goes to output pin the ATE will compare the expected value with the real value identifying a possible fault. The linked proposed was based in Renovell [7], but remodeled to be applied in quaternary circuits. In binary results the test of a complete array of $m \times m$ LUT/RAM modules requires 4 test configurations independently of the size of the array and of the modules.

The time to apply the resource aware technique is four times the configuration time (to configure all the QLUT with the 4 vectors) added with four times the time to value by pass all the QLUTs and go to the path output to be compared in a ATE with the expected value for each configuration. Resource-aware test time: $4 \times \text{configuration time} + 4 \times \text{time to the value by pass all the QLUTs}(\text{depending of the number of I/O pins})$.

5. Timing-Aware Testing Technique

Despite its simplicity, the functional test approach is exhaustive and can be very time-consuming. There is a possibility to reduce this test time. Suppose that we have a FPGA with m rows by n columns ($m \times n$), how showed in fig. 5. If each set of vertical interconnections have more wires than the number of QLUTs in the column more one, the timing-aware technique could be implemented. For a better comprehension the number of vertical wires is called k . The solution could be applied if k is greater than $n+1$ ($k > n+1$). Another important limitation is the output pin number is necessary has at least one output pin for each tested QLUT.

This restriction is necessary because for each QLUT we need to send the output directly to an output FPGA pin because we cannot link all the QLUTs with the minimum vectors number. And the number one in the equation is the input for each QLUT in the same column, but the input value is the same for all QLUT blocks then we could share the same wire.

On the other hand, by analyzing the results presented in tab. 3 one can conclude that only all faults can be detected using as much as two configurations per LUT. Indeed, the configuration A= 3, B = 0, C = 3, D = 0 is essential to detect an open in transistors m1, m4, m7, and m12. A second configuration with A= 0, B = 3, C = 0, D = 3 is also necessary to detect an open in transistors m2, m3, m10, and m11. By further analyzing the results one can observe that these two configurations can sensitize all the remaining faults, including shorts, opens and V_t variations in all transistors. Then, the two configurations shown in fig. 5 are necessary and sufficient to test the quaternary LUT depicted in fig. 3 for the fault model defined in Section 3. For each configuration, all possible values for Z must be applied.

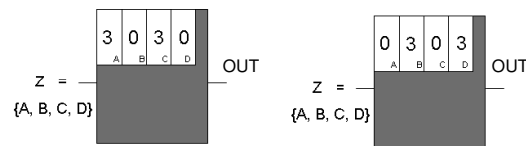


Fig. 5 – Non-Exhaustive Technique configuration per Quaternary Look-Up Table

The time to apply the timing-aware technique is two times the configuration time (to configure all the QLUT with the 2 vectors) added with two times the time to apply the vector in the slowest QLUT comparing all columns. Timing-aware test time: $2 \cdot T_{\text{Config.}} + 2 \cdot \text{time to apply the vector in the slowest LUT comparing all columns.}$ (depending of the number of I/O pins).

6. Conclusion

We discussed in this paper the test of an array of quaternary look-up tables. After evaluating the testing requirements of such a new design paradigm, we devised a resource-and-time-aware test strategy. Preliminary results show that these devices have competitive test costs in terms of time and hardware resources. Considering the other advantages of this design paradigm we conclude that quaternary FPGAs can be a viable solution to deal with current CMOS technology limitations. Current work includes the evaluation of transition and cross-coupling faults for the SRAM as well as the test of the interconnections. However it would be interesting to explore the testing technique under multiple-fault model too.

7. References

- [1] S.D. Brown, R.J. Francis, J. Rose, and S.G. Vranesic, Field Programmable Gate Arrays, Kluwer Academic Publishers, 1992.
- [2] S.M. Trimberger (Ed.), Field-Programmable Gate Array Technology, Kluwer Academic Publishers, 1994. 18. M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian, "Testing the Configurable Logic of RAM-Based FPGA," Proc. IEEE Int. Conf. on Design, Automation and Test in Europe, Paris, France, Feb. 1998, pp. 82–88.
- [3] R. C. G. da Silva, H. Boudinov and L. Carro "CMOS voltage-mode quaternary look-up tables for multi-valued FPGAs". A Novel Voltage-Mode CMOS Quaternary Logic Design. IEEE Transactions on Electron Devices, v. 53, n. 6, p. 1480-1483, 2006.
- [4] W. J. Dally. "Computer architecture is all about interconnect," the 8th International Symposium on High-Performance Computer Architecture, Cambridge, Massachusetts, February 2002.
- [5] Z. Zilic; Z.G. Vranesic. "Multiple-Valued Logic in FPGAs". In: MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS, 36., 1993. Proceedings... [S.l.]: IEEE, 1993. p.1553-1556.
- [6] J. P. DESCHAMPS; J. G. BIOL.; G. D. SUTTER, Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems. Hoboken, NJ: John Wiley & Sons, 2006.
- [7] M. Renovell, J. M. Portal, J. Figueras, Y. Zorian: SRAM-based FPGA's: testing the LUT/RAM modules. ITC 1998: 1102-1111.
- [8] Michel Renovell, Jean Michel Portal, Joan Figueras, Yervant Zorian: Minimizing the Number of Test Configurations for Different FPGA Families. Asian Test Symposium 1999: 363-368.

Model for S.E.T propagation in CMOS Quaternary Logic

Valter Ferreira, Gilson Wirth, Altamiro Susin
{vaferreira,wirth}@inf.ufrgs.br, autamiro.susin@ufrgs.br

Universidade Federal do Rio Grande do Sul – UFRGS

Abstract

This paper describes the investigation of a Model of S.E.T propagation in quaternary logic circuit. The CMOS quaternary logic used was developed in UFRGS and uses transistors with different threshold voltage and three levels of power supply. The propagation model was earlier developed to be used in a binary logic chain. This work intends to verify the applicability of this model to predict the values of transient pulse width through a mixed CMOS quaternary logic chain. The electric simulations with Spice Opus Simulator were made in three quaternary chains and one quaternary generic circuit. The research result shows that the level and width of transient pulse were degraded by electrical masking during propagation process. The measured and calculated pulse width values were compared resulting in a reasonable proximity between them. The analytical model of S.E.T propagation was considered valid to be utilized in a quaternary circuit.

1. Introduction

The constant growth of the semiconductor industry has led to a great improvement in the fabrication of circuits with smaller and faster transistors. It allows the integration of billions of transistors in the same chip, giving the designer the possibility to implement more functions in the same device. However, the technology improvement is bringing an increased concern about the reliability of these new circuits because the new generations of transistors are more sensible to strike with ionizing particle and its effects as Single Event Transient (SET).

On the other hand the interest in electronic circuits that employ more than two discrete levels of signal has increased. In this way a novel quaternary voltage-mode CMOS logic using transistors with different threshold voltage and three levels of power supply can be a good response to performance's necessity. Thus this work intends connect this two approaches and to investigate how the charge particle hit affect a CMOS quaternary combinational circuit.

2. CMOS quaternary logic employed

The voltage-mode quaternary logic CMOS circuits used in this work was proposed by [1] and operates with four voltage levels, corresponding to 0V and three power supply lines of 1, 2, and 3V. The quaternary logic circuits are designed using TSMC 0.18 μ m technology with three different V_t form. Tab. 1 shows transistors and V_t values relative to the transistor source–bulk voltage.

Tab.1 – Transistor V_t values related to V_s

Transistor	T1	T2	T3	T4	T5	T6
V_t	-2.2	2.2	1.8	0.2	-0.2	-1.8
type	PMOS	NMOS	PMOS	NMOS	PMOS	NMOS

The quaternary inverter circuit and its truth table are presented in Fig.1. The inverter topology consists of three PMOS and three NMOS transistors and was designed to ensure that only one of the voltage levels is shorted to the output for a certain input voltage. If the input value is 0V, transistor T1 is turned on, driving the output to 3V, whereas T2, T4, and T6 are turned off, cutting their main paths. When the input is set to 1V, T1 is turned off, whereas T6 is turned on, driving the output to 2V. When the input is set to be 2V, T5 is turned off, whereas T4 is turned on, hence driving the output to 1V. T2 sinks the output to zero only when the input goes to 3V, turning off T3.

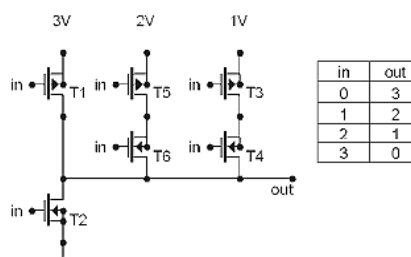


Fig. 1 – Quaternary inverter circuit and its truth table.

In quaternary logic, the AND/NAND logic gates are replaced by MIN/NMIN gates. The MIN operation sets the output of the MIN circuit to be the lowest value of several inputs. The implementation of an NMIN circuit with two quaternary inputs and its truth table are shown in Fig.2(a). This circuit is based on the inverter circuit in Fig.1 and a common binary nand circuit. An input set to zero will produce the output of 3V regardless of the other input voltage level. Then MOS transistors disposed in series make the paths to 1V, 2V, and ground to be opened only of both transistors. When both inputs are equal to or higher than the V_t PMOS transistors are responsible to close the path when both inputs values. OR and NOR logic gates also have no meaning in quaternary logic and these gates are replaced by MAX and NMAX gates. The MAX gate is a circuit of multiple inputs which sets the output in the higher value of all entries. The Fig.2(b) shows an NMAX circuit with two quaternary inputs and its truth table. The highest value of all the inputs sets the output. When both inputs are in 0V, both T1 transistors are turned on, driving the output to 3V. One of the inputs set on a higher value is enough to close the path from the output to the 3V power supply and too penal other paths. Any input in 1 V opens one of the T6 transistors and turns off one of the T1 transistors, placing the output at 2V. In the same way, any 2V inputs will turn on any T4 transistor while at the same time turning T1 and T5 off. Any 3V input turns on a T2 transistor and turns off the T1, T5, and T3 transistors.

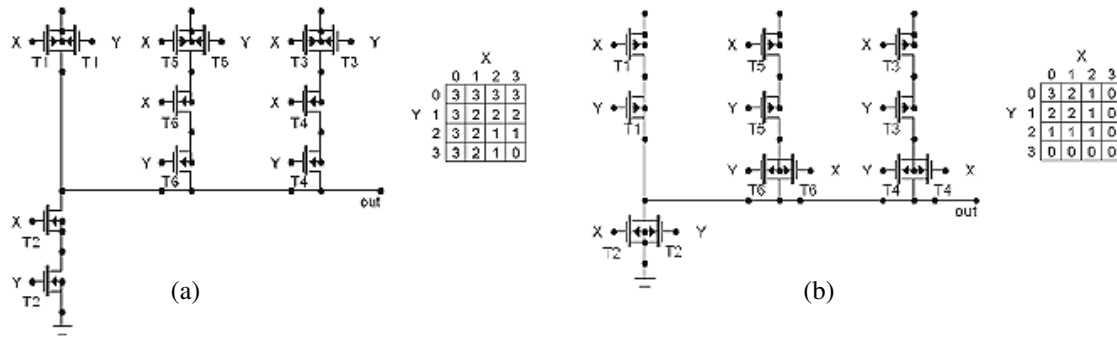


Fig. 2 – (a) NMAX circuit and its truth table; (b) NMIN circuit and its truth table.

3. S.E.T mechanism and modeling

When an ionizing particle strikes a p-n junction, the generate column of carriers disturbs the electric field of the depletion region. The electric field spreads down along the track into regions that originally were field-free and accomplishes the charge collection mechanism known as the funneling effect, producing a sharp peak in the disturbing current of logic circuit sensitive node [2] presented in Fig. 3(a). When the charge collected exceeds the minimum charge, named Critical Charge or Q_c , a Single Event Transient (SET) occurs and the logic value on sensitive node is changed. In the same way if the transient pulse had an enough amplitude and time, the pulse can propagate for others stages of circuit and modify the computing result [3]. The charge collected mechanism can be modeling by a current pulse which is describe by double exponential, see Fig. 3(b) where I_0 is current peak, τ_α is the time constant stabilization and τ_β is the time constant to generate electron-holes pairs [1]. Therefore, amplitude and time duration of pulse are essential parameters to evaluate the circuit sensitivity related to SETs [4]. The model used to realize electrical simulations can be visualized in Fig. 3(c) which shows an exponential current source put in the node hit by energizing particle [1].

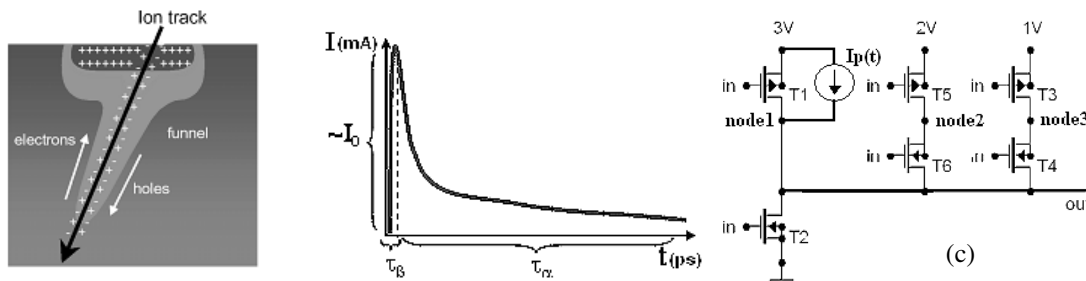


Fig. 3 – (a) charge collection mechanism, (b) current pulse model, (c) electric simulation model.

4. Model of S.E.T propagation in quaternary logic circuit

The model presented in this section was proposed by [5] to study the SET propagation through the binary logic chain. The main contribution of this work was to verify the model's adequacy to describe the SET propagation through the **quaternary** logic chain.

The model is divided in to two regions, according to the relationship between τ_n (width of transient pulse at the n -th logic stage) and the gate delay t_p . Where t_p will be equal to t_{pHL} to $0 \rightarrow 1 \rightarrow 0$ transition at the n -th node, or equal to the t_{pLH} for a $1 \rightarrow 0 \rightarrow 1$ transition. The first region represents the situation where the transient pulse is filtered out. The model evaluates the transient pulse width which is the time to node's voltage change greater than $VDD/2$. The condition for a transient pulse to be propagated to next logic state is related to ratio τ_n / t_p , which must be equal to or greater than 2. If the transient pulses width (τ_n) is smaller than 2 times t_p , the voltage at the next node changes less than $VDD/2$ and it means that the transient is filtered out. So the transient pulse does not propagate to the $(n+1)$ th stage. Thus, the equation for this region is:

$$\text{if } (\tau_n < 2t_p) \rightarrow \tau_n + I = 0 \quad \text{Eq.1}$$

The second region represents the situation which the transient pulse is degraded by electrical masking when during the propagation through the logical gates chain. This fact occurs if the input pulse width (τ_n) is greater than 2 times t_p . The pulse width value in $(n+1)$ th stage can be calculated by following equation:

$$\tau_n + I = \tau_n (r_{ip} \cdot K - e^{(r_{ip} \cdot m / t_p)})^{1/2} \quad \text{Eq.2}$$

where:

' τ_n ' is the pulse width in (n) th stage;

' $\tau_n + I$ ' is the pulse width in $(n+1)$ th stage;

' r_{ip} ' is the result of the following relationship:

$$r_{ip} = (t_{pHL} / t_{pLH})^{1/2} \text{ to } t_{pHL} > t_{pLH} \quad \text{Eq.3}$$

or

$$r_{ip} = (t_{pLH} / t_{pHL})^{1/2} \text{ to } t_{pLH} > t_{pHL} \quad \text{Eq.4}$$

' K ' is a fitting parameter which depends on the interest technology.

The first step of investigation was to simulate a SET at three different quaternary chains composed by 25 gates each. The first chain was composed only by inverters, the second chain only by NMIN gates and the third chain only by NMAX gates. The electrical simulations using Spice Opus software were made to each chain by application of a transient pulse ($I_0 = 0.8\text{mA}$, $\tau_p = 2\text{ps}$ e $\tau_a = 300\text{ps}$) in the input of first logic gate. So only the pulse widths, presented by simulations, with wave form similar to a natural transition were written in the tables 2, 3 and 4. The Fig. 4 presents, for example, (a) the NMIN chain used at the simulation and (b) the pulse degradation wave form.

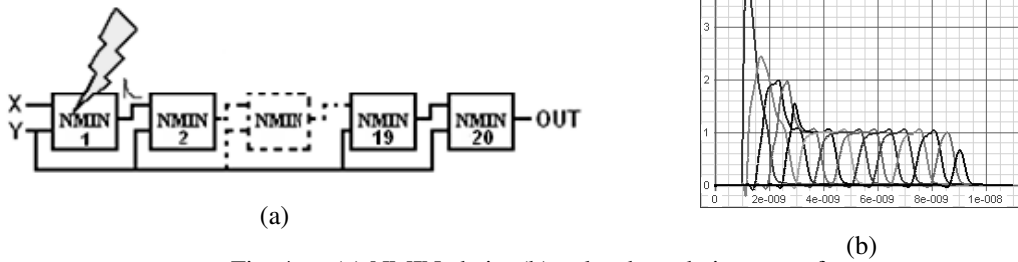


Fig. 4 – (a) NMIN chain; (b) pulse degradation wave form.

The second step was to apply the analytical SET propagation model using the equation Eq.2 from second region of model. Thus the Eq.2 received some data how the width of first pulse similar to a natural transition (τ_n), the respective gate delays how t_{pHL} , t_{pLH} and t_p and finally the K parameter value which stays between 0.8 to 0.9. After that the $\tau_n + I$ value was calculated, written on the table and put at the Eq.2 again to a next stage calculus.

The results of calculation were written in the tables 2, 3 and 4, which presents the measured and calculated transient pulse width.

Tab.2 – pulse width (ps) in inverter chain

	Out13	Out14	Out15	Out16	Out17	Out18	Out19
Spice Opus	1248	1127	1028	874	772	602	486
Model (K=0,80)	1248	1128	1010	892	771	641	494

Tab.3 – pulse width (ps) in NMIN chain

	Out10	Out11	Out12	Out13	Out14	Out15	Out16	Out17
Spice Opus	1739	1680	1523	1454	1294	1223	1055	981
Model (K=0,84)	1753	1649	1546	1442	1335	1224	1106	978

Tab.4 – pulse width (ps) in NMAX chain

	Out14	Out15	Out16	Out17	Out18	Out19	Out20	Out21
Spice Opus	1671	1586	1507	1419	1337	1251	1160	1075
Model (K=0,92)	1674	1595	1516	1437	1357	1275	1190	1101

Finally, the first and second steps were applied to investigate the model's applicability into mixed quaternary chain using a generic combinational quaternary logic circuit. The Fig. 5 shows (a) the generic circuit with the tracing line representing the transient pulse path and (b) the output wave form from each gate in the pulse path. The calculated and measured pulse width values are shown in the table 5.

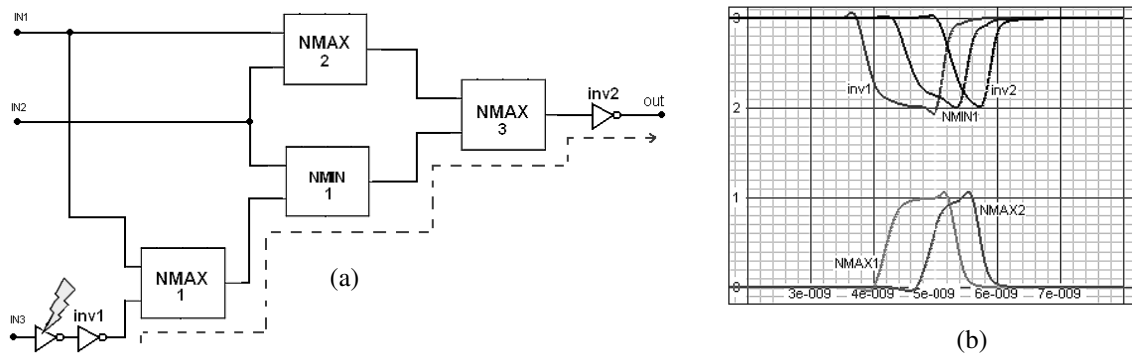


Fig. 5 – (a) Quaternary generic circuit; (b) pulse degradation wave form.

Tab.5 – pulse width (ps) in generic circuit

	inv1	NMAX 1	NMIN 1	NMAX 3	inv2
Spice Opus	1237	1135	946	797	621
Model (K=0,83)	1231	1083	946	784	672

5. Conclusion

The applicability of model for SET propagation in CMOS quaternary logic was studied through electric simulations. The result shows that the transient pulse level and width are degraded by electrical masking during the pulse propagation through quaternary logic gates.

The simulations using only inverters, NMIN and NMAX gate chains presented different K values to each one. The K value used in quaternary generic circuit was consequence of average calculus with values from tab.2, 3 and 4. However, the better value of K to TSMC 0.18um technology, considering the quaternary logic used, was 0.83 because the calculus adjustment. The same analysis utilizing other technologies will make to improve the model's reliability.

The analytical model showed an acceptable proximity between the measured and calculated values of pulse width through the mixed quaternary logic chain. This result can be used to create a computational model to support the circuit test.

6. References

- [1] R. C. G. da Silva, H. Boudinov and L. Carro, "A Novel Voltage-Mode CMOS Quaternary Logic Design," IEEE Transactions on Electron Devices, vol.53, no.6, june2006. pp.1480-1483.
- [2] T. Heijmen, "Radiation-induced soft errors in digital circuits", Philips Electronics, Nederland, 2002.
- [3] G. I. Wirth, M. G. Vieira, E. H. Neto and F. L. Kastensmidt "Generation and Propagation of Single Event Transients in CMOS Circuits", DDECS06, April 2006.
- [4] S. V. Walstra and C. Dai, "Circuit-Level Modeling of Soft Errors in Integrate Circuits," IEEE Transactions on Electron Devices, vol.5, no.3, September 2005. pp.358-362.
- [5] G. I. Wirth, I. Ribeiro and F. L. Kastensmidt "Single Event Transients in Logic Circuits Load and Propagation Induced Pulse Broadening", IEEE Transactions on Nuclear Science, VOL.55, NO.6, December2008.

Fault Tolerant NoCs Interconnections Using Encoding and Retransmission

Matheus P. Braga, Érika Cota, Marcelo Lubaszewski
{matheus.braga, erika, luba}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Programa de Pós-Graduação em Computação
Porto Alegre, RS, Brazil

Abstract

The state-of-art design of Systems-on-Chip (SoCs) is based on reuse of blocks previously designed and verified, named cores or IP (Intellectual Property) blocks. In nowadays SoCs are designed using dozens or hundreds of cores, and for this reason, it's mandatory a communication infrastructure that provides high performance, scalability, reusability and parallelism. For systems with dozens of cores, architectures using point-to-point dedicated channels or based on a multipoint structure (bus infrastructure, for example). However, it's known that SoCs with hundred of cores cannot meet performance requirement of many applications using bus-based communication infrastructure. A solution well known for this communication bottleneck is the NoC (Network-on-Chip) architecture. NoCs connects the IP cores of the chip, using point-to-point channels and switches, like a traditional computer network. Fault Tolerant SoCs provides more reliability and, for this reason, mechanism for tolerating transient or permanent faults must be taking into account. Then, this paper proposes three fault tolerance techniques applied to data and flow-control communication channels of a NoC. One technique is based on Hamming Code encoding and the others use mechanism of data splitting and retransmission.

1. Introduction

The move towards submicron technologies allows design of very complex applications on a single chip by using a system based on pre-designed and verified IP cores [1]. These systems, well known as Systems-on-Chip (SoCs), include processors, controllers and memory in a single chip [2]. SoCs must meet requirements as high performance, parallelism, reusability and scalability, for example. The increases of transistors integration will allow design of SoCs with hundred of cores in a single chip with up to four billions of transistors [3].

Therefore, it's necessary a communication architecture that provides these requirements. Bus-based communication architecture have become a solution for SoCs with dozen of cores. However, as the number of IP cores in a SoC increases, busses may prevent SoCs to meet the requirements of many applications. System with intensive parallel communication requirements, for example, busses may not provide the required bandwidth, latency and power consumption [4].

A new solution for this bottleneck is the use of Networks-on-Chip (NoCs). An NoC is an embedded switching network to connect all the IP cores in SoCs [5]. The IP cores are connected through the NoC infrastructure using point-to-point communication channels and embedded switches, like a traditional computer network. NoCs must provide reusability, scalability and parallel communication between the modules [6]. The fig. 1 shows an NoC architecture known as grid 2-D, where the box represents the switches and the links between them are the communication channels.

However, the reductions on transistors dimensions pose new challenges to circuit design and manufacture. As the dimensions shrink dramatically, it's increasingly difficult to control variances of physical parameters in the manufacture process. This results in transient or permanent faults and decreased yield which increases the costs per functioning chip [7]. To maintain the yield at an acceptable level, it's taking to account some faults in a chip, which can be tolerated with dedicated circuit structures. Nevertheless, different types of faults can be treat using an amount of fault tolerance methods, and then it's necessary a sophisticated combination of them to meet requirements of reliability [8].

Despite of NoCs switch dimensions are very small related to core dimensions, the NoC communication channels may presents great extension. Fig. 2 shows an example of a NoC designed in the boundary of the chip, connecting cores located among the switches. In this case, the interconnections are as great as the size of the cores. Then, long channels length implies in higher wire capacitances and network latency. Furthermore, SoCs using long communication channels presents high fault susceptibility.

To achieve the requirements of reliability in data transmission, fault tolerance methods applied to NoCs interconnections are mandatory. These techniques use redundancy and/or data encoding for error detection

and/or correction. In this paper we focus on design of fault tolerant communication links in NoCs architecture. We take into account both transient and permanent faults that are originated mainly from manufacture process or radiation. We propose three methods to be applied to communication channels: a 1-bit parity for data transmission; a 2-bits parity technique and a Hamming encoding transmitted data technique. The two first mechanisms use split data transmission and retransmission of redundancy bits. We also propose using fault tolerance techniques in flow-control signals to guarantee the correctness of the communication protocol in data transmission.

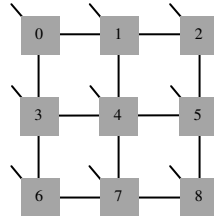


Fig. 1 – An NoC grid 2-D topology

2. Fault Tolerant Data Transmission

As we can note in the fig. 2, NoCs interconnections length may be as great as the dimension of IP modules. This design implies some challenges like wire capacities and network latency. Another problem of this type of design is the high probability of transient and permanent faults. Fault tolerance approaches are necessary to treat with these types of errors and consequently increase the NoC reliability in order to meet the performance requirements.

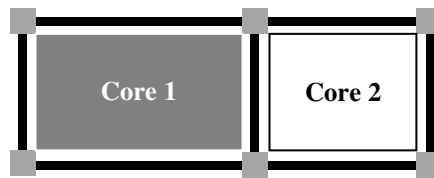


Fig. 2 – A core boundary NoC strategy

Design of fault tolerant NoCs must consider the possible fault scenario. The faults can be divided into three classes: permanent, intermittent, and transient faults [7]. Most of the failures (up to 80%) are caused by transient faults [3], while permanent faults represents one fifth of the others faults. Thus, the fault tolerance method must consider not only transient faults but also permanent ones. For detection of temporary and permanent faults, Hamming Code is widely used [9]. This encoding approach is capable to detect and correct one single error. Hamming can detect two single faults and with one additional check bit it can be extend to three.

Fault tolerance methods besides coding include for instance triple modular redundancy (TMR) and the use of additional bits, as parity bits, together with error detection [8]. So, in this paper we propose three fault tolerance techniques applied to data channels of NoCs interconnections. We present an approach based on Hamming encoding and another two techniques based on parity bits. In the proposed Hamming encoding approach the data is encoding in the sender and decoding in the receiver through two dedicated circuits. In the receiver, if the data is wrong, this circuit can flip the faulty bit, considering only one single fault.

The great advantage using Hamming encoding is the only one transmission even in the presence of a fault due to the correction circuit in the receiver switch that diagnoses and flips the faulty bit. In other hand, this approach implies in an overhead of area related to the circuits responsible to encoding and decoding in each port of the switch.

2.1. Single Parity Approach

The technique uses one parity bit calculated in the output channel of the switch at the last moment before the data transmission by a dedicated circuit (*output_data_parity*). The additional bit can detect one single fault at the data bits and the delimiter packet signals (bop – begin-of-packet, eop – end-of-packet). For this approach the interconnections channel must have a spare wire where this additional bit must be sent together with the data bits.

When the receiver switch receives the data and parity, the parity is over again calculated by a dedicated circuit (*input_data_parity*) at the switch input channel and compared with that one send by the neighbor switch. If they are the same, the data is correct and the receiver signalize that data is correct and the transmission is

finished. In other case, the receiver sends an error signal to the sender informing the presence of a transmission failure and the data must be retransmitted. The retransmission process occurs in two data transmissions:

1. The sender divides the data bits on the half and calculates the parity only of these bits. Then, the less significant half is duplicated and sent with its parity bit.
2. In the other way of the channel, the switch receives the duplicated less significant half, calculates the parity of one copy and compares with the sent parity. If the check is correct the receiver accepts that half or, if not, the receiver stores the other copy of the less significant bits.
3. The sender now calculates the parity of the more significant bits and sends this parity bit together with two copies of the more significant bits.
4. For finish the receiver chooses the correct copy of the more significant bits according to the same algorithm used for the less significant bits.

Notes that this approach avoids single transient faults or permanent faults due to the duplicated data transmission and, for this reasons, at each transmission, considering only single faults, there is always a fault-free copy. Another vantage of this method is the only two wires overhead (a parity bit and an error bit), but this technique implies in two data retransmission at each occurred error.

The fig. 3 shows a scheme of an interconnection using this proposed method, where *data_0to1*, *parity_0to1* and *error_1to0* represent the data channel from the router 0 to the router 1. The others links represents the channel in the other way.

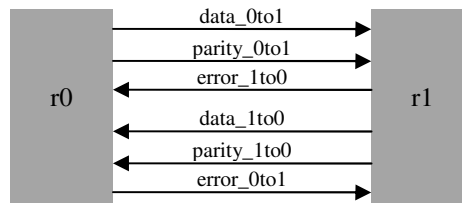


Fig. 3 – Block scheme of the 1-bit data parity

2.2. Single Retransmission Approach

The technique described above presents as the main drawback the necessity of two data retransmissions. Then, this second approach is based on the solution of this drawback reducing the number of retransmissions to one. To reduce the number of retransmissions to one, we propose the addition of one more parity bit and error signal. This method uses the division of the data bits in two halves, as the method of one parity bit. This addition provides the technique to reduce to one retransmission, locating the faulty data half due to dedicated parity and error for each half.

The fig. 4 illustrates the block scheme of this approach. The nomenclature used in the picture is according to the one used in the fig. 3. The *LSB* and *MSB* suffix indicates the wires related to the less and more significant bits, respectively. Below we describe the functionality of this approach.

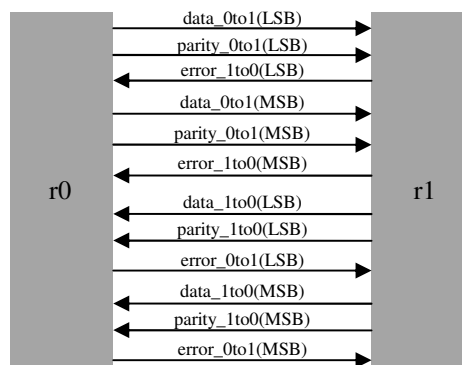


Fig. 4 – Block scheme of the 2-bit data parity

In this approach, the sender calculates two parities: one for the less significant half and another for the more significant half, and sends these bits together with the data bits through the dedicated channels (*parity (LSB)* and *parity (MSB)*, respectively). The receiver checks the sent parities, calculating the parities of the received bits. If the transmitted data is correct the data can be processed in the receiver switch. Otherwise, the receiver signalize the error through the error signals according to its position (*error (LSB)* if the error is in the less significant half or *error (MSB)* in other case).

The sender receives the error signal concerning the faulty half and then, retransmitted the faulty half. To retransmit the requested half, the sender calculates its parity and sends that through the channel *parity... (LSB)*.

The data is doubled in the case of a new error in the retransmission (due to a new transient fault or a permanent fault). The receiver calculates the parity of one half, compares to the parity sent and chooses the correct half.

3. Results

The results presented in this paper concern the wire overhead and delay of data transmission. For the three tables showed in this section considerer NoC_1P, NoC_2P and NoC_HC the single parity approach, single retransmission technique and the hamming encoding method, respectively. In tab. 1 we present the results of wire overhead according to the $n + 2$ bits, where n is the number of data bits and the two bits represents the *bop* and *eop* bits. We can note the increase of number of additional wires in the Hamming encoding technique while in the parity approaches this number is constant.

Tab. 1 – Wire overhead comparison

# of Bits	NoC_1P	NoC_2P	NoC_HC
18 (16+2)	7	10	9
34 (32 + 2)	7	10	10
66 (64 + 2)	7	10	11
130 (128 + 2)	7	10	12

The proposed technique was described in VHDL and synthesized in FPGA. The tab. 2 shows the delay results of the three approaches and the NoC without fault tolerance (*NoC_SP*). Tab. 2 also shows the delay of a data transmission in the occurrence of a fault. Notes the *NoC_1P* approach presents the biggest delay in the both cases, while *NoC_HC* is more effective in a presence of an error. In other hand the *NoC_2P* presents the best results in a faulty-free transmission and in a presence of a fault, its delay is around 41% related to the *NoC_HC*.

Tab. 2 – Results of delay

Delay	NoC_SP	NoC_1P	NoC_2P	NoC_HC
Fault-free transmission (ns)	5.718	9.213	7.836	11.09
Faulty transmission (ns)	5.718	27.639	15.672	11.09

4. Conclusions and Next Steps

This paper proposes fault tolerance techniques for NoCs interconnections. We propose three approaches to be applied to data channels of NoCs interconnections: two methods using additional bits for parity and another using Hamming encoding. Analyzing the results, the two parity bits presents the best scenario due to the constant wire overhead and the best average delay considering a faulty-free transmission and a faulty transmission. As future works explore the results of delay through fault injection campaigns to evaluate more precisely the results of delay. Propose another methods and combination of the techniques proposed on this paper is another step of this work.

5. References

- [1] E. Cota, F. L. Kastensmidt, M. Cassel, P. Meirelles, A. Amory, and M. Lubaszewski, "Redefining and Testing Interconnect Faults in Mesh NoCs," *Proc. International Test Conference (ITC'07)*, 2007.
- [2] C. A. Zefferino, "Redes-em-chip: Arquiteturas e Modelos para Avaliação de Área e Desempenho," doctoral thesis, PPGC, UFRGS, Porto Alegre, 2003.
- [3] G. De Micheli, and L. Benini, *Networks on Chips*, Morgan Kauffman Publishers, San Francisco, Calif, USA, 2006.
- [4] F. G. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an Infrastructure for Low Area Overhead Packet-Switching Network on Chip," in *Integration, the VLSI Journal*, v. 28-1, 2004, pp. 69 – 93.
- [5] P. Guerrier, and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," *Proc. of Design, Automation and Test in Europe, DATE*, 2000, pp. 250 – 256.
- [6] C. Zefferino, and A. Susin, "SoCIN: a parametric and scalable network-on-chip," *Proc. of 16 Symposium on Integrated Circuits and Systems Design, SBCCI'03*, 2003, PP. 169 – 174.
- [7] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, v. 23, n. 4, PP. 14 – 19, 2003.
- [8] T. Lehtonen, P. Liljeberg, and J. Plosila, "Online Reconfigurable Self-Timed Links for Fault Tolerant NoC," *VLSI Design*, 2007, p. 13.
- [9] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 24, n. 6, pp. 818 – 831, 2005.



Video Processing

A Quality and Complexity Evaluation of the Motion Estimation with Quarter Pixel Accuracy

¹Leandro Rosa, ¹Sergio Bampi, ²Luciano Agostini
{leandrozanetti.rosa,bampi}@inf.ufrgs.br, agostini@ufpel.edu.br

¹Universidade Federal do Rio grande do Sul

²Universidade Federal de Pelotas

Abstract

This work presents an algorithmic investigation of motion estimation (ME) with quarter pixel accuracy in video compression, targeting a future hardware design for HDTV (High Definition Television). The Quarter Pixel is used with Full Search and Diamond Search algorithms. The algorithms were described in C language and uses SAD as distortion criterion. For each algorithm, were evaluated four different search area sizes: 46x46, 80x80, 144x144 and 208x208 samples. The algorithms were applied to ten video sequences and the average results were used to evaluate the algorithms performance in terms of quality and computational complexity. The results were compared with the integer accuracy versions of the Full Search and Diamond Search algorithms. This comparison shows a large advantage of the use of quarter sample accuracy when the quality is evaluated. This expressive gain in quality causes an impact in terms of computational complexity.

1. Introduction

The compression of digital videos is a very important task. The industry has a very high interest in digital video coders/decoders, once digital videos are present in many current applications, such as cell-phones, digital television, DVD players, digital cameras and a lot of other applications. This important position of video coding in current technology development has boosted the creation of diverse standards for video coding. Without the use of video coding, the processing of digital videos is impracticable, due to the very high amount of data necessary to represent the video. Many video compression standards were developed to reduce this problem. The currently most used standard is MPEG-2 [1] and the latest and more efficient standard is H.264/AVC [2]. These standards group a set of algorithms and techniques that can reduce drastically the amount of data that is necessary to represent digital videos.

A modern video coder is formed by eight main operations, as shown in fig. 1: motion estimation, motion compensation, intra-frame prediction, forward and inverse transforms (T and T^{-1}), forward and inverse quantization (Q and Q^{-1}) and entropy coding. This work focuses in the motion estimation, which is highlighted in fig. 1.

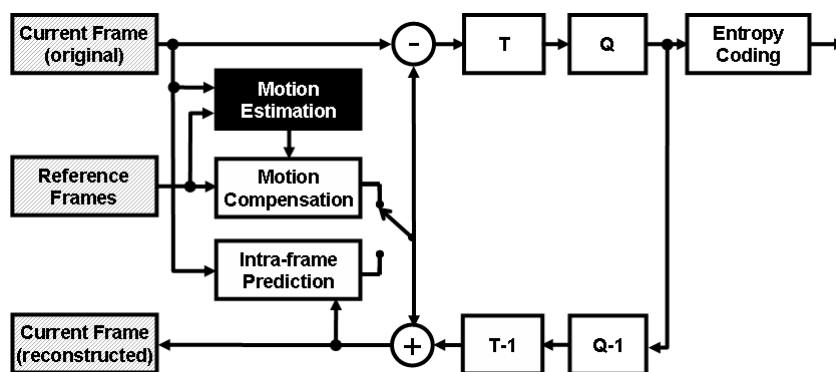


Fig. 1 – Block diagram of modern video coder

This paper presents an algorithmic investigation targeting motion estimation with quarter pixel accuracy in video compression. These algorithms used the Sum of Absolute Differences (SAD) [3] as distortion criterion and were evaluated with four different search area sizes, 46x46, 80x80, 144x144 and 208x208.

This paper is organized as follow: Section 2 presents some basic concepts about motion estimation. Section 3 presents the search algorithms and fourth section shows the results. Section 5 presents the conclusions.

2. Motion Estimation

Motion estimation (ME) operation tries to reduce the temporal redundancy between neighboring frames [3]. One or more frames which were already processed are used as reference frames. The current frame and the reference frame are divided in blocks to allow the motion estimation and in this paper were used blocks with 16x16 samples. The idea is to substitute each block of the current frame by one block of the reference frame, reducing the temporal redundancy. The best similarity between each block of the current frame and the blocks of the reference frame is selected. This selection is done through a determinate search algorithm and the similarity is defined through some distortion criterion [3]. This search is restricted to a specific area in the reference frame, called search area. When the best similarity is found, then a motion vector (MV) is generated to indicate the position of this block inside the reference frame. These steps are repeated for all blocks of the current frame.

An important feature of H.264/AVC, that is the latest one standard, is that it provides an accuracy of 1/4 pixel (quarter pixel) for MV. Typically, the movements that happen between frames are not restricted in integer pixel. So, if they are used only integer values for MV, usually you cannot find good matching. Therefore, the standard provides use of motion vectors with fractional values of 1/2 pixel and 1/4 pixel [4]. Fig. 2 presents one example for fractional MV

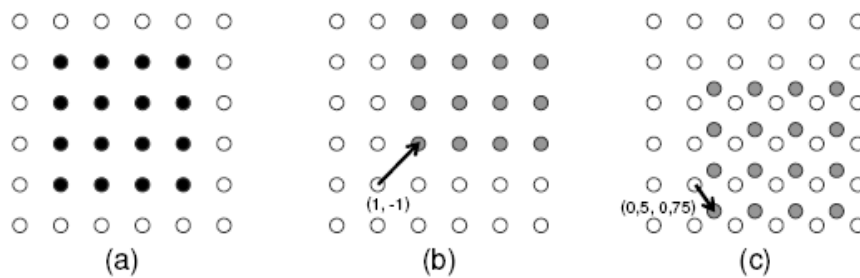


Fig. 2 – ME with accuracy of quarter pixel [4]

Fig. 2(a) shows a 4x4 block of current frame (black dots) that will be predicted. If the horizontal and vertical components of MV are integers, then the samples necessary for matching are present in the reference frame. As an example, fig. 2 (b) presents, a possible matching of the block in current reference frame (gray points), with MV uses only integer values. In this case, MV is (-1, 1). If one or both components of MV have fractional values, then predicted samples are generated from the interpolation (see section 3). Fig. 2 (c) presents a matching with MV using two fractional values. For this example, MV is (0.5, 0.75).

3. Algorithms

The search algorithm defines how the search for the best matching will be done in the search area. The search algorithm choice has a direct impact in the MV quality and in the ME performance.

In [5] is present an algorithmic investigation of ME. Based in this results two algorithms were selected for this paper: Full Search (FS) and Diamond Search (DS).

The Full Search algorithm searches for the best match testing every possible position within the search area of reference frame. For each test match, it calculates the block SAD. When all candidate blocks have been evaluated, the block that provides the lowest SAD will be chosen. After the choice, it generated a MV concerning the displacement of the current block that showed highest similarity within the search area of the reference frame. It will always find the best results [6].

The Diamond Search algorithm uses two diamond-shaped patterns, one for the first steps (LDSP - Large Diamond Search Pattern) and the other for the final refinement (SDSP - Small Diamond Search Pattern). At the first LDSP pattern compare nine positions, and while the central position is not the one with the lowest SAD, a new center is defined and the process is repeated. After finding the center with the lowest SAD, a refinement is done (with SDSP pattern) by testing four new positions forming the second diamond. The position which has the lowest SAD is chosen and the MV is generated [3].

In this paper, first is used the DS algorithm for choose the best MV and after it is interpolated to generate quarter pixel values. The interpolation is performed in two steps and the first interpolation is a 1/2 pixel using FIR filter with six taps (1/32, -5/32, 5/8, 5/8, -5/32, 1/32), as described in fig. 3.

In fig. 3, integer positions are represented by uppercase and 1/2 pixels for lowercase letters. As an example, the position 'b' is calculated as indicated in (1) and position 'j' in (2).

$$b = (E - 5 \times F + 20 \times G + 20 \times H - 5 \times I + J) / 32 \quad (1)$$

$$j = (aa - 5 \times bb + 20 \times b + 20 \times s - 5 \times gg + hh) / 32 \quad (2)$$

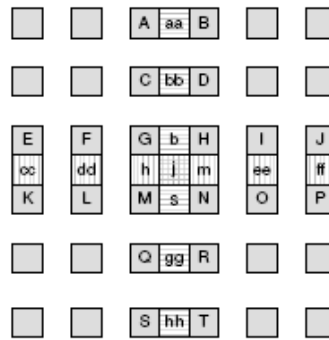


Fig. 3 – Interpolation for 1/2 pixel [4]

In the second step of interpolation, the quarter pixel positions are interpolated through constructed samples in first step and integer samples, using average between two points, as shown in fig. 4. As an example, the position 'a' is calculated as indicated in (3).

$$a = (G + b) / 2 \quad (3)$$

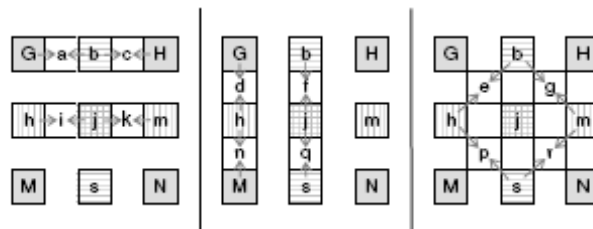


Fig. 4 – Interpolation for 1/4 de pixel [4]

After interpolation, the two selected algorithms are run separately and their results compared with previous works, without use of fractional pixel.

4. Results

The search algorithms were developed in C language and several analyses were developed targeting the algorithms evaluation in accordance with some different metrics. The first 100 frames of 10 different videos were used as input for this comparison. The software analysis was developed considering the Sum of Absolute Differences (SAD) as distortion criterion. Equation (4) shows the SAD criterion, where $SAD(x,y)$ is the SAD value for (x,y) position, R represents the reference sample, P is the search area sample and N is the block size.

$$SAD(x, y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |R_{i,j} - P_{i+x, j+y}| \quad (4)$$

The evaluations presented in this paper were divided in three criteria: PSNR (Peak-to-Signal Noise Ratio), RDA (Reduction of absolute difference) and number of SAD. The RDA is calculated by comparing the difference between the neighboring frames before and after the ME application and PSNR indicates the amount of noise existing through the criterion of similarity MSE (Mean Square Error) between the block being encoded and the block of reference frame.

Tab.1 presents the results of quarter pixel application with full search algorithm used after interpolation. Values with plus signal mean improvements and signal minus mean worsening.

Tab. 1 – Quarter pixel with full search

	DS Algorithm + Quarter in FS x original FS			DS Algorithm + Quarter in FS x original DS		
	RDA (%)	PSNR (dB)	DS+FS / FS SAD Comparisons	RDA (%)	PSNR (dB)	DS+FS / FS SAD Comparisons
46x46	-0.98	-0.29	3.18	+8.34	+0.68	2.46
80x80	-9.21	-0.79	0.78	+7.15	+0.62	2.56
144x144	-11.99	-1.01	0.18	+7.05	+0.61	2.55
208x208	-13.10	-1.10	0.08	+7.04	+0.61	2.55

In RDA terms, the quarter pixel FS provides a gain of more than 7% on the original DS, but not improvement the value of the Full Search RDA in four search areas, which was expected, because quarter FS used Diamond Search in the first and stop in a local minimum.

The second quality criterion evaluation was the PSNR, which gain the results improvement in relation to the original DS is significant, up to 0.6 dB. For another hand, this rate does not exceed the results achieved by the Full Search, but comes very close for search area 46x46 samples, with only 0.29 dB below.

To evaluate the computational cost of each quarter algorithm was used the increase/decrease number of SAD operations is performed to obtain the MV in relation original algorithms. This criterion is important to evaluate the impact that results improvements have on the architecture size.

The use of quarter pixel increases the number of SAD operations required for the generation of MV, passing from 2.5 times more calculations if compared with DS. For another hand, comparing this result with the Full Search, the computational cost is many times lower. Tab. 2 shows the results of quarter pixel DS.

Tab. 2 – Quarter pixel with diamond search

	DS Algorithm + Quarter in DS x original FS			DS Algorithm + Quarter in DS x original DS		
	RDA (%)	PSNR (dB)	DS+DS / FS SAD Comparisons	RDA (%)	PSNR (dB)	DS+DS / DS SAD Comparisons
46x46	-4.01	+0.74	0.05	+5.70	+1.70	2.54
80x80	-12.42	+0.29	0.01	+4.63	+1.70	2.56
144x144	-15.30	+0.06	0.003	+4.55	+1.69	2.55
208x208	-16.45	-0.02	0.001	+4.55	+1.68	2.55

The quarter pixel DS provides a gain of more than 4% on the original DS RDA, bus not improvement the value of the Full Search RDA in four search areas. In terms of PSNR, shown improvement in relation to the original DS is significant, up to 1.6 dB and provides gains in three search areas if compared with FS.

The result improvement of PSNR with even worse results of RDA may indicate that the local minimum that hinders the original algorithms can be good when it comes to quarter pixel.

In computational cost, quarter pixel DS presents increases if compared with original DS and significant decrease comparing with the Full Search, performing less than one 1% of SAD comparisons.

5. Conclusions

This paper presented an algorithmic evaluation of motion estimation, evaluate the computational cost increase with uses quarter pixel. This paper considers 16x16 samples blocks and four different search area sizes: 46x46, 80x80, 144x144 and 208x208 samples. These search algorithms were developed in software targeting an evaluation for a hardware implementation.

Considering the results, the most interesting algorithms to be designed in hardware was quarter pixel DS, because presents more quality results. The quarter DS algorithms use a very low number of SAD calculations, then, this implies in an important reduction in the hardware resources necessities. In the other hand, DS algorithms present some data dependencies, causing some difficulties in the parallelism exploration, but a solution to this has been implemented in others works.

6. References

- [1] INTERNATIONAL TELECOMMUNICATION UNION. ITU-T Recommendation H.262 (11/94): generic coding of moving pictures and associated audio information – part 2: video. [S.l.], 1994.
- [2] Joint Video Team of ITU-T ISO/IEC JTC 1. Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 or ISO/IEC 14496-10 AVC), 2003.
- [3] P. M. Kuhn. “Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation”. Boston: Kluwer Academic Publishers, 1999.
- [4] L. V. Agostini. “Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC. Doctoral thesis, PPGC, UFRGS. Porto Alegre, 2007.
- [5] M. Porto, et al. “Investigation of motion estimation algorithms targeting high resolution digital video compression,” In: ACM Brazilian Symposium on Multimedia and Web, WebMedia, 2007. New York: ACM, 2007. p. 111-118.
- [6] V. Bhaskaran and K. Konstantinides. “Image and Video Compression Standards: Algorithms and Architectures”. 2. ed. Massachusetts: Kluwer Academic Publisher, 1999. 454p.

Scalable Motion Vector Predictor for H.264/SVC Video Coding Standard Targeting HDTV

¹Thaísa Silva, ²Luciano Agostini, ¹Altamiro Susin, ¹Sergio Bampi
{tlsilva, bampi}@inf.ufrgs.br, Altamiro.Susin@ufrgs.br
{agostini}@ufpel.edu.br

¹Grupo de Microeletrônica (GME) – Instituto de Informática – UFRGS
²Grupo de Arquiteturas e Circuitos Integrados (GACI) – DINFO – UFPel

Abstract

The spatial scalability is one of the main features in scalable video coding which can provide efficient video representation with different spatial resolutions. This paper proposes an efficient scalable architecture for the Motion Vector Predictor with spatial scalability based on H.264/SVC standard. The Motion Vector Predictor is an important module of the Motion Compensation. This architecture was developed starting from the Motion Vector Predictor of the non-scalable H.264 standard, aiming to obtain a scalable architecture that attends the requirements to decode HDTV videos in real time. The architecture designed was described in VHDL and synthesized to Xilinx Virtex-4 FPGA. The synthesis results show that the architecture used 11,021 LUTs of the target device and it reached the necessary throughput to decode high definition videos in real time.

1. Introduction

The emerging Scalable Video Coding (SVC) standard [1] is an effort of the Joint Video Team (JVT) [2] that was completed in July 2007. It was developed to cover a wide range of video applications. The SVC is able to encode a video sequence, generating a scalable parent bitstream. From this parent bitstream, different versions containing distinct lower resolutions, frame rates, and/or bit rates can be extracted without re-encoding the original video sequence. Due to this flexibility, a custom SVC stream can be decoded on a broad range of devices.

Scalable Video Coding [3, 4] has been finalized as MPEG-4 Part 10 Advanced Video Coding (AVC) Amendment 3 (or H.264 Scalable Extension) standard, which supports scalabilities in terms of temporal, spatial and quality aspects. It is composed of one base layer coding which is compliantly encoded as H.264/MPEG-4 Part 10 (AVC) [5], and each enhancement layer coding is taken on top of its lower layer. The enhancement layers are coded based on predictions formed by the base layer pictures and previously encoded enhancement layer pictures. For spatial scalability coding in each enhancement layer of SVC, additional inter-layer prediction mechanisms have been added for the minimization of redundant information between the different layers [4]. These mechanisms are: *inter-layer intra prediction*, *inter-layer motion prediction* and *inter-layer residual prediction* from its lower layer [6]. In addition, the enhancements layers can also be encoded in the AVC compatible mode, which is independent of the inter-layer prediction coding tools.

With the inter-layer prediction, the information from the base layer is adaptively used to predict the information of the enhancement layer, which can greatly increase the coding efficiency of enhancement layer. For illustration, Fig. 1 shows a typical coder structure with 2 spatial layers.

This work focuses on inter-layer motion prediction, in which the lower layer motion information can be reused for efficient coding of the enhancement layer motion data, hence an efficient scalable architecture for the Motion Vector Predictor (MVP) is proposed.

This paper is organized as follow. Second section presents an overview of the Motion Vector Prediction in the H.264/AVC standard. Third section presents the designed Scalable Motion Vector Predictor architecture. Section four presents the synthesis results. Finally, section five presents the conclusions and future works.

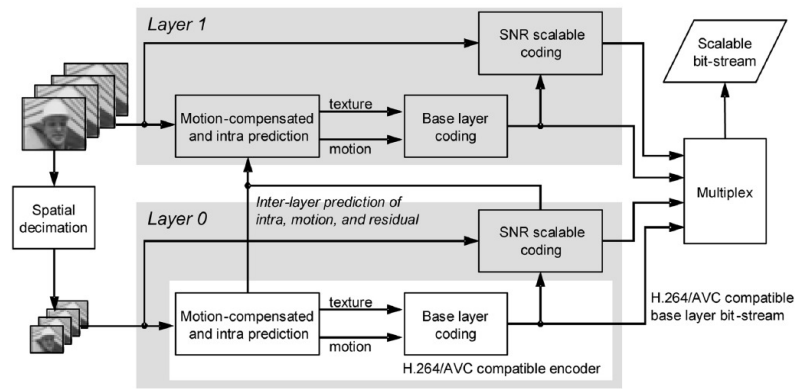


Fig. 1 – Typical coder structure for the scalable extension of H.264/AVC.

2. Overview of the Motion Vector Prediction in H.264/AVC

In the H.264/AVC Motion Estimation/Compensation process, the prediction is indicated through two indexes: the ‘reference frame index’ (a pointer to the referenced frame) and the ‘motion vector’ (MV) (a pointer to a region in the reference frame). These pointers do not have absolute values and their relative values are dependent of the neighbor parameters and samples. The values of these pointers are calculated through a process called Motion Vector Prediction (MVP) which presents some different modes of operation.

The standard prediction mode calculates MVs using neighbor block information, when available, and differential motion vectors from the bitstream. The direct prediction uses information from time co-located blocks of a previously decoded frame. MVP is defined as a highly sequential algorithm and it is frequently solved by a software approach [7].

3. Designed Scalable Motion Vector Predictor Architecture

The architecture designed in this work was developed starting from the architecture designed for the Motion Prediction Vector of the non-scalable standard [8]. However, in the Scalable Motion Vector Predictor architecture the prediction of motion vectors is achieved using the upsampled lower resolution motion vectors.

Since the motion vectors of all layers are highly correlated, the motion vectors of the enhancement layer can be derived from those vectors of the co-located blocks of the base layer when both macroblocks of the base layer and the enhancement layer are coded as inter mode [1]. The macroblock partitioning is obtained by upsampling the partition of the corresponding block of the lower resolution layer, as shown in Fig. 2. For instance, in the Fig. 2(a) the 8x16 and 8x8 blocks of the higher resolution layer are obtained from 4x8 and 4x4 blocks correspond of the lower resolution layer. The Fig. 2(b) shows that 8x16 blocks or larger of the lower resolution layer correspond to 16x16 blocks in the higher resolution layer.

For the obtained macroblock partitions, the same reference picture indexes used in the corresponding submacroblock partition of the base layer block are used in the enhancement layer; and the associated motion vectors are scaled by a factor of 2, as illustrated with the black arrow in Fig. 3. Additionally, a quarter sample motion vector refinement is transmitted for each motion vector. Furthermore, a motion vector could also be predicted from the spatial neighboring macroblocks. A flag is transmitted with each motion vector difference to indicate whether the motion vector predictor is estimated from the corresponding scaled base layer motion vector or from the spatial neighboring macroblocks.

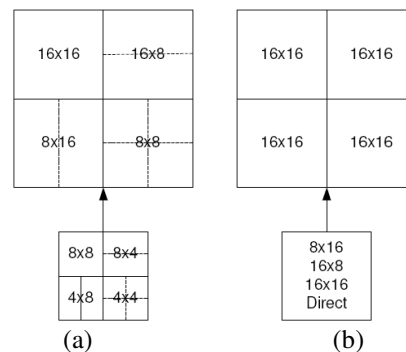


Fig. 2 – The up-sampling method of macroblock partition for inter prediction.

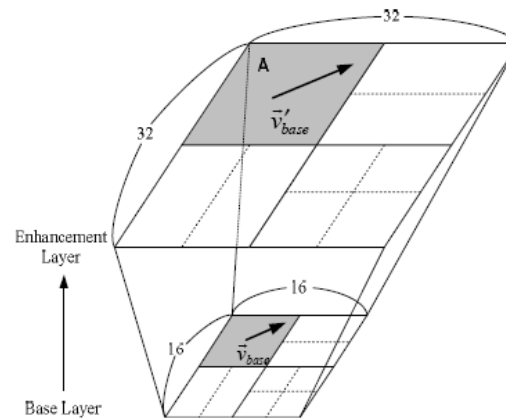


Fig. 3 – Inter-layer motion prediction.

The designed architecture in this work is presented in the Fig. 4 as a high level diagram, where the blocks that compose the Scalable Motion Vector Predictor architecture are highlighted.

The architecture is formed by two non-scalable MVP [8], which the motion vectors and reference indexes predicted in the base layer are wrote in a memory, after this, the motion data decoded in the base layer are operated in the Inter-layer motion prediction block as described above. Finally, this motion data are used in the decoding of the enhancement layer. In the next section, the synthesis results extracted from this architecture will be presented.

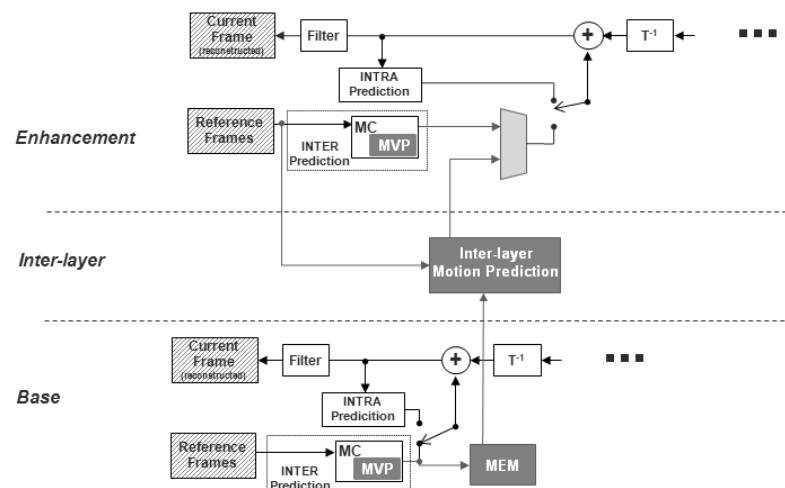


Fig. 4 – Inter-layer motion prediction

4. Synthesis Results

The MVP scalable architecture was described in VHDL and synthesized to the Xilinx Virtex-4 FPGA. These results are presented in tab. 1 and compared with the non-scalable version of the MVP.

Tab.1 – Synthesis results of the scalable and non-scalable MVP

	Scalable MVP	Non-scalable MVP
LUTs	11,021	6,653
Registers	6,192	3,955
Frequency(MHz)	115.8	115.8

Selected Device: Virtex-4 XC4VLX200

Through the obtained results it was possible to verify that the scalable MVP developed in this work used about 2 times more hardware resources than the non-scalable MVP, with an operation frequency of 115.8 MHz. However, it is important to emphasize that the Scalable MVP architecture is able to decode multiple resolutions

from a single bitstream, since it is using the spatial scalability. Instead of one bitstream for each device, only one bitstream is generated and the decoder, in this case the Scalable MVP decoder, selects only the parts related to its capacity. In throughput terms, the architecture designed kept the throughput of the non-scalable MVP architecture [8] processing one sample per clock cycle, reaching the requirements to process HDTV (1920x1088 pixels) frames in real time.

This is the first design related in the literature for the Scalable MVP of the H.264/SVC standard targeting HDTV, hence not was possible to achieve comparisons with others works.

5. Conclusions and Future Works

This work presented the design of a Scalable Motion Vector Predictor architecture targeting to decode HDTV videos in real time using Scalable Video Coding. The architecture designed used more hardware resources than the non-scalable architecture, attaining the same throughput, but with the great advantage of the flexibility, allowing a decoding on a broad range of devices from a single bistream.

The designed solution presented good results. The average processing rate presented reached the requirements necessary to process HDTV (1920x1088 pixels) frames in real time.

As future works it is planned to achieve a design space exploration, aiming to reduce the hardware resources used. Besides, using the Scalable MVP architecture, it is planned the design of a complete Scalable Motion Compensation architecture. Then, a comparison between the results founded with the results of the Non-Scalable Motion Compensation architecture [9] will be possible.

6. References

- [1] J. Reichel, M. Wien, and H. Schwarz, Eds., Joint Scalable Video Model JSVM 6, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, 2006.
- [2] INTERNATIONAL TELECOMMUNICATION UNION. Joint Video Team (JVT), Available in: www.itu.int/ITU-T/studygroups/com16/jvt/, 2008.
- [3] T. Wiegand, G. Sullivan, J. Reichel, H. Schwarz and M. Wien, ISO/IEC JTC 1/SC 29/WG 11 and ITU-T SG16 Q.6: JVT-W201 'Joint Draft 10 of SVC Amendment,' 23th Meeting, San Jose, California, April 2007.
- [4] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," IEEE Transaction on Circuits and Systems on Video Technology, vol.17, no.9, Sep. 2007
- [5] ITU-T and ISO/IEC JVT 1, "Advanced video coding for generic audiovisual services," ITU-T Recommendation H.264 and ISO/IEC 14496-1- (MPEG-4 AVC), Version 1:May 2003, Version 2: Jan 2004, Version 3:Sep.2004, Version 4:July 2005
- [6] C.A. Segall and G.J. Sullivan, "Spatial Scalability Within the H.264/AVC Scalable Video Coding Extension," IEEE Transaction on Circuits and Systems on Video Technology, vol.17, no.9, Sep. 2007.
- [7] S. Wang et al; "A Platform-Based MPEG4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining", ICICS-PCM, 2003. p.51-55.
- [8] B. Zatt, A. Azevedo, A. Susin, and S. Bampi, "Preditor de Vetores de Movimento para o Padrão H.264/AVC Perfil Main". In: XIII IBERCHIP Workshop, 2007, Lima. XIII IBERCHIP Workshop, 2007.
- [9] A. Azevedo, B. Zatt, L. Agostini, and S. Bampi, "Motion Compensation Decoder Architecture for H.264/AVC Main Profile Targeting HDTV". In: VLSI-SOC 2006 - 14TH IFIP International Conference on Very Large Scale Integration, 2006, Nice. VLSI-SOC 2006 - 14TH IFIP International Conference on Very Large Scale Integration, 2006. p. 52-57.

Decoding Significance Map Optimized by the Speculative Processing of CABAD

Dieison Antonello Deprá, Sergio Bampi

{dadepra, bampi}@inf.ufrgs.br

Instituto de Informática – PPGC – GME

Federal University of Rio Grande do Sul (UFRGS)

Abstract

This paper presents the design and implementation of a hardware architecture dedicated to binary arithmetic decoder (BAD) engines of CABAD, as defined in the H.264/AVC video compression standard. The BAD is the most important CABAD process, which is the main entropy encoding method defined by the H.264/AVC standard. The BAD is composed by three different engines: Regular, Bypass and Final. We conducted a large set of software experiments to study the utilization of each kind of engine. Based on bitstream flow analysis we proposed a new improvement to significance map decoding through a dedicated hardware architecture, which maximizes the trade-off between hardware cost and throughput of BAD engines. The proposed solution was described in VHDL and synthesized to Xilinx Virtex2-Pro FPGA. The results shown that developed architecture reaches the frequency of 103 MHz and can deliver up to 4 bin/cycle in bypass engines.

1. Introduction

Today we can observe a growing popularity of high definition digital videos, mainly for applications that require real-time decoding. This creates the need for higher coding efficiency to save storage space and transmission bandwidth. Techniques for compressing video are used to meet these needs [1]. The most advanced coding standard, which is considered state-of-the-art, is the H.264/AVC, defined by the ITU-T/ISO/IEC [2]. This standard defines a set of tools, which act in different domains of image representation to achieve higher compression rates, reaching gain in compression rate of 50% in relation to MPEG-2 standard. The H.264/AVC standard introduces many innovative techniques to explore the elimination of redundancies, present in digital video sequences. One of these main techniques is related to the entropy coding method.

The CABAC (Context-Adaptive Binary Arithmetic Coder) [3] is the most important entropy encoding method defined by the H.264/AVC standard, allowing the H.264/AVC to reach 15% coding gain over CAVLC (Context-Adaptive Vary Length Coder) [4]. However, to obtain this gain it is necessary to pay the cost of increasing computer complexity. The CABAC is considered the main bottleneck in the decoding process because its algorithm is essentially sequential. Each iteration step produces only one bin and the next step depends on the values produced in the previous iteration [1].

Techniques to reduce the latency and data dependency of CABAD (Context-Adaptive Binary Arithmetic Decoding) have been widely discussed in the literature. Basically, five approaches are proposed: 1) pipeline; 2) contexts pre-fetching and cache; 3) elimination of renormalization loop; 4) parallel decoding engines, and 5) memory organization. The pipeline strategy is employed by [5] to increase the rate of bins/cycle. In [6] are present an alternative to solve the latency of renormalization process. The speculative processing through the use of parallel decoding engines is explored in [7], and [8] and [9]. High efficiency decoding process by employing pre-fetching and cache contexts is discussed in [7]. The memory optimization and reorganization are addressed in [5].

This work presents the development of hardware architecture dedicated to binary arithmetic decoder engines of CABAD. The design aims to high efficiency implementation, based on software experiments of bitstream flow analysis. In the next section are presented details for all BAD engine kinds of CABAD as defined by H.264/AVC standard. Section 3 presents the bitstream flow analysis. The architecture proposal and results achieved for the proposed architecture are presented in Section 4. Finally, some conclusions are made in Section 5.

2. Context Adaptive Binary Arithmetic Decoder

The CABAC is a method of entropy encoding which transforms the value of a symbol in a word of code. It generates variable length codes near to the theoretical limit of entropy. The CABAC works with recursive subdivision of ranges combined with context models that allow reaching better coding efficiency. However, modeling probabilities of occurrence of each symbol brings great computational cost. To decrease computational complexity, CABAC adopts a binary alphabet. Thus, it is necessary to do an additional step that

converts the symbol values in binary sequences reducing to two the set of possible values, zero or one. For more details see [2].

The H.264/AVC standard defines CABAC as one of the entropy encoding methods available in the Main profile. Here the CABAC is employed, from the macroblock layer, for encoding data generated by tools that act on the transformation of spatial, temporal and psycho-visual redundancies [2].

CABAD is the inverse process of CABAC and it works in a very similar way. CABAD receives as input an encoded bitstream, which contains information about each tool and its parameters used by the encoder to make its decisions. Inside CABAD these information are named syntax elements (SE) [3]. To decode each SE, the CABAD can perform N iterations and for each step one bit is generated consuming zero, one or more input bits. Each generated bit is named "bin" and the "bin" produced on each iteration is concatenated with the previous forming a sequence named "binstring".

The flow for decoding each SE involves five procedures that can be organized into four basic stages: I) Context address generating II) Context memory access; III) Binary arithmetic decoding engines; IV.a) Binstring matching; and IV.b) Context updating. In the first stage, the context address is calculated according to SE type that is being processed. Following, the context memory is read returning one state index and one bit with the value of the most probable symbol (MPS). From this index, the estimated probability range of the least probable symbol (LPS) is obtained. After, BAD engine receives the context model parameters and produces a new bin value. In the fourth stage, generated bins are compared with a pattern of expected bins and, in parallel, the context model should be updated in memory to guarantee that the next execution will receive the updated probability of occurrence of each SE.

The CABAD become the bottleneck of decoding process because its nature is essentially sequential due to the large number of data dependencies. To decode each "bin" it is necessary to obtain information from previous decoded "bins" [9]. For hardware architectures implementation, this fact causes a huge impact because four cycles are needed for decoding one bin and one SE can be represented by several bins.

3. Bitstream Flow Analysis

To explore the appropriate level of parallelism for decoder engines, a series of evaluations over bitstream generated by the reference software (JM) of H.264/AVC standard [10] was performed. To conduct these experiments were used the classical digital video sequences, such as: rush-hour, pedestrian_area, blue_sky, riverbed, sunflower, carphone, foreman, costguard, tractor and others. In the total were utilized 60 digital video sequences organized into four typical resolutions: QCIF, CIF, D1 and HD1080p. The tool used for encoding was JM, version 10.2, with the following encoding parameters: Profile IDC = 77, Level IDC = 40, SymbolMode = CABAC. For all the video sequences, seven encoding configurations were performed varying the quantization parameters of QPISlice and QPPSlice, the following value pairs were used: 0:0, 6:0, 12:6, 18:12, 24:18, 28:20 e 36:26, resulting in 420 different encoded bitstreams.

To find the average of bypass decoder engine utilization for two consecutive decoding bins changes were implemented in the JM10.2 decoder to collect statistical information about usage of bypass engines for every SE kind. The implemented modification collects information on how much times each kind of engine is triggered to each SE type, how often bypass the engines is used in consecutive order for the same SE and how many times more than four consecutive bins are processed via bypass engine.

Analyzing the results, is possible to note that only two types of SEs present significant use of bypass coding engine. So SE was fetched in only three classes, which are: 1) coefficients transformed (COEF); 2) Component of the motion vector differential (MVD); and 3) Other SEs. The results for all digital video sequences were summarized and, on average, SE COEF and MVD match for 69.06% of all occurrences of SEs in the bitstream, but these two SE kinds produce approximately 84.06% of all the bins. Moreover, COEF and MVD together use the bypass engine to produce, on average, 68.20% of their bins. The rate of bypass bins processed in consecutive way corresponds approximately to 28.39% of the total. However, we observed that on average 28% of times that the bypass engine is performed, it produces 4 or more consecutive bins. Thus, the use of four bypass engines in parallel can provide a reduction to 3.98% in the total number of cycles, considering an average throughput of one bin/cycle in regular engines.

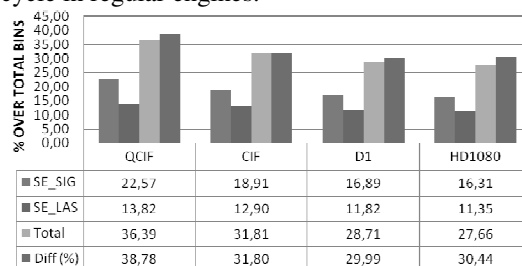


Fig. 1 – Analysis of differences among significance map bins occurs for many resolutions.

4. Designed Architecture and Results

The diagram illustrates the proposed architecture. It features four main input blocks at the top: RPS, STATE, BS, and MPS. RPS and STATE feed into a block labeled S10, which then connects to a block labeled RANGE. BS and MPS feed into a block labeled OFFSET. The RANGE block outputs to a block labeled TERMINATE. The OFFSET block outputs to a block labeled REGULAR 1..2. The TERMINATE block also outputs to the REGULAR 1..2 block. The REGULAR 1..2 block outputs to a stack of blocks labeled BYPASS 1..4. The BYPASS 1..4 blocks output to an OUTPUT MERGE block. The OUTPUT MERGE block outputs to four outputs: new MPS, new State, bin value, and bit Stream.

The proposal of [11] uses speculative processing to accelerate the decoding of the significance map. For this, the first engine decodes the regular SE SIG_FLAG while the second engine decodes the regular SE LAST_SIG so speculative. However, as shown in Fig. 1 at 30% of the time this premise fails, because when the value of SE SIG_FLAG is equal to zero the next subsequent SE should be the SIG_FLAG instead of a LAST_FLAG. To solve this problem, in our proposal, three different contexts are provided for binary arithmetic decoding block. The first context is related to the type SE SIG_FLAG to be decoded by the first regular engine, the second context is related to the SE type LAST_FLAG, while the third relates to the other SE of type SIG_FLAG.

The developed architecture was described in VHDL and synthesized to a Xilinx FPGA device XUPX2CPVP30. The Xilinx ISE 9.1 was used for the architecture synthesis and validation. The maximum achieved frequency was 103 MHz. The synthesis results and comparison with previous proposals are presented in tab. 1. Our architecture consumes $\sim 5\%$ more hardware resources than the proposal of [7] and $\sim 1\%$ more than [11] while the maximum frequency of operation remains almost unchanged, suffering reduced below 0.5 MHz. However, as shown in Fig. 4, our proposal has a potential gain in throughput when compared to previous work.

being ~9% when compared to [7] and ~5% when compared to [11]. The gain obtained over the previous works comes from the fact that our proposal uses two new approaches, such as: four bypass engines (4 BYPASS) and speculative processing for significance map (SP SIGMAP). In the [11] proposal just four bypass engines are used while in the [7] neither of them is used.

5. Conclusions

This work presents the challenges of the high efficiency implementation for BAD engines and proposes a new architecture based on an analysis of bitstream flow and common approaches found in literature. The new architecture is more efficient in relation to [11] because solve the misses in speculative process of significance map through the smart approach that have the small hardware cost. The developed work demonstrates that is possible to discover new alternatives to CABAD constrain thought analysis of bitstream flow. As future work, we plan to extend these experiments to evaluate the efficiency of regular engines branch to obtain greater throughput to complete system.

Tab.1 – Synthesis results and comparison for BAD architectures.

Resource	[7] Proposal	[11] Proposal	Our Proposal	Differences (%)	
				[7]	[11]
Slices	788	824	834	+5.84	+1.21
LUTs	1446	1503	1520	+5.12	+1.13
Frequency (MHz)	103.5	103.4	103.0	-0.48	-0.39

* Device Xilinx XUPX2CPVP30.

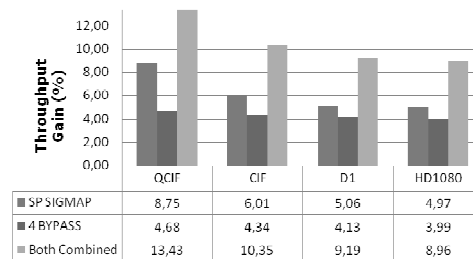


Fig. 4 – Gain analysis for system throughput with speculative process and parallel engines approaches.

6. References

- [1] WEIGAND, Thomas; et al. Overview of the H.264/AVC Video Coding Standard. In: IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, pp. 560-576, N° 7, July 2003.
- [2] ITU, INTERNATIONAL TELECOMMUNICATION UNION. ITU-T Recommendation H.264 Advanced video coding for generic audiovisual services. 2005.
- [3] MARPE, Devlet. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. In: IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, N° 7, July 2003.
- [4] CHEN, Tung-Chien; et. al. Architecture Design of Context-Based Adaptive Variable-Length Coding for H.264/AVC. Circuits and Systems II: Express Briefs, IEEE Transactions on, vol.53, no.9, pp.832-836, Sept. 2006.
- [5] YANG, Yao-Chang; LIN, Chien-Chang; CHANG, Hsui-Cheng; SU, Ching-Lung; and GUO, Jiun-In. A High Throughput VLSI Architecture Design for H.264 Context-Based Adaptive Binary Arithmetic Decoding With Look Ahead Parsing. In: (ICME) Multimedia and Expo, 2006 IEEE International Conference on, pp. 357-360, July 2006.
- [6] EECKHAUT, Hendrik; et al. Optimizing the critical loop in the H.264/AVC CABAC decoder. In: Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on. December 2006.
- [7] YU, Wei; HE, Yun. A High Performance CABAC Decoding Architecture. In: IEEE Transactions on Consumer Electronics, Vol. 51, pp. 1352-1359, No. 4, November 2005.
- [8] BINGBO, Li; DING, Zhang; JIAN, Fang; LIANGHAO, Wang; MING, Zhang. A high-performance VLSI architecture for CABAC decoding in H.264/AVC. In: ASICON '07. 7th International Conference on, pp. 790-793, October 2007.
- [9] KIM, Chung-Hyo; PARK, In-Cheol. High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction. In: Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on. May 2006.
- [10] SUHRING, Karsten. H.264/AVC Reference Software. In: Fraunhofer Heinrich-Hertz-Institute. Available in: <http://iphome.hhi.de/suehring/ttml/download/>. [Accessed: March 2008].
- [11] DEPRA, Dieison, A.; ROSA, Vagner, S.; BAMPI, Sergio. A Novel Hardware Architecture Design for Binary Arithmetic Decoder Engines Based on Bitstream Flow Analysis. In: SBCCI '08. 21st Symposium on Integrated Circuits and Systems Design, pp. 239-244, September 2008.

H.264/AVC Variable Block Size Motion Estimation for Real-Time 1080HD Video Encoding

¹Roger Porto, ²Luciano Agostini, ¹Sergio Bampi
{recporto,bampi}@inf.ufrgs.br, agostini@ufpel.edu.br

¹Universidade Federal do Rio Grande do Sul

²Universidade Federal de Pelotas

Abstract

Amongst the video compression standards, the latest one is the H.264/AVC [1]. This standard reaches the highest compression rates when compared to the previous standards. On the other hand, it has a high computational complexity. This high computational complexity makes it difficult the development of software applications running in a current processor when high definitions videos are considered. Thus, hardware implementations become essential. Addressing the hardware architectures, this work presents the architectural design for the variable block size motion estimation (VBSME) defined in the H.264/AVC standard. This architecture is based on full search motion estimation algorithm and SAD calculation. This architecture is able to produce the 41 motion vectors within a macroblock as specified in the standard. The implementation of this architecture was based on standard cell methodology in 0.18 μ m CMOS technology. The architecture reached a throughput of 34 1080HD frames per second.

1. Introduction

This paper presents the design of an architecture for variable block size motion estimation according to the H.264/AVC digital video coding standard. This architecture is capable to generate all the 41 different motion vectors defined by the H.264/AVC standard. These motion vectors refer to each sub-partition within a 16x16 macroblock. This architecture employs the full search algorithm and the sum of absolute differences (SAD) calculation. The synthesis results shows that the architecture can be used in real-time 1080 HDTV applications.

This paper is organized as follows. The section two presents the architecture for variable block size motion estimation and its main modules. After, the section three presents the synthesis results of the architecture. Section four presents the comparison of the present work with related works found in literature. Finally, the conclusions of this work are presented in section five.

2. Architecture for Variable Block Size Motion Estimation

The strategy used in this work to perform variable block size motion estimation was to merge the SADs of smaller blocks to calculate SADs of larger blocks. The architecture for variable block size motion estimation is shown in fig. 1. The main modules of this architecture will be detailed in the next items of the paper.

For a search area of 16x16 pixels there are 13 candidate blocks in a row and 13 candidate blocks in a column. Thus, 169 candidate blocks are compared to the current block to find the best match. In this way, in each level of the architecture will be 169 SADs referent to each new candidate block. Summarizing the process of merging SADs, in each level, 169 SAD values from each memory are grouped two by two to form 169 new SAD values. These new SAD values refer to the SADs of candidate blocks of the next level. One value between these 169 new SADs is chosen as minimum and its motion vector is sent to the output. The 169 new SAD values of SAD are stored in another memory. These 169 new values will be grouped two by two with another 169 values and so on.

2.1. Architecture for 4x4 Motion Estimation

Fig. 2 shows the block diagram of the architecture for the 4x4 motion estimation (4x4 ME). The size of the search area was defined in 16x16 pixels. The details of the main modules that form this part of the architecture will be presented in the next items of the paper. These main modules are the SAD rows and its processing units (PUs), the comparators, the memories and the memory manager.

2.1.1. Memory Manager

The memory manager is responsible for read these memories according with the needs of the architecture. The samples read from the search area memory are copied to the search area register (SAR). The current block

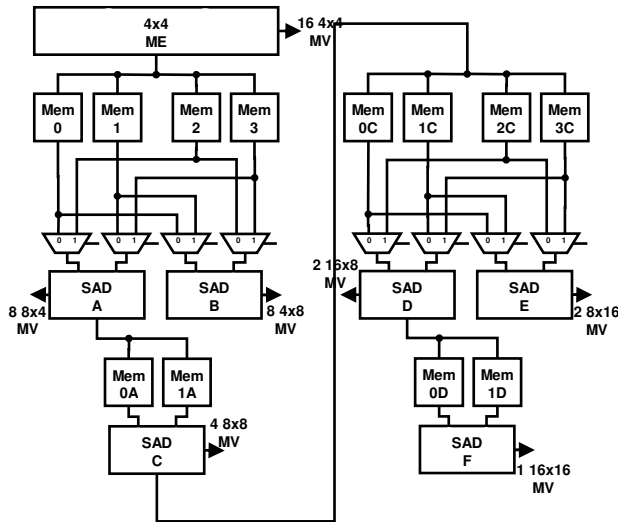


Fig. 1 – Block diagram of the VBSME.

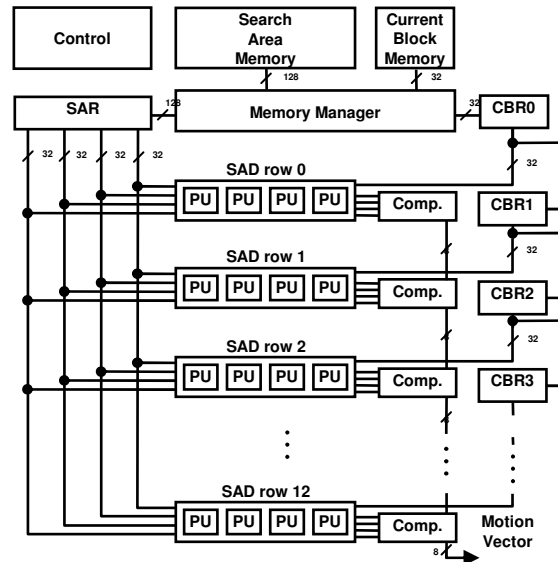


Fig. 2 – Block diagram of the 4x4 ME module.

registers (CBR0 to CBR12) stores the samples read from the current block memory. These registers provide the correct data at the inputs of the PUs. The memory manager, the registers and the memories are depicted in fig. 2.

2.1.2. Architecture for SAD Calculation

The architecture for SAD calculation was designed hierarchically. The highest hierarchical level is the matrix of SADs. The matrix of SADs is formed by 13 SAD rows. Each SAD row is formed by 4 processing units (PUs). The PU calculates the distortion between a row of the current block and a row of the candidate block. Thus, each PU receives 4 samples of the current block (A0 to A3) and 4 samples of the candidate block (R0 the R3) as inputs. The PUs were designed to operate in a pipeline of 3 stages, as depicted in fig. 3.

The partial SAD of a candidate block generated by a PU (SAD of a row) need to be added to the SADs of other rows to generate the total SAD of a block. The total SAD is the sum 16 SAD values (four rows with four samples each). The SAD row, presented in fig. 4, groups four PUs and generates the total SAD of each candidate block. In total there are 13 accumulators, one for each candidate block in the row. When a SAD row completes its calculations, the registers ACC0 to ACC12 will hold the final SADs of the 13 candidate blocks.

The first SAD row calculates the SAD values for the 13 candidate blocks that begin at the first row of the search area, comparing the current block to these candidate blocks. The second SAD row does the same for the 13 candidate blocks that begin at the second row of the reference area. This process is carried out until, in the thirteenth SAD row, the current block is compared with the 13 last candidate blocks. In this way, the current block is compared to the 169 candidate blocks to find the best match.

2.1.3. Architecture of the Comparator

The comparator was designed in a pipeline of 4 stages. It can perform a comparison of 4 SADs in parallel. In four cycles all 13 values generated by the SAD row have been delivered to the comparator. The architecture of the comparator should be able to identify what is the minimum SAD among the 13 values. When this minimum SAD is found, the comparator still needs to compare it with the minimum SAD from the previous SAD row.

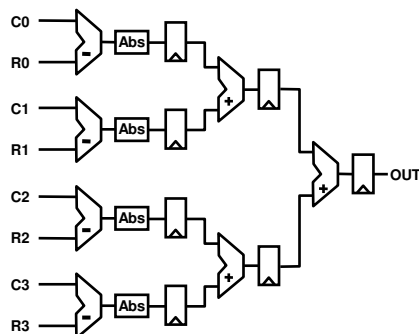


Fig. 3 – Architecture of the PU.

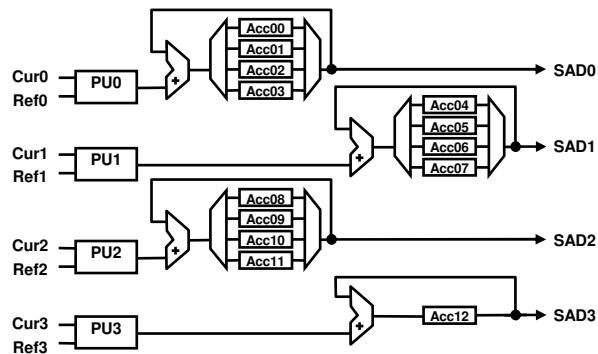


Fig. 4 – Block diagram of the SAD row.

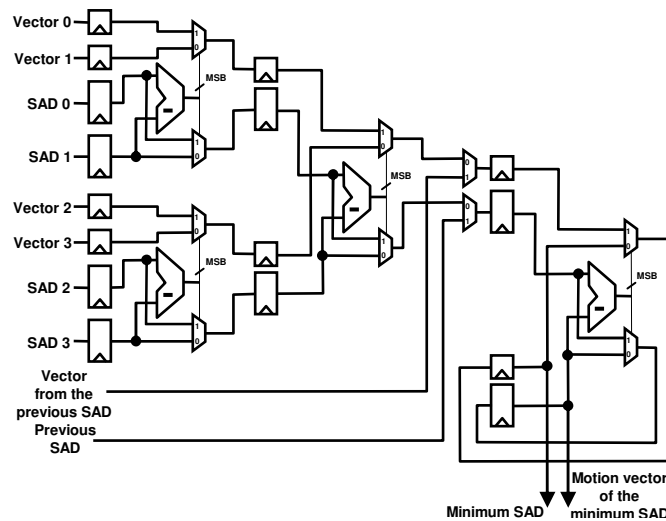


Fig. 5 – Architecture of the comparator.

2.2. Memories

The memories shown in fig. 1 are responsible to store the SADs of the candidate blocks so they can be grouped to form larger SADs. The total amount of memory used was around 51 KB, including the memory used by the 4x4 ME module. Four memories have been implemented with different sizes. These memories are used to store the results of the calculation of SADs for 4x4, 8x4, 8x8 and 16x8 block sizes. Each one of the 12 memories stores the SADs of 169 candidate blocks.

2.3. Architecture for SAD Merging

The architecture for SAD calculation for the 4x8, 8x4, 8x8, 8x16, 16x8 and 16x16 block sizes is presented in fig. 6. This module is very similar to the architecture of the comparator previously shown. SAD A, SAD B, SAD C, SAD D, SAD E, and SAD F represent this module in fig. 1. This module was designed in a pipeline with five stages as can be seen in fig. 6. In the first stage, eight SADs are grouped, four for each block, forming four new SADs. In the last four stages the four SADs generated are compared to search the minimum SAD among these four SADs. The last stage generates the minimum SAD and its motion vector.

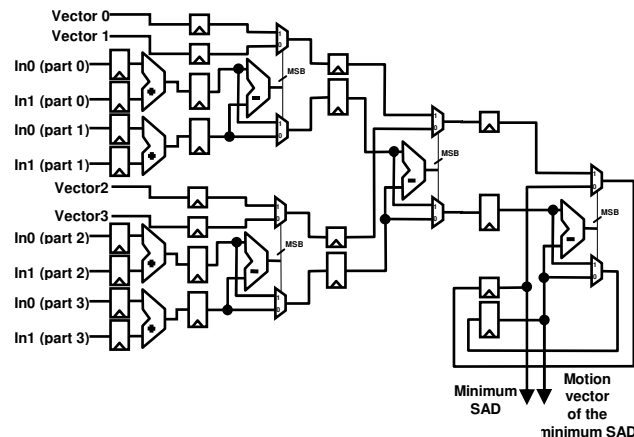


Fig. 6 – Architecture for SAD merging.

3. Synthesis Results

The VBSME and all its modules were described in VHDL. The implementation was based on standard cell methodology in 0.18 μ m CMOS technology. The architecture was synthesized using the LeonardoSpectrum tool by Mentor Graphics [2]. Tab. I presents the synthesis results of the VBSME. The maximum frequency was 296.3 MHz. Therefore, 34 1080HD frames per second can be processed. The architecture occupied 113 K gates.

Tab 1 – Synthesis Results.

Area	Frequency
113 K gates	296.3 MHz
Technology: TSMC 0.18 μ m	

4. Comparisons with Related Works

This section presents comparisons with some related works found in literature. All works listed here use the full search algorithm and SAD calculation to do the motion estimation. Tab. II shows the comparative results highlighting the number of processing units, number of gates, size of search area, technology, frequency, and throughput. In addition, the relation between the number of processing units and the reached throughput is also presented. The most efficient architectures in this last criterion are the ones with smaller results. Thus, it is possible to do a fair evaluation on the performance of the works.

From tab. II it is possible to see that the solution designed in this work is capable to process 1080 HDTV at 34 frames per second. Another conclusion on the results of tab. II is that our work presents the second highest throughput among all the investigated solutions. A highlight in table II is the high throughput achieved by the architecture designed by Ou [5]. It is important to say that Ou uses almost five times more processing units than our work to achieve this throughput. Our architecture is the second best in number of PUs / throughput. The solution developed by Kim [3] is the first one. On the other hand, the solution presented by Kim has a throughput lesser than that obtained by our work. The architecture presented by Kim is not able to reach real-time at 30 fps when processing 1080 HDTV resolutions.

Tab. 2 – Comparative Results.

Solution	# of PUs	# of Gates	Search Area	Technology	Frequency (MHz)	Throughput		# of Pus / Throughput
						<i>Msamples/s</i>	<i>Frames/s</i>	
Kim [3]	16	39K	16x16	DonbuAnam 0.18 μ m	416	25.95	12 (1080 HD)	0.61
Ours	52	113K	16x16	TSMC 0.18 μ m	296	72.52	34 (1080 HD)	0.72
Yap [4]	16	61K	16x16	TSMC 0.13 μ m	294	18.24	8 (1080 HD)	0.87
Ou [5]	256	597K	16x16	UMC 0.18 μ m	123	125.33	60 (1080 HD)	2.04
Liu [6]	192	486K	192x128	TSMC 0.18 μ m	200	62.66	30 (1080 HD)	3.06

5. Conclusions

This paper presented an architecture design for variable block size motion estimation according to the H.264/AVC standard. Details of the whole architecture and its main modules are shown in this paper.

This architecture is able to generate all 41 motion vectors defined by the standard. The implementation of this architecture was based on standard cell methodology in 0.18 μ m CMOS technology. The maximum reached operation frequency was of 296.3 MHz. With a throughput of 72.52 million samples per second the architecture can process 34 1080HD frames per second. This throughput is the second largest one among all solutions investigated only losing to an architecture that uses five times more processing units than our work. The architecture designed in this work was the second best in use of PUs, but the best solution is not able to reach real-time when processing 1080HD frames.

6. References

- [1] International Telecommunication Union, ITU-T Recommendation H.264 (05/03): Advanced Video Coding for Generic Audiovisual Services, 2003.
- [2] Mentor Graphics, “LeonardoSpectrum”, Feb. 2009; http://www.mentor.com/products/fpga_pld/synthesis/leonardo_spectrum.
- [3] J. Kim, and T. Park, “A Novel VLSI Architecture for Full-Search Variable Block-Size Motion Estimation”, *Proc. IEEE Region 10 Conference (TENCON 2007)*, IEEE, 2007, pp. 1-4.
- [4] S. Yap, and J. McCanny, “A VLSI Architecture for Variable Block Size Video Motion Estimation”, *IEEE Transactions on Circuits and Systems – II: Express Briefs*, vol. 51, July 2004, pp. 384-389.
- [5] C. Ou, C. Le, and W. Huang, “An Efficient VLSI Architecture for H.264 Variable Block Size Motion Estimation”, *IEEE Transactions on Consumer Electronics*, vol. 51, issue 4, Nov. 2005, pp. 1291-1299.
- [6] Z. Liu, et al, “32-Parallel SAD Tree Hardwired Engine for Variable Block Size Motion Estimation in HDTV1080P Real-Time Encoding Application”. *Proc. IEEE Workshop on Signal Processing Systems, (SiPS 2007)*, IEEE, 2007, pp. 675-680.

An Architecture for a T Module of the H.264/AVC Focusing in the Intra Prediction Restrictions

Daniel Palomino, Felipe Sampaio, Robson Dornelles, Luciano Agostini

{danielp.ifm, rdornelles.ifm, fsampaio.ifm, agostini}@ufpel.edu.br

Universidade Federal de Pelotas – UFPel
Grupo de Arquiteturas e Circuitos Integrados – GACI

Abstract

This Paper presents an architecture for a forward transforms module dedicated to the Intra Prediction of the main profile of the H.264/AVC video coding standard. This architecture was designed intending to achieve the best possible relation between throughput and latency, since these characteristics are extremely important to define the Intra Prediction performance. The architecture was described in VHDL and synthesized to Altera Stratix II FPGA and to TSMC 0.18 μm standard-cells technology. The architecture was validated using Mentor Graphics ModelSim. The architecture reaches an operation frequency of 199.5 MHz when mapped to standard-cells, processing till 239 QHDTV frames per second. Besides, considering the Intra Prediction restrictions, the architecture presents the best results when compared with all related works.

1. Introduction

H.264/AVC [1] is the latest video coding standard. It was developed by experts from ITU-T and ISO-IEC intending to double the compression rates when compared with the previous standards, like MPEG-2. The investigation about hardware solutions for the H.264/AVC is inserted in the Brazilian research effort to develop the Brazilian System of Digital Television broadcast (SBTVD) [2].

The main modules in a H.264/AVC coder are: Inter Prediction (composed by motion estimation and motion compensation), Intra Prediction, Forward and Inverse Transforms, Forward and Inverse Quantization, Deblocking Filter and Entropy Coder.

The Intra Prediction module explores the spatial redundancy in a frame, exploring the similarities of neighbor blocks. It works using the data of blocks previously processed to predict the current one. In the coding process, H.264/AVC standard defines that the residual data between the original and the predict block must pass through the Forward Transform (T), Forward Quantization (Q), Inverse Quantization (IQ) and Inverse Transform (IT) to generate the reconstructed block that will be used as reference by the Intra Prediction. The Intra Prediction stays idle while this block is being processed by the T/Q/IQ/IT loop [3]. Thus, the number of clock cycles that a block needs to be processed by the T, Q, IQ and IT modules affects the performance of the Intra Prediction. It means that the latency of architectures designed for this reconstruction path must be as low as possible and the throughput must be as high as possible.

This work presents an architecture design for the Forward Transforms module (T module) intending to reduce the impact of this module in the Intra Prediction process. Then, this module was designed to reach the best possible relation between throughput and latency.

H.264/AVC Main profile defines a color relation of 4:2:0. It means that a macroblock contains 16x16 luma (Y) samples, 8x8 chroma blue (Cb) samples and 8x8 chroma red (Cr) samples. The Intra Prediction works in three different modes: luma 4x4, luma 16x16 and chroma 8x8 [4].

Fig. 1 shows the block diagram of an H.264/AVC encoder, where the designed architecture is highlighted.

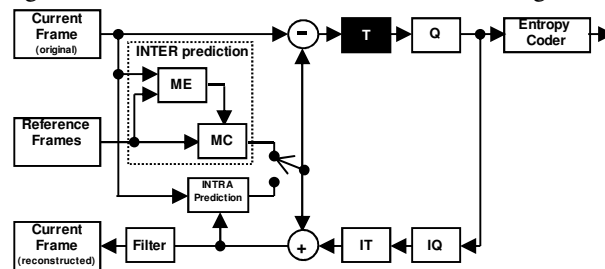


Fig. 1 – Block Diagram of an H.264/AVC encoder.

This work is presented as follows. Section 2 presents the forward transforms. Section 3 presents some details about the designed architecture. Section 4 shows the synthesis results and some related works. Finally, Section 5 presents the conclusions of this work and future works.

2. Forward Transforms

The input data for the T module are 4x4 residual blocks generated in the Intra Prediction process. H.264/AVC standard defines three forward transforms: the 4x4 Forward Discrete Cosine Transform (4x4 FDCT), the 4x4 Forward Hadamard Transform (4x4 FHAD) and the 2x2 Forward Hadamard Transform (2x2 FHAD).

Equation (1) shows the 4x4 FDCT definition. In the H.264/AVC standard, the multiplication by the scalar E_f is made in the Quantization process [4]. Equation (2) shows the 4x4 FHAD definition, and finally, Equation (3) shows the 2x2 FHAD definition.

The 4x4 FDCT transform is applied to all chroma and luma samples in all Intra Prediction modes. The 4x4 FHAD processes only the luma DC coefficients when the Intra Prediction mode is intra 16x16. The 2x2 FHAD processes the chroma DC coefficients when the mode is chroma 8x8.

$$Y = C_f X C_f^T \otimes E_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 2 & -1 \end{bmatrix} \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & b^2 & \frac{ab}{2} & b^2 \\ \frac{a^2}{2} & \frac{ab}{4} & \frac{a^2}{2} & \frac{ab}{4} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (1)$$

$$Y_D = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} W_D \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} / 2 \quad (2)$$

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} W_D \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (3)$$

Considering the equations (1), (2) and (3) for the three transforms, the respective algorithms were extracted and they were based in shifts and adders, making simpler their mapping to hardware.

3. Designed Architecture

This work presents a design of a T module dedicated to the Intra Prediction. This module is composed by the three transforms previously presented and by two buffers. Two main design decisions were done to achieve the best possible relation between throughput and latency: (1) All the transforms were designed with the maximum parallelism allowed (targeting high throughput); (2) The three transforms were designed with only one pipeline stage, processing till 16 samples in only one clock cycle (targeting low latency). Fig. 2 shows the block diagram for this architecture.

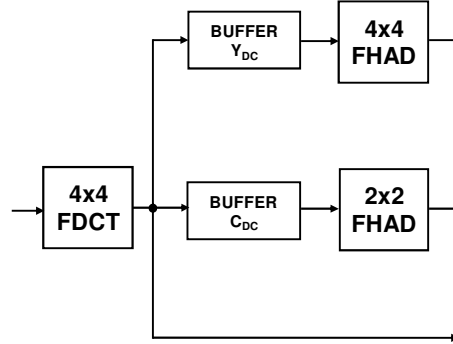


Fig. 2 – T module Block Diagram.

The buffers (Y_{DC} and C_{DC}) shown in fig. 2 are ping pong buffers and they are used to allow the synchronization among these transforms. These ping pong buffers are used to storage the DC coefficients when necessary. The Buffer Y_{DC} stores 16 luma samples, when the Intra Prediction mode is intra 16x16, and the Buffer C_{DC} stores four chroma samples when the Intra Prediction mode is chroma 8x8. Both buffers consume one sample per cycle. When the Intra Prediction mode is intra 4x4 for luma samples, the buffers are not used, since all samples are processed only by the 4x4 FDCT. In this case, one block is processed by the T module in only one clock cycle.

Considering intra 16x16 mode for luma samples, the data is divided over the data path. At first, each 4x4 block of the whole 16x16 luma block is processed by the 4x4 FDCT. After that, the AC coefficients are sent to the output. Meanwhile, each output block of the 4x4 FDCT has its DC coefficient stored in the Y_{DC} buffer, as shown in fig. 3 (a). As soon as the Y_{DC} buffer is filled with 16 DC coefficients, this 4x4 block of DC

coefficients can be processed by the 4x4 FHAD and, finally, sent to the output. In this case, the T module processes a 16x16 luma block in 17 clock cycles.

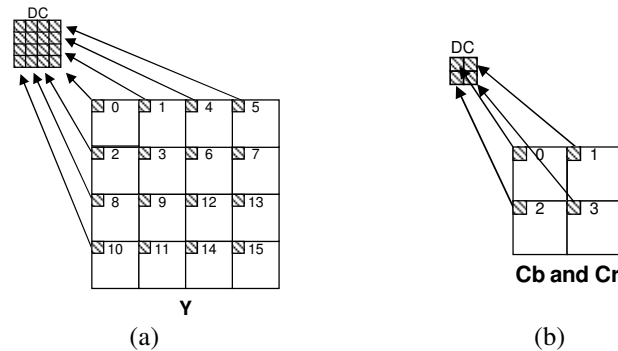


Fig. 3 – Samples processed by the T module.

The Intra Prediction for chroma samples is always performed in the 8x8 mode. In this case, the data flow is similar to that used in the intra 16x16 for luma samples. The difference is that only four DC coefficients are stored in the Buffer C_{DC} , since there are only four 4x4 chroma blocks in one macroblock, as shown in fig. 3 (b). Then the DC coefficients are processed by the 2x2 FHAD transform and they are sent to the output. Therefore, the time to process two 8x8 chroma blocks (Cb and Cr) is 10 clock cycles, five for each 8x8 block.

The low latency for all Intra Prediction modes is a very important characteristic in the T module, since it will define the amount of time that the Intra Prediction stays idle waiting for reconstruction blocks.

4. Synthesis Results and Related Works

The designed architecture was described in VHDL and synthesized for two different technologies: Altera Stratix II EP2X60F102C3 FPGA [5], using Quartus II tool and TSMC 0.18 μ standard-cell library [6], using Leonardo Spectrum tool. The architecture was validated using Mentor Graphics ModelSim.

Tab. 1 shows the synthesis results for each internal module and for the whole T module, considering operation frequency and hardware consumption (number of ALUTs and DLRs for FPGA and number of gates for standard cells).

Tab. 1 Synthesis Results.

Module	Stratix II FPGA			TSMC 0.18 μ m	
	Freq. (MHz)	#ALUTs	#DLRs	Freq. (MHz)	#Gates
4x4 FDCT	176.0	748	361	235.8	5.5k
4x4 FHAD	159.1	1,129	528	182.3	10.4k
2x2 HAD	222.0	224	126	223.3	2.1k
T module	149.4	1919	1151	199.5	17.5k

The T module architecture achieved 199.5 MHz as maximum operation frequency when mapped to standard-cells. Considering the worst case, i.e., when intra 16x16 mode is always chosen, one QHDTV frame (3840x2048 pixels) takes 829,440 clock cycles to be processed by the T module. Then, this architecture is able to process 239 QHDTV frames per second in the ideal case (when Intra Prediction is able to deliver one 4x4 block per cycle). In a real case, the inputs are delivered in bursts by the Intra Prediction module.

Considering the hardware consumption, the 4x4 FHAD architecture consumes more hardware than the others two transforms. It was expected when the comparison is done with 2x2 HAD, because this transform works with 2x2 blocks. The 4x4 FHAD consumes more hardware than 4x4 FDCT because the dynamic range grows through the data path to avoid overflows in the operations.

There are few works which implement a dedicated hardware designs for H.264/AVC T module, like [7] and [8]. The solution [7] also implements all transforms with the maximum parallelism allowed. However, four pipeline stages were used for the 4x4 FDCT and 4x4 FHAD and two pipeline stages for the 2x2 FHAD. The solution presented in [8] also was designed with four pipeline stages for the 4x4 FDCT and 4x4 FHAD and two pipeline stages for the 2x2 FHAD, but the parallelism level for each transform is of one sample per cycle.

The comparison was made considering: the parallelism level, the macroblock latency for intra 4x4 and intra 16x16, since this is a very important restriction for the Intra Prediction module, and throughput achieved by all solutions. To realize a fairer comparison with these works, we used the Stratix II FPGA results. Tab. 2 shows this comparison.

Tab. 2 Related works.

Solution	Technology	// Level	Macroblock Latency intra 4x4 (clock cycles)	Macroblock Latency intra 16x16 (clock cycles)	Throughput (Msamples /s)
Our	Stratix II	16	26	27	2,390.6
Porto [7]	Virtex II Pro	16	76	31	4,856.6
Agostini [8]	Virtex II Pro	1	1160	440	126.1

This work shows better results when compared with [8]. As the architecture proposed by [8] consumes only one sample per second, its macroblock latency is extremely high, 44.6 times higher for intra 4x4 and 16.2 times higher for intra 16x16, when compared with our work. Considering throughput, the architecture designed in [8] is 18.9 times lesser than ours. These facts decrease the Intra Prediction performance for all modes.

The only solution founded in the literature that presents a parallel T module is [7]. The throughput shown in this work is the highest than all works. It occurs because, as mentioned before, all transforms in that solution were designed with a higher number of pipelines stages, decreasing the critical path. For the Intra Prediction restrictions, architectures with many pipeline stages are not desirable, since it increases the macroblock latency, limiting the Intra Prediction performance. Then, even with the throughput about two times higher when compared with our work, the architecture designed in [7] presents macroblock latency for intra 4x4 three times higher than our solution.

5. Conclusions and Future Works

This paper presented an architecture for a T module dedicated to the Intra Prediction of the main profile of the H.264/AVC coder. The complete architecture for the T module was presented and the synthesis results were compared with related works.

The presented architecture was described in VHDL and synthesized to the Altera Stratix II EP2S60F1020C3 FPGA and to the TSMC 0.18 μ m CMOS standard-cells. The architecture was validated using Mentor Graphics ModelSim. The architecture achieved a maximum operation frequency of 199.5 MHz, processing about 239 QHDTV frames per second. Combined with the high throughput, the architecture presents a very low latency. The latency for luma samples is only one clock cycle for intra 4x4 and 17 clock cycles for intra 16x16. The latency for chroma samples is 10 clock cycles. Besides, the designed architecture achieved the best results for the Intra Prediction restrictions when compared with all related works. The designed architecture is able to be used in H.264/AVC encoder reaching real time (30 fps) when high resolution videos are processed.

As future works, we plan to integrate and validate the others modules of the T/Q/IQ/IT loop, that are being designed, with the Intra Prediction module.

6. References

- [1] ITU-T Recommendation H.264/AVC (05/03): advanced video coding for generic audiovisual services. 2003.
- [2] Brazilian Forum of Digital Television. ISDTV Standard. Draft. Dec. 2006 (in Portuguese).
- [3] T. Wiegand, et al, "Overview of the H.264/AVC Video Coding Standard", IEEE Trans. On Circuits and Systems for Video Technology, v. 13, n.7, pp. 560-576, 2003.
- [4] I. Richardson, *H.264 and MPEG-4 video compression – Video Coding for Next-Generation Multimedia*. John Wiley&Sons, Chichester, 2003.
- [5] Altera Corporation. "Altera: The Programmable Solutions Company". Available at: www.altera.com.
- [6] Artisan Components. *TSMC 0.18 μ m 1.8-Volt SAGE-XTM Standard Cell Library Databook*. 2001.
- [7] R. Porto et al., "High Throughput Architecture For Forward Transforms Module of H.264/AVC Video Coding Standard", in *IEEE International Conference on Electronics Circuits and Systems Conf. (ICECS 2007)*, 2007. pp. 150-153.
- [8] L. Agostini et al., "High Throughput Architecture for H.264/AVC Forward Transforms Block", in *16th ACM Great Lake Symposium on VLSI conf (GLSVLSI)*, 2006.

Dedicated Architecture for the T/Q/IQ/IT Loop Focusing the H.264/AVC Intra Prediction

Felipe Sampaio, Daniel Palomino, Robson Dornelles, Luciano Agostini
{fsampaio.ifm, danielp.ifm, rdornelles.ifm, agostini}@ufpel.edu.br

Grupo de Arquiteturas e Circuitos Integrados – GACI
Universidade Federal de Pelotas – UFPel

Abstract

This paper presents a low latency and high throughput architecture for the Transforms and Quantization loop to be employed in the critical path of the Intra Prediction module. The main goal is to reduce the Intra Prediction waiting time for new references to code the next block. The architecture was described in VHDL and synthesized targeting two different technologies: Altera Stratix II FPGA and TSMC 0.18 μm Standard Cell. The results show that the architecture achieves 129.3 MHz as maximum operation frequency. Considering the H.264/AVC main profile, the architecture is able to process 48 QHDTV frames per second when mapped to Standard Cell technology. The architecture reaches the best results in latency when compared with other published works in the literature. These results show that the designed architecture can easily reach real time (30 frames per second) when high resolution videos are being processed.

1. Introduction

The H.264/AVC [1] is the newest video compression standard. It was defined intending to double the compression rates when compared with previous standards, like MPEG-2. This work is inserted in the academic effort to design hardware solutions for the Brazilian System of Digital Television broadcast (SBTDV).

The main modules presented in a H.264/AVC encoder are: Inter Frame Prediction, composed by the Motion Compensation (MC) and the Motion Estimation (ME), Intra Frame Prediction, Deblocking Filter, Entropy Coding, Forward and Inverse Quantization (Q and IQ) and Forward and Inverse Transforms (T and IT). The Q, IQ, T and IT modules are the main focus of this work.

The Intra Prediction explores the spatial redundancy presented in a frame. This module works coding the current block using as reference previously coded blocks in the same frame [2]. To be used as reference, the residues of a block must pass through the T, IT, IQ and Q modules, generating the “reconstructed blocks”. Therefore, the number of clock cycles spent in these modules directly affects the Intra Prediction performance, since the Intra Prediction module must be idle, waiting for the reconstructed block while it is being processed by the T/Q/IQ/IT loop. This way, architectures designed focusing this reconstruction path must have low latency and high throughput.

This work is based on the H.264/AVC main profile [1], which defines a color relation of 4:2:0, i.e., one macroblock is composed by 16x16 luma (Y) samples, 8x8 chroma blue (Cb) samples and 4x4 chroma red (Cr) samples. The Intra Prediction works in two different modes for luma samples (intra 4x4 and intra 16x16) and only one mode for chroma samples (chroma 8x8).

Fig. 1 shows the H.264/AVC block diagram, where the designed T/Q/IQ/IT loop is highlighted. This dedicated loop was designed to reduce the impact of these modules in the Intra Prediction process. For this reason, this loop was designed intending to maximize the relation between throughput and latency.

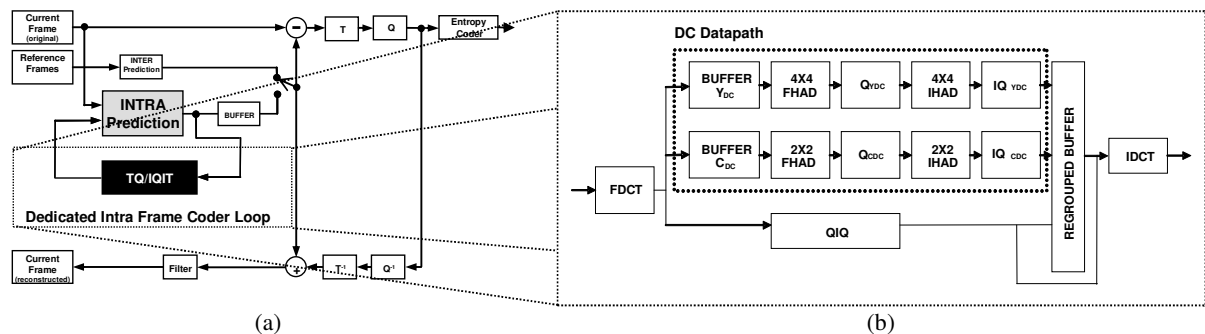


Fig. 1 – (a) H.264/AVC Block Diagram with the designed loop and (b) the proposed T/Q/IQ/IT loop.

The forward and inverse transforms defined in the H.264/AVC main profile are: Forward and Inverse 4x4 DCT (FDCT and IDCT), Forward and Inverse 4x4 Hadamard (4x4 FHAD and 4x4 IHAD) and 2x2 Hadamard

(2x2 HAD). The Equation (1) represents the calculation of the 4x4 FDCT transform. The multiplication by Ef is performed in the quantization process [1].

$$Y = C_f X C_f^T \otimes E_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 2 & -1 \end{bmatrix} \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (1)$$

The quantization calculation is different accordingly with the target sample. Generically, this process is controlled by the Quantization Parameter (QP), which is generated by the global encoder control and defines the quantization step that must be used [2]. Equations (2) and (3) present the forward and inverse quantization for all luma and chroma AC coefficients and luma DC coefficients that were not predicted in the intra 16x16 mode.

$$\begin{aligned} |Z_{(i,j)}| &= (|W_{(i,j)}| \cdot MF + f) \gg qbits \\ sign(Z_{(i,j)}) &= sign(W_{(i,j)}) \end{aligned} \quad (2)$$

$$W'_{(i,j)} = Z_{(i,j)} \cdot V_{(i,j)} \cdot 2^{\lfloor QP/6 \rfloor} \quad (3)$$

All the other transforms and quantization definitions are very similar and they are presented in [2].

The paper is organized as follows: Section 2 describes the designed architecture, Section 3 shows the pipeline scheme used in the architecture design, Section 4 presents synthesis results and the comparison with other published works and, finally, Section 5 concludes this paper and proposes future works.

2. Designed Architecture

This work presents an architectural design for the T/Q/IQ/IT loop with low latency and high throughput focusing the Intra Prediction constrains. Fig 1b shows the block diagram of this architecture.

In order to achieve the best possible relation between throughput and latency, two main decisions in the architectural design were done: (a) all the transforms and quantization were designed with the maximum parallelism allowed (targeting high throughput) and (b) the pipeline stages were balanced and almost all modules presented in Fig 1b represents one pipeline stage in the final design (focusing low latency and high throughput). The following subsections will explain more details about the loop implementation.

2.1. Forward and Inverse Transforms

All the architectures designed in this work were implemented with only one pipeline stage and with a parallelism level as high as possible. Then, the complete transform of one block is done in a single clock cycle. It means that there are four operators in the critical path for the 4x4 architectures and two operators for the 2x2 architecture. These operators are dedicated to perform a specific operation (addition or subtraction), decreasing the control unit complexity.

2.2. Forward and Inverse Quantization

The quantization process was divided in five different modules: the QIQ module for all luma and chroma AC coefficients and for the luma DC coefficients that were not predicted in the intra 16x16 mode, the Q_{YDC} and IQ_{YDC} modules that process the luma DC coefficients when the intra 16x16 mode is used, and, finally, the Q_{CDC} and IQ_{CDC} modules for all chroma DC coefficients.

The most complex operations in these modules are multiplications by some constants. These multiplications were decomposed in shift and add operations, decreasing the critical path when compared with a traditional matrix multiplication. In order to simplify the QIQ module, which joins the forward and the inverse quantization in a single module, the equations (2) and (3) were grouped in a single expression and some optimizations in the architecture were done.

2.3. Buffers

Two buffers were designed to store the DC coefficients that were processed by the FDCT: the Luma DC Buffer and the Chroma DC Buffer. They are used for luma blocks predicted in the intra 16x16 mode and for all chroma blocks, respectively. They are ping-pong buffers and they store one sample per cycle. The buffer groups the DC samples of the 16x16 luma block (or 8x8 chroma block) in order to process these samples in the DC datapath.

The Regrouped Buffer was designed to regroup the DC and the AC coefficients of the blocks when the Intra Prediction mode is intra 16x16. This buffer maintains the AC samples that were processed by the QIQ module, while the DC samples are being processed by the DC datapath. Thus, when the DC samples are ready, the whole regrouped 4x4 blocks are sent to the IDCT.

3. Pipeline Schedule

Fig. 2 presents the timing diagrams for all Intra Prediction modes defined in the H.264/AVC main profile: the (a) intra 16x16 and (b) intra 4x4 for luma samples and the (c) chroma 8x8 for chroma samples.

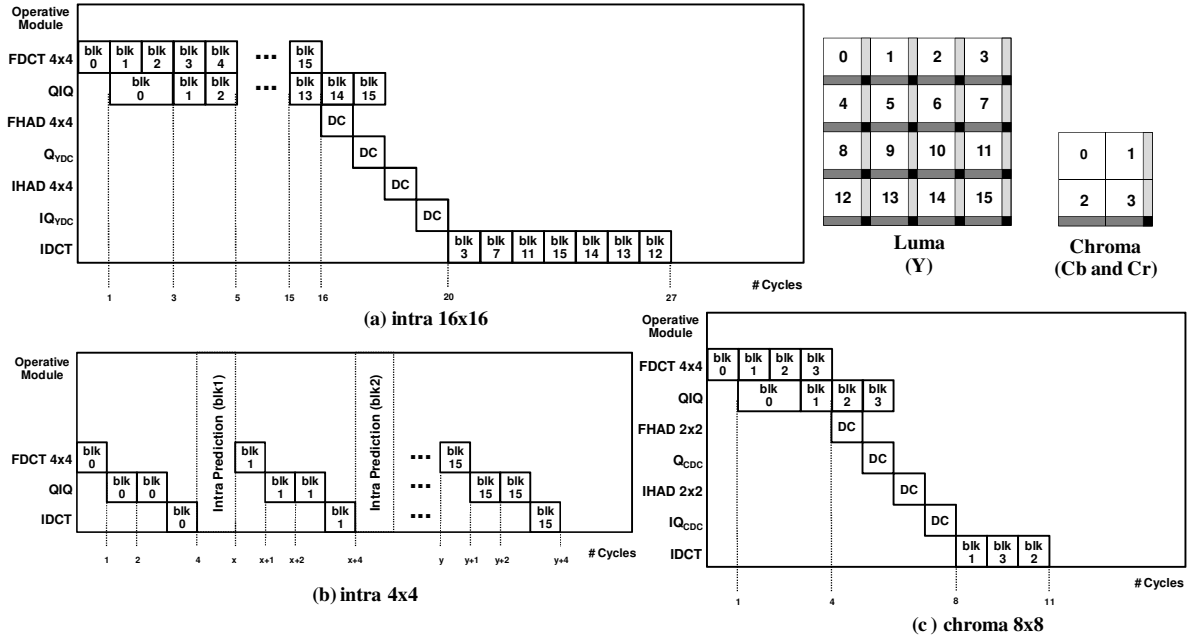


Fig. 2 – Timing diagram for all Intra Prediction modes: (a) intra 16x16, (b) intra 4x4 and (c) chroma 8x8.

Considering the intra 16x16 mode, the data flow is divided over the data path. In the first place, the whole 4x4 block is processed by the 4x4 FDCT. Then, the AC coefficients are processed by the QIQ module and they are stored in the Regrouped Buffer. Meanwhile, each output block of the 4x4 FDCT has its DC coefficient stored in the Y_{DC} buffer. As soon as the Y_{DC} buffer is full with the sixteen DC coefficients, this new block composed only by the DC coefficients is able to be processed by the 4x4 FHAD, Q_{YDC} , 4x4 IHAD and IQ_{YDC} . Finally, the DC block can be stored in the Regrouped Buffer. At this moment, the AC coefficients are already ready to be regrouped with their respective DC coefficients. At last, the block edges are processed by the 4x4 IDCT, since only these samples are needed to predict the next block by the Intra Prediction [1]. This data flow is represented by the timing diagram in fig. 2a. In this case, the T/Q/IQ/IT loop takes 27 clock cycles to deliver all the reconstructed luma edge samples for the Intra Prediction.

When the prediction mode is intra 4x4, the data processing is performed by the following steps: (1) the block is processed by the 4x4 FDCT, (2) QIQ operation is applied and (3) 4x4 IDCT is performed. In this case, both AC and DC coefficients are processed by these operations. This way, neither the DC path nor the Regrouped Buffer are used. The timing diagram in fig. 2b represents this operation flow. In this case, the Intra Prediction must be idle only four clock cycles.

For chroma samples, the Intra Prediction is always performed in the chroma 8x8 mode [2]. Then, the data flow is similar to that used in the intra 16x16 mode for luma samples. The difference is that only four DC coefficients are stored in the C_{DC} Buffer, since there are only four 4x4 chroma blocks in an 8x8 chroma block. Then, the DC coefficients are processed by the 2x2 FHAD, Q_{CDC} , 2x2 IHAD and IQ_{CDC} . After that, they are ready to be stored in the Regrouped Buffer. The timing diagram in fig. 2c shows that are needed 22 clock cycles to generate all edge samples of chroma, 11 cycles for each chroma component.

4. Synthesis Results and Related Works

The designed architecture were described in VHDL and synthesized for two different technologies: EP2S60F1020C3 Altera Stratix II FPGA and TSMC 0.18 μ m Standard Cell library. The architecture was validated using the Mentor Graphics Modelsim tool. Tab. 1 presents the synthesis results.

Considering the worst case, i.e., when the intra 4x4 mode is always chosen for luma, one QHDTV frame (3840x2048 pixels) takes 2,641,920 clock cycles to be processed and one HDTV frame (1920x1080 pixels) is wholly processed in 699,600 cycles. This way, the architecture can process 48 QHDTV frames or 186 HDTV frames per second in the ideal case (when Intra Prediction is able to deliver one 4x4 block per clock cycle). In a real situation, the inputs are delivered in bursts by the Intra Prediction module. Then, considering the integration of the T/Q/IQ/IT loop and the Intra Prediction, the Intra Prediction module can use until 3,435,300 clock cycles (at the same frequency of the loop) to generate the prediction of one HDTV frame. In this case, the integrated

module will be able to reach real time (30 fps). Considering the first results of an Intra Prediction module designed in our group, the number of cycles available to the Intra Prediction is really enough.

Tab. 1 - Synthesis Results

Module	Stratix II FPGA			TSMC 0.18 μ m	
	Freq. (MHz)	#ALUT	#Register	Freq. (MHz)	#Gate
FDCT	176.0	748	361	235.8	5.5k
4x4 FHAD	159.1	1,129	528	182.3	10.4k
4x4 IDCT	154.3	1,445	684	180.0	8.6k
4x4 IHAD	123.8	1,919	689	143.1	16.9k
2x2 HAD	222.0	224	126	223.3	2.1k
QIQ	117.1	5,612	888	125.9	49.9k
Q_{CDC}	103.1	1,417	220	136.9	11.6k
Q_{YDC}	103.1	5,950	608	121.0	47.5k
IQ_{CDC}	185.8	2,537	640	161.1	4.0k
IQ_{YDC}	174.3	2,647	624	134.9	21.0k
Control Unit	217.8	146	73	335.0	0.8k
Loop T/Q/IQ/IT	98.5	9,903	3,445	129.3	210k

We did not find any published work that presents a dedicated architecture to the T/Q/IQ/IT loop. However, there are published works which implement the loop already integrated with the Intra Prediction module, like [3], [4] and [5]. These papers do not present more detailed synthesis results about the T/Q/IQ/IT loop, and this way, a more consistent comparison were not possible. But a comparison considering the latency of the T/Q/IQ/IT loop for each Intra Prediction mode was done and it is presented in tab. 2.

Tab. 2 - Latency comparison with related works

Mode	This Work (cycles)	Suh [3] (cycles)	Kuo [4] (cycles)	Hamzaoglu [5] (cycles)
Intra 4x4	4	34	16	100
Intra 16x16	27	441	159	1,718
Chroma 8x8	22	242	77	-

In all Intra Prediction modes, the architecture designed in this work presents the lowest latency when compared with related works. In the worst case, our architecture presents a latency 3.5 times lesser than [4] for chroma 8x8 mode. In the best case, our architecture has a latency 63.6 times lesser than [5] for intra 16x16.

5. Conclusions and Future Works

This paper presented a low latency and high throughput architecture dedicated to the transforms and quantization loop of the H.264/AVC main profile encoder, targeting the Intra Prediction module constraints. The presented architecture were described in VHDL and synthesized to the Altera Stratix II EP2S60F1020C3 FPGA and to TSMC 0.18 μ m CMOS standard cells. The architecture achieved a maximum operation frequency of 129.3 MHz, processing about 48 QHDTV or 186 HDTV frames per second. Combined with this high throughput, the architecture presents a very low latency. The latency for luma samples are four cycles for the intra 4x4 mode and 27 cycles for intra 16x16 mode. The latency for chroma is 22 clock cycles. This fact highly increases the Intra Prediction performance, since it decreases the waiting time for new references to perform the prediction of the next block. The obtained results show that our dedicated loop presents the lowest latency among all related works.

As future works, it is planned the integration of the designed loop with an Intra Prediction module. This way, other optimizations can be performed and better results can be achieved.

6. References

- [1] ITU-T. ITU-T Recommendation H.264/AVC (05/03): Advanced video coding for generic audiovisual services. 2003.
- [2] I. Richardson, "H.264 and MPEG-4 Video Compression – Video Coding for the Next-Generation Multimedia". John Wiley&Sons, Chichester. 2003.
- [3] K. Suh, et al, "An Efficient Hardware Architecture of Intra Prediction and TQ/IQ/IT Module for H.264 Encoder", ETRI journal, 2005, vol. 27, n5, pp. 511-524.
- [4] H. Kuo et al, "An H.264/AVC Full-Mode Intra-Frame Encoder for HD1080 Video", IEEE International Conference on Multimedia & Expo, 2008, pp.1037-1040.
- [5] I. Hamzaoglu et al, "An Efficient H.264/AVC Intra Frame Coder System Design" IFIP/IEEE Int. Conference on Very Large Scale Integration, 2007, vol. 54, pp. 1903-1911.

A Real Time H.264/AVC Main Profile Intra Frame Prediction Hardware Architecture for High Definition Video Coding

¹Cláudio Machado Diniz, ¹Bruno Zatt, ²Luciano Volcan Agostini, ¹Sergio Bampi,
¹Altamiro Amadeu Susin

{cmdiniz, bzatt, bampi, susin}@inf.ufrgs.br, agostini@ufpel.edu.br

¹GME – PGMICRO – PPGC – Instituto de Informática
Universidade Federal do Rio Grande do Sul - UFRGS
Porto Alegre, RS, Brasil

²Grupo de Arquiteturas e Circuitos Integrados – GACI – DINFO
Universidade Federal de Pelotas - UFPel
Pelotas, RS, Brasil

Abstract

This work presents an intra frame prediction hardware architecture for H.264/AVC main profile encoder which performs high definition video coding in real time. It is achieved by exploring the parallelism of intra prediction and by reducing the latency for Intra 4x4 processing, which is the intra encoding bottleneck. Synthesis results on Xilinx Virtex-II Pro FPGA and TSMC 0.18 μ m standard-cells indicate that this architecture is able to real time encode HDTV 1080p video operating at 110 MHz. Our architecture can encode HD1080p, 720p and SD video in real time at a frequency 25% lower when compared to similar works.

1. Introduction

H.264/AVC [1] is the latest video coding standard developed by the Joint Video Team (JVT), assembled by the ITU Video Coding Experts Group (VCEG) and the ISO/IEC Moving Pictures Experts Group (MPEG). Intra frame prediction is a new feature introduced in the H.264/AVC to reduce the spatial redundancy present in a frame. A block is predicted using spatial neighbor samples already processed in the frame. Due to block-level data dependency on the same frame, intra frame prediction introduces a high computational complexity and latency to H.264/AVC encoder design, mainly when encoding high definition videos in real time is needed. The work in [2] shows the compression gains and computational complexity overhead of H.264/AVC intra frame encoder when compared to JPEG and JPEG2000 encoders.

This work proposes an H.264/AVC intra frame prediction hardware architecture designed to reach real time encoding for HD 1080p (1920x1080) at 30 fps. The proposed architecture supports all intra prediction modes and block sizes defined by the standard and was designed to achieve high throughput, at the expense of an increase in hardware area. The similarity criterion used was SAD (Sum of Absolute Differences). This work is organized as follows. Section 2 introduces the basic concepts about H.264/AVC intra frame prediction. Section 3 presents the designed architecture for the intra frame prediction. Section 4 presents the synthesis results and comparisons with related works. Section 5 concludes the paper and comments on future work.

2. Intra Frame Prediction

H.264/AVC main profile encoder works over macroblocks (MB), composed by 16x16 luma samples, and two 8x8 Cb and Cr samples (4:2:0 color ratio). Two intra prediction types are defined for luma samples: Intra 16x16 (I16MB) or Intra 4x4 (I4MB). I16MB contains four prediction modes (horizontal, vertical, DC, plane) and is applied over the whole macroblock using 32 neighbor samples for prediction. In I4MB the MB is broken into sixteen 4x4 blocks that are individually predicted with nine prediction modes using up to 13 neighbor samples (A-M in Fig. 1) for prediction. The chroma prediction is applied always over 8x8 blocks and with the same I16MB modes. The same prediction mode is applied for both Cb and Cr.

Each mode defines different operations using neighbor samples to generate a predicted block. In vertical and horizontal modes, a block is predicted by copying the neighbor samples as indicated by the arrows. The DC mode calculates an average of neighbor samples and produces a block where all the samples are equal to this average. The I4MB diagonal modes produce some interpolation on the neighbor samples, e.g. $(A+2B+C+2)>>2$ [1], depending on the direction of the arrows in Fig. 1. The plane mode is the most complex because it needs to generate three parameters (a, b and c) to calculate the prediction for each sample of MB, as shown in (1). The position [y,x] in (1) represents a luma sample in the y row and x column (0..15) related to top-left sample in the block and P[y,x] are the predicted samples for I16MB plane mode.

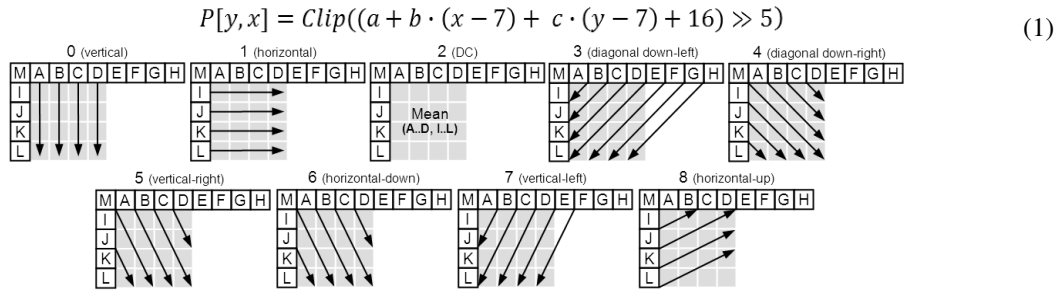


Fig. 1 – I4MB prediction modes.

3. Intra Frame Prediction Architecture

The proposed architecture was defined based on two main guidelines: i) to reach the required throughput for real time encoding of 1080p video sequences at 30 fps; ii) to use the smallest hardware area that provides the desired performance. Fig. 2 shows the architecture, which is divided into four main modules: i) Intra Neighbors (IN): manages the neighbor samples memory and controls other modules; ii) Sample Predictor (SP): proceeds the interpolations over the neighbor samples to generate the predicted samples; iii) SAD Calculator and I4MB Mode Decision (SADI4): calculates the SAD between the predicted block and the original block and compares the SADs of each prediction mode selecting the best mode; iv) Prediction Memory (PM): stores the predicted samples to eliminate the need of recalculation. The architecture is implemented in a pipeline way and it performs the prediction based on the interleaved partition schedule in [2]. It is formed by only one datapath for I4MB, I16MB and chroma 8x8. During the process of an I4MB block by T/Q/IT/IQ modules, I16MB and chroma partitions (4x4 block) can be processed in parallel. This approach forces the use of buffers to store the predicted samples while the final mode decision has not been taken (the Prediction Memory). Next sections will detail the four main modules of the architecture.

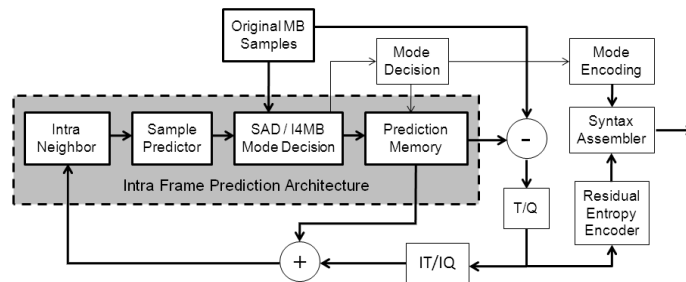


Fig. 2 – Proposed intra frame prediction hardware architecture.

3.1. Intra Neighbors (IN)

This module manages neighbor information and controls the entire coding process. It is composed by a neighbor memory, an ASM (algorithmic state machine) and a FSM. The neighbor memory stores one frame line of samples to be later referenced as the north (up) neighbors for the other blocks prediction. Every time a MB is processed, the up neighbor samples are replaced by its own bottommost reconstructed samples. This memory must be sized to fit the line size, so for a 1080p video it must store 3840 samples (1920 for luma and 960*2 for chroma). This approach requires 30,720 memory bits but reduces extra-chip frame memory bandwidth.

The ASM is responsible to read the neighbor memory, to calculate the DC values and the plane parameters (a , b and c) and to control the encoding process. This machine was described in 44 states where the control signals are generated. The FSM (13 states) stores the reconstructed samples back to the neighbor memory.

3.2. Sample Predictor (SP)

Sample Predictor generates the predicted samples for the I4MB blocks, I16MB partitions and chroma 8x8 blocks, depending on the operation mode. It works over 4x4 partitions and generates 4 samples per clock cycle for up to 9 prediction modes. In this way, after 4 cycles, a predicted 4x4 block is done for all modes in parallel. Horizontal and vertical modes (0-1) are a copy of neighbor samples and do not require operators to do the prediction. I4MB diagonal modes (3-8) are generated using three types of filters. I4MB modes share many operations with the same input neighbor samples, so it is possible to generate all diagonal prediction modes in

parallel using only 26 adders. Multiplexers were placed after the filters to select only the four output samples depending on the block line (0, 1, 2 or 3) to be predicted.

A simple FSM controls mux signals. DC levels and plane mode parameters are generated by the IN module and the sample predictor only performs the equation (1). For ASIC implementation, multipliers in (1) were eliminated. Two plane filters, three adders, a shift and a clip operation were used to perform the plane mode for one sample/cycle (Fig. 3a). This hardware is replicated four times to generate all predicted samples.

3.3. SAD Calculation / 4x4 Mode Decision (SADI4)

The SAD Calculation module, presented in Fig. 3b, is composed by 9 PEs (processing elements) which subtract the four predicted samples from the original samples generating the difference among each sample. These four differences are accumulated by an adder tree (inside each PE) to generate the SAD for one predicted 4x4 block line. To accumulate the SAD for the entire 4x4 block, there is one additional adder and one accumulator register. The SAD calculator works in pipeline with the Sample Predictor, so that after the first predicted line is generated, the SAD calculator starts to accumulate the SAD during four consecutive cycles. This module was designed in three pipeline stages to reduce the combinational delay, although the temporal barriers are omitted in Fig. 3b. Nine PE were instantiated to calculate the SAD of the nine I4MB modes or the four I16MB plus the four chroma modes in parallel. I4MB Mode Decision is also performed and consists in comparators tree that receive SAD and neighbor information as input and returns the best I4MB mode.

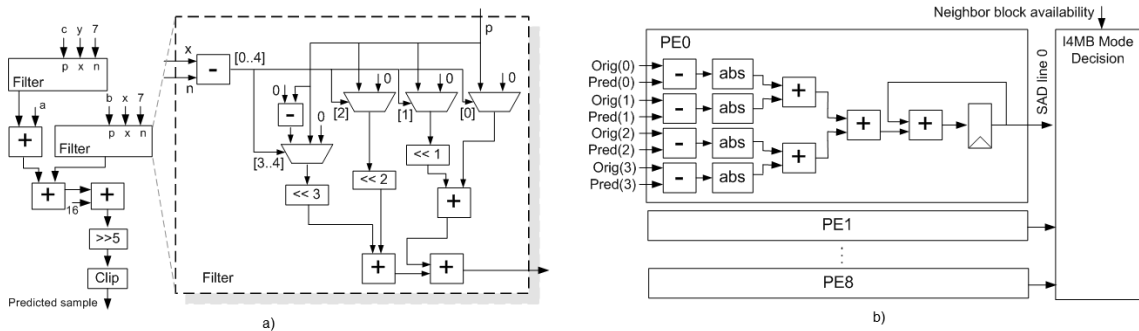


Fig. 3 – a) Processing filter for I16MB plane mode; b) SAD calculation and 4x4 mode decision modules.

3.4. Prediction Memory (PM)

The output predicted samples are required in three different moments in the encoding process: i) SAD calculation; ii) generation of entropy residue and iii) reconstruction of the reference samples. However, in the second moment these samples are not available anymore, so two solutions can be used: insert a unit to recalculate the prediction for the chosen mode or insert a memory to keep the predicted samples available for more time. The second solution was adopted to spend fewer cycles.

The Prediction Memory is organized as follows: the 4 first lines store the nine I4MB modes for one block. Next 64 lines store I16MB and chroma modes, first Cb and then Cr. In each cycle one line of all prediction modes is written totalizing 36 samples per cycle. It was used a 4-entry 288-bit dual-port SRAM memory for I4MB modes and a 64-entry 256-bit dual-port SRAM memory for I16MB and chroma 8x8 modes.

4. Results and Comparisons

The architecture was verified using our H.264/AVC Intra Frame Encoder SystemC model [3], using Mentor Graphics ModelSim, and synthesized to Xilinx Virtex-II Pro XC2VP30 FPGA using Xilinx ISE 8.1i. Tab. 1 shows the results. Our target FPGA device implements 18x18-bit multipliers, so we implement 10 multipliers of the plane mode in the sample predictor (SP) using macro-functions. Neighbor and Prediction memories used 30,720 and 13,440 memory bits, respectively, and occupy 8% of FPGA BRAMs. The complete module occupies only 29% of slices and can achieve a frequency of 112.8 MHz in XC2VP30. The ASIC version (in TSMC 0.18 μ m, using Leonardo Spectrum) achieves a maximum frequency of 156 MHz and occupies 42,737 gates. SRAM memories are suppressed and the plane mode multipliers are implemented as (Fig. 3a) to reduce area and critical path.

The proposed architecture supports all main profile intra prediction modes, calculates the SAD cost and decides for the best I4MB mode in 15 cycles. The architecture does not implement T/Q/IT/IQ loop, but considers a loop delay of 7 cycles, and it implements all block reconstruction control. During this process, I16MB partitions are processed, so the prediction for a MB is performed in 355 cycles, if the decision for I4MB partitioning is taken. If I16MB partitioning is decided, 96 extra cycles are necessary to process the MB by the T/Q/IT/IQ loop. This case does not occur on I4MB mode because the MB has been already processed by the T/Q/IT/IQ loop. I4/I16 decision is an encoder issue and was not considered in this paper. The throughput

needed to process HD 1080p 4:2:0 at 30 fps is 243 Kmacroblocks/s. The FPGA and ASIC versions achieved throughputs of 248 and 344 Kmacroblocks/s, working at 112.8 MHz and 156 MHz, respectively. Therefore, the proposed architecture reaches the necessary throughput to process HD 1080p in real time, considering the worst case. It is able to real-time encode HD 1080p running at 110 MHz, 720p at 48 MHz and SD at 18 MHz.

Comparing with related works [2, 4-6], our design can achieve performance for real time processing of different video resolutions (1080p, 720p, SD) at a lower operation frequency. The work in [2], in TSMC 0.25 μ m, can encode only SD (720x480) 4:2:0 at 30 fps, operating at 55 MHz. The work in [4] (tab. 2) is able to process 720p (1280x720) 4:2:0 at 30 fps, working at 117 MHz but do not implement I16MB plane mode, which is the most complex operation. The work in [5] defines three different quality levels using fast algorithm techniques and achieves performance for 720p operating at 70 MHz for the worst quality and 85 MHz for intermediary quality. No frequency data for the better quality level is presented in [5] for 720p resolution. Our design implements the original intra prediction algorithm and performs all modes (full-mode). The work in [6] is the only which achieves performance for 1080p, but with an increase of the operating frequency by 25% when compared with our architecture. The gate count (in tab. 2) considers only the intra predictor for the proposed work and [4, 5]. For [6] the whole intra coder gate count is considered. The work in [4] presents a much smaller gate count since it does not implement the complex I16MB plane mode (not full-mode).

Tab.1 – Synthesis Results

	IN	SP	SADI4	PM	Total	% Usage
Slices	2,231	793	660	193	4,102	29%
Flip Flops	2,049	326	603	314	3,293	12%
LUTs	3,965	1,469	1,071	71	7,330	26%
BRAM	3	-	-	8	11	8%
Multipliers	-	10	-	-	10	7%

Device: Xilinx Virtex-II Pro XC2VP30-7

Tab.2 – Comparisons With Related Works

	[4]	[5]	[6]	This work
Gate count	36 K	80 K	199 K	42 K
Technology	UMC 0.18 μ	TSMC 0.13 μ	TSMC 0.13 μ	TSMC 0.18 μ
Memory bits	9,728	5,120	60,000	44,160
Max. frequency	125 MHz	130 MHz	142 MHz	156 MHz
Full-mode	No	Yes	Yes	Yes
Freq. for real time	1080p	-	138 MHz	110 MHz
	720p	117 MHz	70/85 MHz	48 MHz
	SD	43 MHz	23 MHz	18 MHz

5. Conclusions

This work proposed an intra frame prediction hardware architecture for H.264/AVC coding. It supports all main profile intra modes, calculates the blocks costs and decides for the best I4MB prediction. It can process a macroblock in 451 cycles (worst case). The modules in VHDL were verified using our Intra Frame SystemC model [3] and synthesized for Xilinx Virtex-II FPGA and TSMC 0.18 μ m standard-cells. Synthesis results confirmed that the proposed architecture is able to process HD 1080p video at 30 fps when operating at 110 MHz for FPGA and ASIC platforms. This design achieved the highest throughput when comparing to other works reducing the required frequency of operation by 25%. The performance results are enough to integrate the architecture in a complete H.264/AVC compliant encoder, which is planned for future work.

6. References

- [1] ITU-T Recommendation H.264/AVC: advanced video coding for generic audiovisual services, Mar. 2005.
- [2] Y.-W. Huang, B.-Y. Hsieh, T.-C. Chen, and L.-G. Chen, "Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder", IEEE TSCVT, IEEE, Mar. 2005, p. 378-401.
- [3] B. Zatt, C. Diniz, L. Agostini, A. Susin, S. Bampi. "SystemC Modeling of an H.264/AVC Intra Frame Encoder Architecture", IFIP/IEEE VLSI-SoC, Oct. 2008.
- [4] C.-W. Ku; C.-C. Cheng, G.-S. Yu, M.-C. Tsai, T.-S. Chang, "A High-Definition H.264/AVC Intra frame Codec IP for Digital Video and Still Camera Applications", IEEE TCSVT, Aug. 2006, p. 917-928.
- [5] C.-H. Chang, J.-W. Chen, H.-C. Chang, Y.-C. Yang, J.-S. Wang, J.-I. Guo. "A Quality Scalable H.264/AVC Baseline Intra Encoder For High Definition Video Applications", IEEE SiPS, Oct. 2007, p. 521-526.
- [6] H.-C. Kuo, Y.-L. Lin. "An H.264/AVC Full-Mode Intra-Frame Encoder for 1080HD Video", IEEE ICME, Apr. 2008, p. 1037-1040.

A Novel Filtering Order for the H.264 Deblocking Filter and Its Hardware Design Targeting the SVC Interlayer Prediction

¹Guilherme Corrêa, ²Thaís Leal, ³Luís A. Cruz, ¹Luciano Agostini
{gcorrea_ifm, agostini}@ufpel.edu.br, tilsilva@inf.ufrgs.br, lcruz@deec.uc.pt

¹Grupo de Arquiteturas e Circuitos Integrados
Universidade Federal de Pelotas, Pelotas, Brasil

²Grupo de Microeletrônica
Universidade Federal do Rio Grande do Sul, Brasil

³Instituto de Telecomunicações
Universidade de Coimbra, Coimbra, Portugal

Abstract

This work presents a novel and efficient filtering order for the deblocking filter of the H.264 standard and an architectural design for the deblocking filter of the scalable version of this standard, using this new filtering order. The proposed filtering order enables a better exploration of the filter parallelism, decreasing the number of cycles used to filter the videos. Considering the same parallelism level, this new filtering order reduces in 25% the number of cycles used in the filtering process if compared with related works. The designed architecture is focused in the scalable version of the H.264 standard and targets high throughputs. Then, four concurrent filter cores were used, decreasing significantly the number of cycles used to filter the videos. The architecture was described in VHDL and synthesized for an Altera Stratix III FPGA device. It is able to filter at most 130 HDTV (1920x1080 pixels) frames per second.

1. Introduction

With the technological advances in video coding and with the continuous development of network infrastructures, storage media and computational power, the number of devices which support video applications have increased significantly. This fact leads to a problem related to the transmission and reception of videos, since the sent data must be decoded and displayed by different types of devices with different characteristics. One possible solution for this problem lies on the use of adaptation functions that convert the received videos to its visualization capabilities [1]. However, the use of these functions can increase the cost and the power consumption of the receiver. Another possible solution for the video provider would be the transmission of a multitude of coded representations of each video, one for each possible combination of target devices. This solution, however, presents another problem related to the waste in the data channel because the same information would be transmitted more than once.

Since both solutions had disadvantages, scalable coding techniques were defined in an extension of the H.264/AVC standard [2], here forth designated as H.264/SVC [3]. The general operating principle of scalable coding defines a hierarchy of layers where a base layer represents the video signal at a given minimum temporal, spatial and/or quality resolution, and additional layers are used to increase each of the resolutions.

Spatial scalability in H.264/SVC is performed using, among other operations, interlayer prediction whereby a higher layer image region is predicted from the corresponding lower layer area by a process of upsampling. In order to reduce the visible block-edge artifacts at the boundaries between blocks, caused by high quantization steps during the coding at the reference layer, H.264/SVC defines an inter-layer filtering procedure that is carried out across block boundaries, called deblocking filter. This filter is very similar to the deblocking filter used at the end of the coding/decoding process of the H.264/AVC without scalability, even though it performs a different calculation for the boundary strength.

This work presents a novel and optimized filtering order for the H.264 deblocking filter based on sample level operations. This novel filtering order can be applied to both the non scalable and scalable versions of the H.264 standard (AVC and SVC). The proposed order allows a better exploration of the parallelism level, reducing significantly the use of cycles to perform the filtering process in comparison with the previous solutions presented in the literature.

This new filtering order was applied to the deblocking filter of the SVC interlayer prediction and an architectural design for this filter was developed. The architecture was described in VHDL and synthesized for Altera Stratix III FPGAs.

This paper is structured as follows: in section 2 we outline the operation of the deblocking filter. Section 3 is devoted to the exposition of the filter ordering solutions published in the technical literature as well as our

novel solution. In section 4 the architecture of the proposed filter is presented. Section 5 reports on the results and compares them to related works. Section 6 concludes with a discussion about the current results and shows promising directions for future work.

2. Deblocking Filter

The H.264/AVC standard specifies a deblocking filter which is used in both the encoder and decoder in order to smooth the boundaries of the reference reconstructed block before using its pixels in the prediction of the current block. The deblocking filter of the scalable standard is similar to the non scalable version, although the boundary strength (bS) calculation performs different operations. Also, only the intra-coded blocks from the reference layers are filtered by this filter. This happens due to the single-loop concept explored in H.264/SVC [1], which defines that only the highest layer can perform motion compensation. Therefore, inter coded blocks are not filtered and the application of this filter is less expensive than in the case for the non-scalable one.

Both versions of the filter perform the same filtering operations, which are applied across the horizontal and vertical boundaries of the 4x4 blocks. The filtering operation occurs according to these steps: (a) filtering vertical edges of luminance macroblock (MB), (b) filtering horizontal edges of luminance MB, (c) filtering vertical edges of chrominance MBs and (d) filtering horizontal edges of chrominance MBs.

Each filtering operation modifies up to three pixels on each side of the edge and involves four pixels of each of the two neighboring blocks (p and q) that are filtered. The filter strength is adjustable and depends on the quantization step [1] used when the block was coded, on the coding mode of neighboring blocks and on the gradient of the values of the pixels across the edge which is being filtered. The H.264/SVC standard specifies four different strengths parameterized as bS which takes values 0, 1, 2 or 4 (0 standing for “no filtering” and 4 indicating maximum smoothing). The value of bS is chosen according to the following algorithm [1]:

```

If p or q belong to an intra macroblock (MB) different from I_BL
  Then bs = 4
If p and q belong to a intra MB of type I_BL
  Then
    If at least one of the transform coefficients of the blocks to which samples p0 and q0
    belong is  $\neq 0$ 
      Then bs = 1
      Else bs = 0
If p or q belong to an inter MB
  Then
    If p (or q) belong to a inter MB and the residues matrix rSL has at least one value  $\neq 0$ 
    in the blocks associated with sample p0 (or q0).
      Then bs = 2
      Else bs = 1

```

However, for values of $bs > 0$, filtering takes place only if condition (1) is met, where p_2, p_1, p_0, q_0, q_1 , and q_2 represent the samples on both sides of the edge and occurring in this order. In (1), α and β defined in the standard increase with the increasing of the quantizer step (QP) used in the blocks **p** and **q**. Then, if QP is small, so will be α and β and the filter will be applied. For more details please consult [2].

$$|p_0 - q_0| < \alpha \text{ and } |p_1 - p_0| < \beta \text{ and } |q_1 - q_0| \leq \beta \quad (1)$$

3. Proposed Filtering Order and Related Works

The only restriction imposed by H.264/AVC (and, also, by H.264/SVC) on the filtering order is that if a pixel is involved in vertical and horizontal filtering, then the latter should precede the former. This allows the exploration of different filtering scheduling aiming at faster operation through the use of parallelism or at solutions that use less memory. The filtering order proposed by H.264/AVC [2] is shown in fig. 1. In this order, since the results of vertical filtering are needed for the horizontal filtering, they have to be kept in memory making this solution quite costly in terms of memory usage.

Khurana proposes in [4] a different order, where horizontal and vertical filtering are applied alternately, as illustrated in fig. 2. Using this order, only a line of 4x4 blocks must be stored in memory to be used during the next edge filtering. A pixel can be written back to the main memory after to be filtered in both directions.

Still based on the alternation principle, Shen [5] proposed the order depicted in fig. 3, where a higher frequency of vertical-horizontal filtering direction change is observed, resulting in a decrease in the temporary memory size. Li[6] proposed another solution based on the alternation principle, but also involving a degree of parallelism with vertical and horizontal filtering, speeding up the filtering at the cost of having to use two filtering units. The scheduling for this solution is presented in fig. 4.

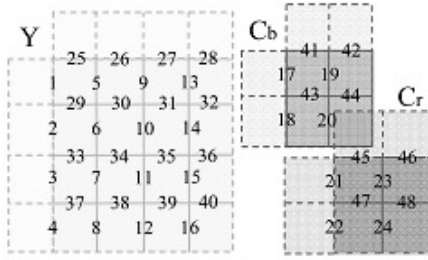


Fig. 1 – Filtering order of H.264/AVC [2]

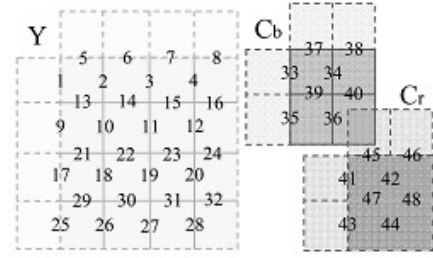


Fig. 2 – Alternate order of Khurana [4]

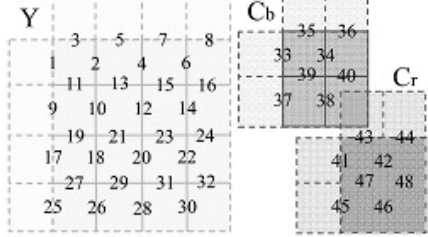


Fig. 3 – Shen's filtering order [5]

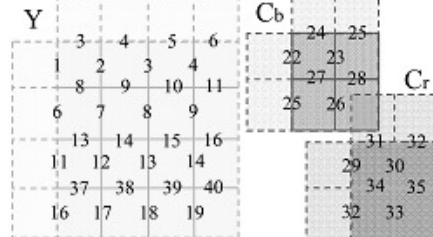


Fig. 4 – Li's filtering order [6]

All the filtering orders presented are performed in block level, i.e., the calculations of a border between two 4x4 blocks are performed serially by the same filter and each filtering between two blocks starts just when the filtering of all the LOPs (Line of Pixels) of the predecessor block (left) finishes. This work proposes a novel and efficient processing order in sample level, instead of block level. This way, it is possible to start the first LOP filtering of each block when the result of the filtering which involves the first LOP of the predecessor block is concluded.

The computation in sample level allows a better exploration of parallelism in the architecture, as will be shown in the next section. fig. 5 presents the proposed filtering order. The equal numbers correspond to the filtering occurring concurrently by different cores.

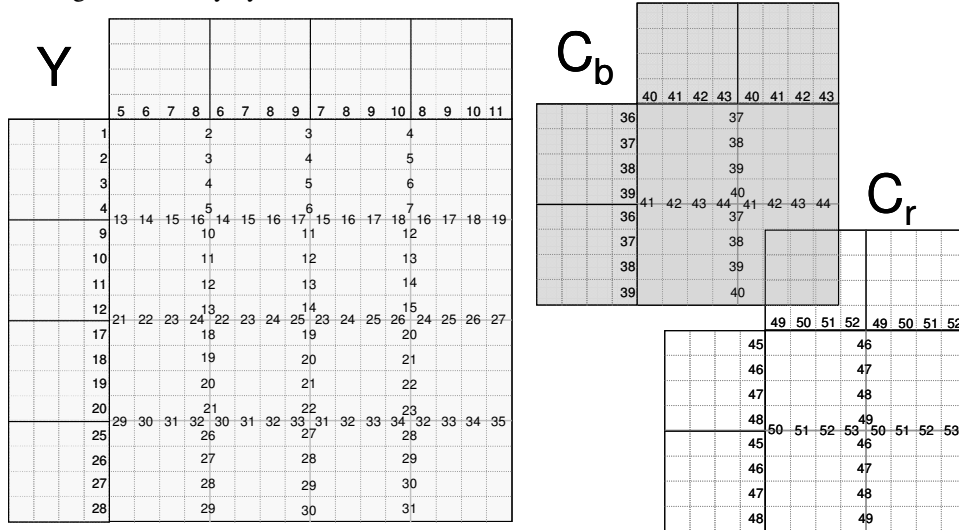


Fig. 5 – Proposed filtering order

4. Developed Architecture

The developed architecture uses four filter cores who share a bS computing unit as well as a threshold computing unit. Since filtering occurs along a LOP, which can be vertically or horizontally aligned, we would need a transposition operation to filter orthogonally to the pixel storage direction. Instead, we use transposition matrices to be able to access pixel memory in either direction. The filter architecture is composed by the following modules: eight transposition matrices, four filter cores, one bS calculation unit, a threshold calculator, a c1 calculator and a control unit.

The architecture flow is based on a pipeline structure which executes the filtering concurrently. This way, the filtering of each block consumes 7 cycles, even though a line of 4 4x4 blocks is filtered in 10 cycles, as shown in fig. 6. In the four beginning cycles, the four first LOPs of the four first blocks of a macroblock are written in the transpose matrices. In the second cycle, the threshold and bS calculations are also performed for the first border. In the third cycle, the threshold and the bS calculations are also performed for the second border

and, at the same time, the c1 calculation happens for the first border. From the fourth cycle on, the filtering occur in the first core and, in each one of the three following cycles, another filtering core is enabled.

cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6	cycle 7	cycle 8	cycle 9	cycle 10
read MB line 1 + coding info	calc. thresh. calc. bS	calc. c1	filter LOP 1 blocks L & 1	filter LOP 2 blocks L & 1	filter LOP 3 blocks L & 1	filter LOP 4 blocks L & 1			
	read MB line 2	calc. thresh. calc. bS	calc. c1	filter LOP 1 blocks 1 & 2	filter LOP 2 blocks 1 & 2	filter LOP 3 blocks 1 & 2	filter LOP 4 blocks 1 & 2		
		read MB line 3	calc. thresh. calc. bS	calc. c1	filter LOP 1 blocks 2 & 3	filter LOP 2 blocks 2 & 3	filter LOP 3 blocks 2 & 3	filter LOP 4 blocks 2 & 3	
			read MB line 4	calc. thresh. calc. bS	calc. c1	filter LOP 1 blocks 3 & 4	filter LOP 2 blocks 3 & 4	filter LOP 3 blocks 3 & 4	filter LOP 4 blocks 3 & 4

Fig. 6 – Execution flow for horizontal filtering (four first blocks)

5. Results and Comparisons

The architecture proposed in this paper takes 53 cycles to filter a complete macroblock, which is about 25% less than the best result of the previous solutions with the same number of filtering cores [7]. Tab. 1 shows other solutions concerning the number of cycles necessary to filter one macroblock and the size of temporary memory that would be used to implement each architecture. Even though only one of the other methods uses four filter cores, our proposal is faster and uses less memory. The architecture was described using VHDL and synthesized for Altera Stratix III FPGA (EP3SL50F484C2 device). The core filter uses 737 ALUTs and the complete architecture uses 7868 ALUTs. The architecture is able to run at around 270 MHz which allows the filtering of all blocks of a video with HDTV resolution (1920x1080) at a rate of 130 frames per second.

Tab.1 – Comparison between the number of cycles used in previous works

Author	# cycles	# cores	memory size
[2]	192	1	512 bytes
[4]	192	1	128 bytes
[5]	192	1	80 bytes
[6]	140	2	112 bytes
[7]	70	4	224 bytes
This work	53	4	128 bytes

6. Conclusions and Future Work

This work presented a competitive filtering order for the implementation of the deblocking filter of H.264 standard. Results showed its advantages over similar published designs, outperforming the best one by 25%. This paper has also described the architecture which performs the filtering according to the novel processing order proposed. The maximum operation frequency of the filter is around 270 MHz, which means that it is able to process 130 HDTV frames per second. These results satisfy and outperform the minimum requirements of H.264/AVC standard.

The complete validation of the architecture is one of the future works. Also, it is intended to prototype the architecture, synthesize it for Altera Stratix IV devices and, finally, integrate it with the upsampling architecture in order to create an Inter-layer Intra Prediction module.

7. References

- [1] C. A. Segall and G. J. Sullivan, "Spatial Scalability Within the H.264/AVC Scalable Video Coding Extension", IEEE Transactions on Circuits and Systems for Video Technology, 2007, pp. 1121-1135.
- [2] T. Wiegand, G. Sullivan and A. Luthra, "Draft ITU-T Recommendation and final draft international standard of joint video specification", (ITU-T Rec.H.264/ISO/IEC 14496-10 AVC), 2003.
- [3] T. Wiegand, G. Sullivan, J. Reichel, H. Schwarz, M. Wien, "Joint Draft 11 of SVC Amendment", Joint Video Team, Doc. JVT-X201, 2007.
- [4] G. Khurana, T. Kassim, T. Chua, M. Mi, "A pipelined hardware implementation of In-loop Deblocking Filter in H.264/AVC", IEEE Transactions on Consumer Electronics, 2006.
- [5] B. Shen, W. Gao, D. Wu, "An Implemented Architecture of Deblocking Filter for H.264/AVC", Proceedings - International Conference on Image Processing, ICIP, 2004.
- [6] L. Li, S. Goto, T. Ikenaga, T., "A highly parallel architecture for deblocking filter in H.264/AVC", IEICE Transactions on Information and Systems, 2005.
- [7] E. Ernst, "Architecture Design of a Scalable Adaptive Deblocking Filter for H.264/AVC". MSc thesis, Rochester, New York, 2007.

High Performance and Low Cost CAVLD Architecture for H.264/AVC Decoder

¹Thaísa Silva, ¹Fabio Pereira, ²Luciano Agostini, ¹Altamiro Susin, ¹Sergio Bampi
{tlsilva, Fabio.pereira, bampi}@inf.ufrgs.br, Altamiro.Susin@ufrgs.br
agostini@ufpel.edu.br

¹Grupo de Microeletrônica (GME) – Instituto de Informática – UFRGS
²Grupo de Arquiteturas e Circuitos Integrados (GACI) – DINFO – UFPel

Abstract

This paper proposes a low cost and high performance architecture for the Context Adaptive Variable Length Decoder (CAVLD) of the H.264/AVC video coding standard. Usually, a large number of memory bits and memory accesses are required to decode the CAVLD in H.264/AVC since a great number of syntax elements are decoded based on look-up tables. To solve this problem, we propose an efficient decoding of syntax elements through tree structures. Besides, due to the difficulty to parallelize operations with variable length codes, we performed some optimizations like merging coeff_token and T1 calculation, fast level bit count calculation, a synchronous on-the-fly output buffer, and a barrel shifter without input accumulator aiming to improve the performance. The architecture designed was described in VHDL and synthesized to TSMC 0.18 μ m standard-cells technology. The obtained results show that this solution reached great saves in hardware resources and memory accesses and it reached the necessary throughput to decode HDTV videos in real-time.

1. Introduction

H.264/AVC (or MPEG 4 part 10) [1] is the latest video coding standard developed by the Joint Video Team (JVT). JVT is formed by experts from ITU Video Coding Experts Group (VCEG) and ISO/IEC Moving Pictures Experts Group (MPEG). The first version of the H.264/AVC was approved in 2003 [2].

This standard aims at a wide range of applications such as storage, entertainment, multimedia short message, videophone, videoconference, HDTV broadcasting, and Internet streaming. Besides, this standard achieves significant improvements over the previous standards in terms of compression rates [1]. However, the computational complexity of the decoder and the encoder was increased. This complexity hinders, in the current technology at least, the use of H.264/AVC codecs designed in software when the video resolutions are high and when real time is desired. In these cases, a hardware design of H.264/AVC codec is necessary.

The H.264/AVC decoder is formed by the following modules: inter-frame predictor, intra-frame predictor, inverse transforms (T^{-1}), inverse quantization (Q^{-1}) and entropy decoder, as presented in Fig. 1.

The entropy decoder of the H.264/AVC standard (main, extended and high profiles) uses two main tools to allow a high data compression: Context Adaptive Variable Length Decoding (CAVLD) and Context Adaptive Arithmetic Decoding (CABAD) [3]. The focus of this work is the CAVLD, highlighted in Fig. 1.

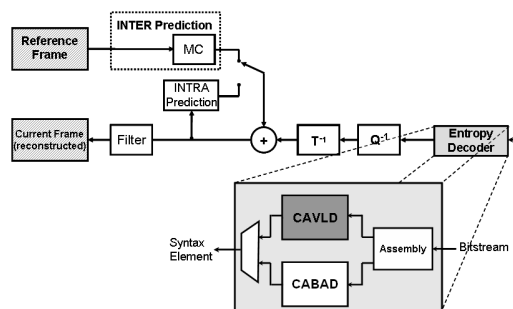


Fig. 1 – H.264/AVC Decoder Block Diagram with emphasis in the CAVLD

This paper is organized as follows. Second section presents a summary of the Context Adaptive Variable Length Decoder in the H.264/AVC standard. Third section presents the designed CAVLD decoder architecture. Section four presents the synthesis results. Section five presents comparisons of this work with related works. Finally, section six presents the conclusions and future works.

2. CAVLD Overview

Context-based adaptive variable length coding decoder (CAVLD) is used to decode residual blocks of transform coefficients. Several features are used to enhance data compression:

- Run-zero encoding is used to represent strings of zeros compactly;
- High frequency non-zero coefficients are often +/-1 and CAVLD handle those coefficients efficiently;
- The number of non-zero coefficients of neighboring blocks is used to choose the table employed during the decoding process;
- The absolute values of non-zero coefficients are used to adapt the algorithm to the next coefficient.

In main, extended and high profiles, the residual block data is decoded using CAVLD or CABAD tools, and other variable-length coded units are decoded using Exp-Golomb codes [2][3].

The coefficients are processed and decoded in zigzag order, from the highest to the lowest frequencies. Firstly the number of coefficients and high frequency coefficients of magnitude 1 (T1s) are transmitted, and then the sign of each T1 is transmitted, followed by the value of the other non-zero coefficients (Levels). Finally, the total of zeros before the last non-zero coefficient is decoded. Variable Length Codes (VLCs) are used in most of these steps, hindering their parallelization. Then, the decoder must be able to decode the block in few cycles at a satisfactory clock rate.

3. Designed CAVLD Architecture

The proposed architecture aims to decode CAVLC residual blocks with a low cost, maintaining high throughput. This architecture was designed through five main processing modules, one barrel shifter, one reassembly buffer, router elements and supplementary modules. Each one of the processing modules of the CAVLD architecture is responsible to decode one type of element, to know: *Coeff_Token*, T1s (Trailling Ones), Levels, Total_Zeros and Run_Before. Fig. 2 shows the highest hierarchy level, in a simplified view, of the designed CAVLD architecture.

The control of this architecture was implemented using a Finite State Machine (FSM) with six states. This FSM controls all CAVLD modules allowing a correct decoding process, including the *Coeff_Token* decoding (using trees), the *Trailing Ones* handling, *Levels* decoding, the *Total_Zeros* decoding (also using trees), *Run_Before* decoding (using trees) and the final assembly of the syntax element.

The datapath was designed through five main processing modules and the sub-sections below describe each one of the previous cited modules.

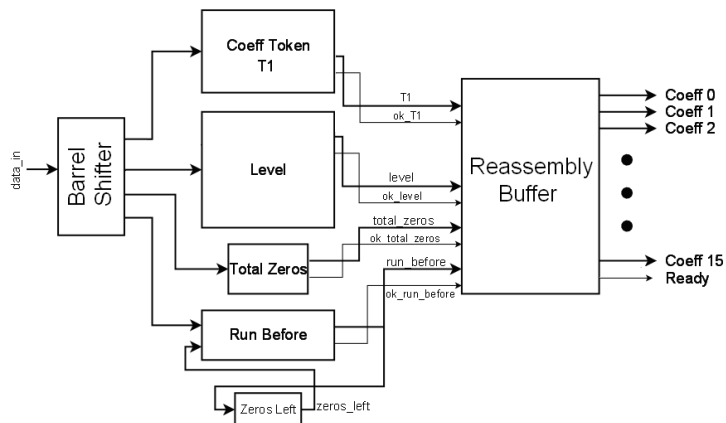


Fig. 2 – CAVLD Block Diagram

3.1. Barrel Shifter

In order to keep the real time requirement of our design, it was decided do not register the amount of used bits, before feeding the barrel shifter. So as soon as the size of a VLC is known, it is directly sent to the shifter and the input bits will be correctly aligned in the next clock cycle. The barrel shifter shifts all the output bits by any value in the range of 0 to 28 in a single clock cycle.

3.2. Coeff_token and T1s, Total_zeros and Run_before

Normally, the syntax elements such as *Coeff_token*, *Total_zeros* and *Run_before* are decoded based on look-up tables. The decoding based on look-up tables requires memory elements and multiple memory accesses, generating a high hardware cost and a high power consumption. Hence, in this work the decoding of the *Coeff_token*, *Total_zeros* and *Run_before* is accomplished through a tree structure, which is searched from the root to the leaves as presented in Fig. 3. When the search through the nodes reaches some of the leaves, this

means that a symbol was decoded. Based on the H.264/AVC CAVLD look-up tables, were created six trees to decode the *Coeff_token*, three to decode the *Total_zeros* and one to decode the *Run_before*. The choice of the tree used is made using the number of non-zero coefficients of decoded neighboring blocks.

The modules that implement the trees are completely combinational, getting the input from the barrel shifter and producing the number of levels, number of T1s, sign of T1s and number of bits in the VLC. Then the T1s are sent to the output buffer in parallel, and the next elements can be processed in the following cycle.

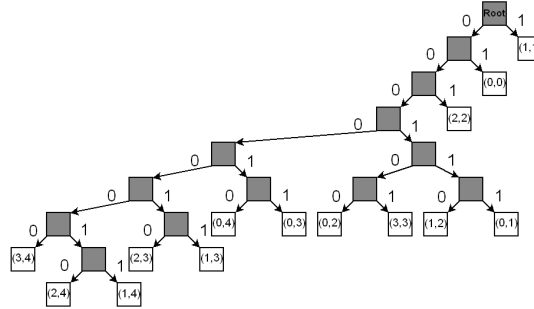


Fig. 3 – Example of tree structure used to decode the coeff_token

3.3. Levels

The remaining non-zero coefficients are decoded in this step. Each level is composed of a prefix and a suffix. The suffix size is adapted according to the magnitude of each processed coefficient. The Levels decoder is usually the bottleneck in CAVLD decoder. Therefore a pipeline approach with fast suffix size calculation was used to enable one level calculation per cycle, maintaining an acceptable critical path delay.

The level prefix is the number of zeros preceding the first one. The level suffix is an unsigned integer with a size adapted at each decoded level. Usually the size of the next suffix is only known at the end of the level calculation, but using the optimizations reported by [4], we can discover the next suffix length using the prefix-size and the current suffix length.

The level code then is calculated as showed in (1) and (2):

$$level = \frac{1}{2}(level_code + 2) \quad (1) \quad level = \frac{1}{2}(-level_code - 1) \quad (2)$$

(1) is used when level_code is even and (2) when level_code is odd. The level_code is given by (3):

$$level_code = level_prefix * 2^{suffix_length} + level_suffix \quad (3)$$

The final Levels block diagram is shown in Fig. 4.

3.4. Reassembly Buffer

Initially each decoded level is shifted into a buffer. When the run-zeros are decoded, the elements after the position where the zeros must be inserted, are again shifted by that number of zeros, as presented in Fig. 5. This procedure is repeated until the elements reach their final position in the buffer. As this is done while decoding, this step does not increase the total latency of the system.

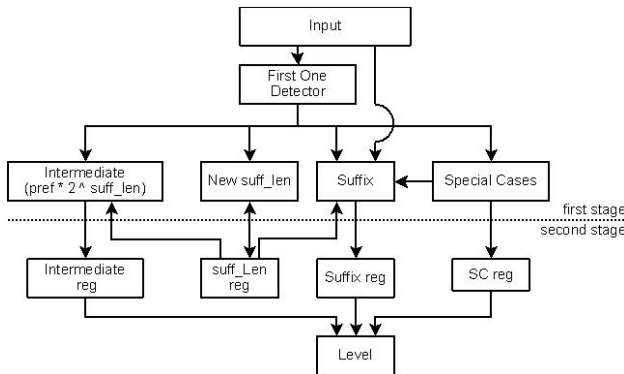


Fig. 4 – Levels Block Diagram

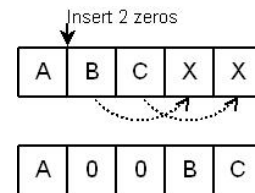


Fig. 5 – Example of data movement in the output buffer

4. Synthesis Results

The CAVLD architecture was described in VHDL and synthesized to TSMC 0.18 μ m standard-cell technology. The synthesis results show that the designed architecture used 8,476 gates and it allowed a maximum operation frequency of 155 MHz, reaching an average processing rate between 0.54×10^6 and 1.82×10^6 macroblocks/second in conformity with QP (quantization parameter) used in the validation process.

The number of cycles used to decode a 4x4 block depends on the block content (number of non-zero coefficients and the number of zero runs). The performance was measured using the Foreman sequence, coded with only I frames in QCIF resolution. Considering the 1920x1088 resolution at 30 fps, it is necessary to process 0.24×10^6 MB/s to reach real time. Thus, as the processing rate of this design is 0.54×10^6 MB/s in the worst case, our design reach easily the requirements to process HDTV videos in real time.

Moreover, from tab. 1 it is possible to notice that the architecture proposed in this work used about 35% fewer hardware resources than the designs [5] and [6] with a higher throughput. The design [7] used near to 52% more hardware resources than our architecture and used two blocks of RAM with 2,560 bits, reaching a throughput similar to that reached in our work. The design [4] used 19% less gates than our design, but it is not possible to compare the throughputs since the work [4] does not presented their average of used cycles per macroblock.

Tab.1 – Hardware cost and throughput comparison between ASIC Designs

	Our Design	Design [4]	Design [5]	Design [6]	Design [7]
Technology	0.18 μ m	0.18 μ m	0.18 μ m	0.18 μ m	0.13 μ m
Gates	8,476	6,883	13,189	13,192	17,586
RAM (in bits)	Not used	Not used	Not used	Not used	5,120
Throughput (MB/s)	1.82×10^6	Not shown	1.17×10^6	0.58×10^6	1.77×10^6

It is important to emphasize that the CAVLD architecture designed in this work does not use memory, since the decoding of the syntax elements that used look-up tables were replaced by efficient tree structures.

The designed architecture presented a high save in hardware resources and power dissipation. In terms of throughput, the architecture reached the requirements to process HDTV (1920x1088 pixels) frames in real time.

5. Conclusions and Future Works

This work presented the design of a CAVLD architecture targeting the H.264/AVC video compression standard. The architecture was designed without use memory, to save hardware resources consumption and power dissipation.

The designed solution used one cycle per VLC, it sends the consumed bits to the barrel shifter without registers, it implements a pipeline with a fast suffix length calculation on the Levels module and it uses a synchronous output buffer that reorganizes the outputs “on the fly”. Hence, the architecture achieved a high throughput rate which is enough to process HDTV frames (1920x1088 pixels) in real time (30 fps).

As future works it is planned a more consistent evaluation of the power consumption gains of this design in relation to designs that use look-up-tables. The first results in this direction indicate that the gains are really significant, but we need to finish a CAVLD design using memories to make this evaluation.

6. References

- [1] Joint Video Team of ITU-T, and ISO/IEC JTC 1: Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 or ISO/IEC 14496-10 AVC). JVT Document, JVT-G050r1, 2003.
- [2] INTERNATIONAL TELECOMMUNICATION UNION. ITU-T Recommendation H.264 (03/05): Advanced Video Coding for Generic Audiovisual Services, 2005.
- [3] I. Richardson, H.264 and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia. Chichester: John Wiley and Sons, 2003.
- [4] H. Lin, Y. Lu, B. Liu, and J. Ynag, “A Highly Efficient VLSI Architecture for H.264/AVC CAVLC Decoder,” *Proc. IEEE Trans. On Multimedia*, vol. 10, Jan. 2008, pp. 31-42.
- [5] T. Tsai and D. Fang, “An Efficient CAVLD Algorithm for H.264 Decoder,” *Proc. International Conference on Consumer Electronics (ICCE 2008)*, 2008, pp. 1-2.
- [6] G. Yu and T. Chang, “A Zero-Skipping Multisymbol CAVLC Decoder for MPEG-4 AVC/H.264,” *Proc. IEEE International Symposium on Circuits and Systems (ISCAS 2006)*, 2006, pp. 21- 24.
- [7] M. Alle, J. Biswas and S. K. Nandy, “High Performance VLSI Architecture Design for H.264 CAVLC Decoder,” *Proc. International Conference on Application-specific Systems (ASAP 2006)*, 2006, pp. 317–322.

A Multiplierless Forward Quantization Module Focusing the Intra Prediction Module of the H.264/AVC Standard

Robson Dornelles, Felipe Sampaio, Daniel Palomino, Luciano Agostini
 {rdornelles.ifm, fsampaio.ifm, danielp.ifm, agostini}@ufpel.edu.br

Universidade Federal de Pelotas – UFPel
 Grupo de Arquiteturas e Circuitos Integrados – GACI

Abstract

This work presents a multiplierless architecture of a forward quantization module focusing on the Intra Prediction of the H.264/AVC Standard. The architecture was described in VHDL and synthesized to Altera Stratix II FPGA and also to the TSMC 0.18 μ m CMOS technology. The designed architecture can achieve a maximum operation frequency of 126.8MHz, processing up to 137 QHDTV (3840x2048 pixels) frames per second. Another important characteristics of the presented architecture is the very low latency and the high throughput, features that can highly improve the Intra Prediction performance.

1. Introduction

The H.264/AVC is the latest video coding standard [1]. It was developed by experts of ISO/IEC and ITU-T and it intends to double the compression rates that the old standards were able to achieve. To encode a video, the standard can explore several types of redundancies: spatial, temporal and entropic. H.264/AVC encoder scheme is presented in fig.1. The standard implements a variety of profiles and this work focuses on the *Main* profile, that represents its colors in the YCbCr 4:2:0 color space [2].

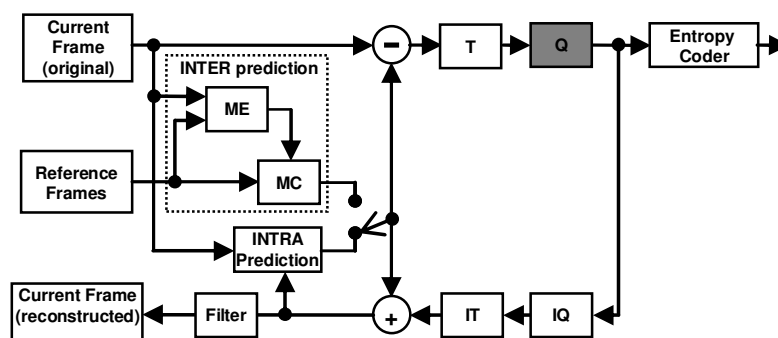


Fig. 1 – Block Diagram of an H.264/AVC Encoder.

The spatial redundancy of a video is explored, in the H.264/AVC standard, by Intra Prediction and Quantization modules. Intra Prediction explores the similarities between the blocks inside a frame, using the edges of neighbor blocks to represent the adjacent blocks. This way, a new block is created through a copy of those edges. There are a wide variety of copy modes: for the luminance layer, there are 9 modes for 4x4 blocks and 4 modes to 16x16 blocks; for the chrominance layer, there are only 4 modes for 8x8 blocks. After choosing a mode, the difference of the predicted block and the original one generates residual information that needs to be transformed and quantized to increase the compression rates. This process, although, is not lossless.

The edges of the neighbor blocks must be already decoded to intra predict the current block, since this is how they will be available on the decoder (the encoder and the decoder must use the same references). This way, the IQ and IT modules, that are part of the decoder, must be inserted also in the encoder. Thus, the residual information generated by a prediction must pass through the T, Q, IQ and IT (TQ/IQIT loop) modules, and then incorporated to the predicted block to finally be used as reference to the next prediction. While the residual information is being reconstructed in the TQ/IQIT loop, the Intra Prediction can not start its next prediction. So, the TQ/IQIT loop represents a bottleneck to the Intra Prediction module.

This work focuses in the Q module (highlighted in fig. 1), and proposes a fully parallel architecture, able to process up to 16 samples per clock cycle (an entire 4x4 block). This architecture may be integrated to a high performance TQ/IQIT loop dedicated to the Intra Prediction. The architecture was described in VHDL and synthesized to Altera Stratix II FPGA family and also to the TSMC 0.18 μ m standard-cells technology.

This work is presented as follows: Section 2 presents the Q module and its definitions; Section 3 presents the designed architecture; Section 4 presents the synthesis results and shows a comparison with related works; Section 5 concludes this work and suggests future works.

2. The Q (Forward Quantization) Module

In the H.264/AVC standard, the residual data generated by the differences between predicted blocks and the original ones are transformed to the frequencies domain and then this data suffers a quantization process to eliminate the frequencies that cannot be seen by the human eye. The quantization process also generates values that are most easily encoded by the entropy coder [2], generating high compression rates.

Since this process is applied to the frequency domain, all samples must be processed by the T module (formed by 4x4 FDCT, 4x4 Hadamard and 2x2 Hadamard transforms). After the transform, a 4x4 block has two kind of coefficients: the **DC coefficient**, on position (0,0); the **AC coefficients**, that are all the other coefficients.

The Quantization is done in different ways, depending on the coefficient nature and the prediction mode. All equations can be found in [2].

- When the prediction mode was the Intra_4x4, all coefficients, DC and AC, receive the same quantization process, presented in (1). The quantization presented in (1) is also applied to all AC coefficients when the prediction mode was the Intra_16x16 or Chroma_8x8. All coefficients that are processed only by the 4x4 FDCT in the T module will be processed this way:

$$\begin{aligned} |Z_{(i,j)}| &= (|W_{(i,j)}| \cdot MF + f) \gg qbits \\ \text{sign}(Z_{(i,j)}) &= \text{sign}(W_{(i,j)}) \end{aligned} \quad (1)$$

- When the prediction mode was the Intra_16x16 or Chroma_8x8, the quantization for the DC coefficients is made by equation (2). All coefficients that are processed by the Hadamard Transforms (FHAD 4x4 or FHAD 2x2) in the T module will be processed this way:

$$\begin{aligned} |Z_{D(i,j)}| &= (|Y_{D(i,j)}| \cdot MF_{(0,0)} + 2f) \gg (qbits + 1) \\ \text{sign}(Z_{D(i,j)}) &= \text{sign}(Y_{D(i,j)}) \end{aligned} \quad (2)$$

The values involved in the quantization processes are defined following tab. 1 and in equations (3) and (4). In tab.1, QP is the *Quantization Parameter*, a global parameter of the encoder. The QP can assume a value between 0 and 51. Higher QP values will decrease the video quality and increase the compression rates [2]. The operator % in tab.1 is the *modulus* operator, which means that **a%b** returns the **remainder of a divided by b**. The presented values are simplified and the original description and the simplification steps can be found in [2].

Tab. 1 - MF values depending on QP

QP%6	Positions (0,0),(2,0),(2,2),(0,2)	Positions (1,1),(1,3),(3,1),(3,3)	Other Positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

$$f = 2^{qbits}/3, \text{ for Intra Prediction} \quad (3)$$

$$qbits = 15 + \lfloor QP/6 \rfloor \quad (4)$$

Example: An hypothetical situation where the input has value **Y=150**, and **position (1,1)** in the 4x4 block. The prediction mode was **Intra_4x4**. **QP=16**, so **QP%6=4**, which means that **MF=3355**. With equations (3) and (4), **qbits=17** and **f=43690**. So, applying those values in equation (1) will results in the quantized value **Z=4**.

3. Designed Architecture

As seen in the previous section, the equations (1) and (2), that implement the forward quantization, are composed by one multiplication (by *MF*), one sum (by *f* or *2f*) and one right shift (by *qbits* or *qbits+1*). So, the datapath of a generic forward quantizer must contain a multiplier, an adder and a barrel shifter.

The bit size of the input is of 11 bits for the equation (1), and 14 bits for equation (2). This increase in the dynamic range is necessary to avoid overflows in the transform modules [3]. The proposed architecture is able to perform both equations (1) and (2), so, it was designed to support the worst case – 14 bits.

Since the MF value is a constant that depends on the coefficient position, and since there are only 3 different sets of positions, as seen in tab. 1, it is possible to design three dedicated multipliers, using only controlled shifts and sums. Each one of those multipliers is able to perform a multiplication by the constants of

one column of tab. 1. So, three different datapaths were developed; the only difference between them is the multiplier, which means that a datapath is able to process samples of only one set of positions (see tab. 1).

In the barrel shifter, the minimum value of $qbits$ is 15, which is the minimum shift that will be performed. This shift was done in the input values, before the multiplications. This way, the adders size can be reduced, as well as the critical path. To avoid errors, a parallel verification is done with the discarded less significant 15 bits. If there was an overflow on this verification, it will be added to the output of the main multiplication. Fig. 2 shows the detailed datapath with the related improvements.

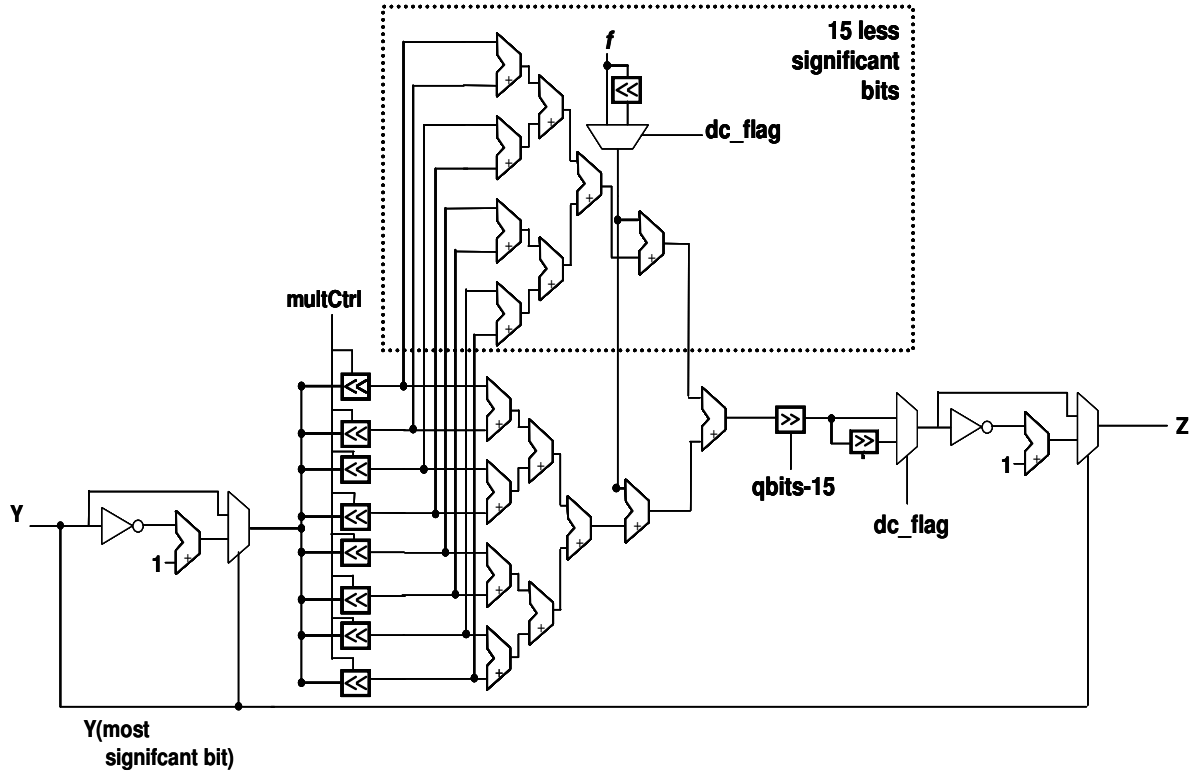


Fig. 2 – A single Quantization datapath with the shift-add multiplier.

All the control signals and constants values are pre-calculated and stored in memory, inside the control unit. This memory is indexed by the QP value. The control unit also generates the control of the multipliers. Thus, the complete architecture is composed by sixteen datapaths, chosen in a way that an entire 4×4 block of AC coefficients may be processed.

When the coefficients came from the 2×2 FHAD, they can be processed in just one clock cycle (using the datapaths (0,0), (2,0), (2,2) and (0,2)). When they came from the 4×4 FHAD, they will need 4 clock cycles to be processed, also by the datapaths (0,0), (2,0), (2,2) and (0,2). This procedure is defined by the control unit, and it can be activated by a flag called **HAD_4x4** or a flag called **HAD_2x2**. The activation of one of those flags will activate the **dc_flag**, which means that the equation (2) needs to be performed. Otherwise, the upcoming block has AC coefficients (and was processed only by the FDCT 4×4), and the equation (1) will be performed. In this case, all coefficients will be processed in one clock cycle. So, in the worst case, when the prediction was made in an Intra_16x16 mode, the architecture will take 16 cycles to process the 16 4×4 luma AC blocks, plus 4 cycles to process the Luma DC coefficients, plus 8 cycles to process the 4 4×4 chroma Cb and Cr AC blocks, and 2 more cycles to process the Chroma Cb and Cr DC coefficients. This way, 30 clock cycles are necessary to process an entire macroblock.

4. Synthesis Results

The presented architecture was described in VHDL and synthesized for the Altera Stratix II EP2S60F1020C3 FPGA using the software Altera Quartus II [4], and also to the TSMC $0.18 \mu m$ standard-cells technology [5]. Tab. 2 shows the FPGA synthesis results, and tab. 3 shows the standard-cells results.

Tab. 2 – Synthesis Results for Altera Stratix II EP2S60F1020C3 FPGA.

	Frequency (MHz)	#ALUTs	#ALMs	#DLRs	Throughput MSamples/sec	Frames QHDTV (3840x2048) / sec
<i>Stratix II FPGA</i>	118.51	1961	1099	480	1,896	128.5

Tab. 3 – Synthesis Results for 0.18 μ m TSMC Standard-Cells.

	Frequency (MHz)	#Gates	Throughput MSamples/sec	Frames QHDTV (3840x2048) / sec
0.18 μm TSMC	126.8	18219	2,028	137.5

As seen on the tables above, the proposed architecture easily achieves real time when processing very high resolution videos (like the QHDTV).

We found only two related works in the literature. Kordasiewicz [6] presents two solutions for the Q module: one is area optimized, and the other is speed optimized. The second solution presents a performance, with a throughput of 1,551 MSamples per second, against the 2,028 MSamples per second of our design. Our solutions also use less hardware resources, with 18k gates, against the 39k gates of Kordasiewicz [6]. The better results of our work occur in function of the optimized and efficient multiplierless solution that we designed. The solution proposed by Zhang [7] brings an improved multiplier dedicated to the quantization, but it does not present a complete Q module design. Zhang [7] presents six different multipliers for each set of positions in a 4x4 block, while in our work only one shift-add multipliers can perform the same operations that those six multipliers. This simplification drastically reduces the hardware cost of our Q module when compared to a hypothetical Q module composed by the multipliers proposed by Zhang [7].

5. Conclusions and Future Works

The Intra Prediction module of the H.264/AVC encoder has the TIT/QIQ loop as its bottleneck. This work presented an architecture of a Q module dedicated to the Intra Prediction constraints, using an efficient multiplierless scheme, that can process up to two billions of samples per second, processing till 137 QHDTV frames (3840x2048 pixels) per second. The high throughput achieved in this design enables the proposed architecture to process very high resolution videos in real time (30 frames per second). When compared to related works, the designed architecture presented the highest throughput and the lowest cost in hardware resources among all compared works.

The presented architecture is being integrated to a TQ/ITI loop dedicated to the intra prediction and a future work is to modify the Q architecture to also perform the inverse quantization process.

6. References

- [1] ITU-T. Recommendation H.264/AVC (11/2007). “Advanced Video Coding for Generic Audiovisual Services”. 2007.
- [2] I., Richardson, “H.264/AVC and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia”. John Wiley&Sons, Chichester, 2003.
- [3] R.S.S. Dornelles, F. Sampaio, D.M. Palomino, G. Corrêa, D. Noble and L.V. Agostini, “Arquitetura de um Módulo T Dedicado à Predição Intra do Padrão de Compressão de Vídeo H.264/AVC para Uso no Sistema Brasileiro de Televisão Digital”. Hifen (Uruguaiana), v. 82, p. 75-82, 2008.
- [4] Altera Corporation, “Altera: The Programmable Solutions Company”. Available at: www.altera.com.
- [5] Artisan Components, “TSMC 0.18 μ m 1.8-Volt SAGE-XTM Standard Cell Library Databook”. 2001.
- [6] R.C. Kordasiewicz, S. Shirani, “Asic and FPGA Implementations of H.264 DCT and Quantization Blocks”. In *IEEE International Conference on Image Processing*, IEEE, Piscataway, 2005, vol.3.
- [7] Y. Zhang, G. Jiang, W. Yi, M. Yu, F. Li, Z. Jiang and W. Liu, “An Improved Design of Quantization for H.264 Video Coding Standard”. *8th International Conference on Signal Processing*, IEEE, Piscataway, 2006, vol. 2.

H.264/AVC Video Decoder Prototyping in FPGA with Embedded Processor for the SBTVD Digital Television System

¹Márlon A. Lorencetti, ^{1,2}Fabio I. Pereira, ¹Altamiro A. Susin
{marlon.lorencetti, fabio.pereira, altamiro.susin}@ufrgs.br

¹Universidade Federal do Rio Grande do Sul
²Helsinki Metropolia University of Applied Sciences

Abstract

The SBTVD uses the H.264/AVC video coding standard for video transmission. This paper presents H.264/AVC in a software-hardware co-design architecture. The program runs on an embedded PowerPC processor present in the Virtex-II Pro FPGA of the Digilent XUPV2P prototyping board. The hardware modules are synthesized from a HDL description and mapped to the FPGA logic of the same chip. The communication between hardware modules and the PowerPC program is implemented by control signals and messages. The proposed prototype architecture was designed to interface the modules in such a way to minimize the need of a centralized global control. Also, the paper presents a hardware architecture designed to obtain a minimalist video decoder integrating the existing HDL modules. New hardware modules will be integrated as they become available till a complete HW decoder is obtained. Meanwhile the program will be helpful also as a debugging tool.

1. Introduction

Digital video coding is the solution for both transmission and storage of high definition videos. In a digital television system, the generated video must be compressed in order to allow its transmission, given the huge amount of data needed to represent a raw video. In 2003, the first version of H.264/AVC standard [1] was released, with features that allowed higher compression rates and better image quality compared to its predecessors, becoming the state-of-art in video coding systems.

The H.264 standard brought several improvements, in costs of a higher computational complexity. The image coding is performed in elementary pieces of image, called macroblocks, which are blocks of 16x16 pixels of luma, and its correspondents for chroma [2]. The decoding process is modular, where the operations are made in fairly independent modules. This modularity enhances a parallel hardware approach, instead of a software implementation, which would not be able to achieve enough performance to decode high definition videos at real time.

This work is part of the development and prototyping of a hardware H.264 decoder for the SBTVD (Sistema Brasileiro de Televisão Digital) [3], the digital television system adopted in Brazil. There is a consortium of research laboratories in universities all over Brazil, developing the whole coding and decoding IPs that are compliant with this system.

Now, this project is in its second stage, where the goal is to obtain a fully functional system, and also research possible new features either for digital television or video and audio coding. In the case of the video decoder, the final product will be a complete decoder in FPGA and the specification for an ASIC production.

This paper will present the current state of the video decoder development, showing the backgrounds, the software support in embedded processor and the prototyping strategy that is being used to reach a functional decoder.

2. Backgrounds

In order to offer some support to the hardware developers, a software has been created. This software, called PRH.264, is presented in [4]. It is based on a minimalist decoder implementation presented in [5], which is aimed to DSP processors, which was not the development main focus. Also available, were the source codes of the codecs used in computer media players, but those implementations used resources of hardware acceleration from the installed graphics card, which will not be available in our final prototype. PRH.264 was then created to decode H.264 videos and extract information from within the decoding process to serve as a test vector generator. So, a hardware developer is able to simulate the module being coded at the moment using real test vectors, extracted from known video sequences, and compare its results with the ones extracted from PRH.264. At present, the software decodes videos with some limitations, which include: no inter predicted frames, one slice per picture, no interlaced mode, no CABAC (Context Adaptive Binary Arithmetic Coding) entropy coding. The capability to decode inter-predicted frames and multiple slices per picture are now under development.

Besides, PRH.264 has been coded in a modular way. It means that all the main processes are encapsulated into functions, and all the functions needed to execute a part of the decoding process are written together in a source/header file pair. This software is entirely described in C language, for its advantages in code portability and coding easiness. Code portability is a main issue here, because the purpose of using PRH.264 to aid design and validation means it has to be as platform-independent as possible.

In the first stage of the project, some hardware coding effort was made. Hence, some modules of the video decoder are already described in HDL. These modules are: intra prediction, CAVLD (Context Adaptive Variable Length Decoder), deblocking filter, inverse transform and inverse quantization. These modules are again under a validation process, using the data extracted from PRH.264 and simulation test benches.

3. PRH.264 in Embedded Processor

As already discussed before, the final product in this branch of the project will be a hardware H.264 video decoder. The prototyping is being performed using FPGA, in a XUPV2P board from Digilent. This board contains a Xilinx Virtex-II Pro FPGA, which has two embedded PowerPC processors. Using EDK, part of Xilinx design toolkit, the C code of PRH.264 was ported to run in one of the PowerPCs.

3.1. Bitstream Processing

The image processing part of the software had a straightforward porting process, taking advantage of the C language code portability. Every module that accesses the bitstream, does so, using a NAL (Network Access Layer) unit buffer through functions that return a given number of bits, or an entire byte, or an Exp-Golomb word, etc. These functions take care of the maintenance of byte and bit position in the NAL unit buffer. Hence, if the data is contained in this buffer, these functions guarantee the data feeding to the processing modules, which do not need modification when the code runs on a different platform.

3.2. Input Data Management

Some changes were required in the inputs and outputs of the software. While running on a PC, PRH.264 reads the video bitstream from a file into a ring buffer, then the bits are analyzed to find NAL unit boundaries and an entire NAL unit is copied into the appropriate buffer. On the other hand, while running in the Digilent board, the software downloads the bitstream from a host computer before the decoding process begins. So, there is no need for the ring buffer, considering that all the required data is already loaded into memory. However, the NAL unit buffer was kept, in order to avoid changes in the units' detection process.

3.3. Video Exhibition

Another block that was modified is the one responsible for the video exhibition. On the PC, there are two versions of PRH.264. One of them runs on Windows using the Borland C++ Builder libraries, which were used to create a graphical interface. This graphical interface also helps the user choose the type of data to be extracted from the decoding process. Note that this version uses C++ techniques in the user interface, but everything else, either data input, processing, or data output, is made in ANSI C language. The other version runs on Linux, using the SDL (Simple DirectMedia Library) library to display the video, which provides the functions needed to create a window and update it with the decoded image. To display the decoded video, the application project created in the EDK tool instantiates the VGA IP required to use the video output of the XUPV2P board. This IP defines a region in the DDR memory that stores the exhibited pixels and controls the synchronization signals to manage the desired screen resolution. So, in the end of the decoding of a frame, a function of PRH.264 converts the YCbCr pixels to RGB space color and writes them in the correct memory position for exhibition.

3.4. Test Vectors Capturing

The extracted data from the decoding process is also displayed to the user in a different way. In both Windows and Linux versions, PRH.264 writes everything in files, separating the data in different files depending on the probed module and whether it is input or output data of that module. For the software running on the Digilent board, everything is sent as a text message via the serial connection, displayed in a host computer through a communication terminal.

For the Linux version, the user chooses the type of data extracted and other decoding constraints through a configuration file that is interpreted in the beginning of the program. On Windows, this choice is made through check boxes in the main user interface. In the Digilent board, the user interface is also made by a configuration file, like in the Linux version.

3.5. Embedded Software Validation

The PC based version was validated comparing the decoded video from PRH.264 with the decoded video from the JM H.264/AVC Reference Software Decoder [6]. Considering that the part of the software that actually processes the bitstream to generate the decoded video was not changed at all, the embedded version of PRH.264 is also considered validated.

4. Pre Prototype

As previously described, the set-top box is a complex device being developed by different institutions spread all-over Brazil. It is thus not hard to guess that problems during the integration stage are quite likely to arise. If the developers let that happen only in the final stage of the project, a great risk is taken.

Aiming to minimize problems during the integration phase of the set-top box, three milestones were elaborated for the H.264 decoder: pre-prototype, baseline prototype and final prototype. The idea behind the pre-prototype is as soon as possible, have a minimalist decoder, able to decode just a small set of H.264 streams but with the interfaces well defined and ready to be integrated in a set-top box. This pre-prototype is an intermediate step, creating a solid base where new features will be integrated in the future. Then the group may proceed the development to achieve a decoder capable of decoding baseline videos, and finally a full-featured high-profile H.264 decoder. The Windows version of PRH.264 is in use for validation purposes. The embedded version is likely to be used to perform the tasks of some absent hardware modules.

4.1. Pre-Prototype Architecture

The pre-prototype then is designed to be composed by the following modules: NAL unit decoder, parser, inverse quantization and transform, intra prediction module, adder, descrambler, and FIFO blocks. The blocks diagram is shown in fig. 1. The NAL unit decoder receives coded video through a SPI (Serial Peripheral Interface) interface, detects NAL boundaries, removes stuffed bytes, and feeds an input buffer with coded video. The parser then reads the coded video from the FIFO, decodes configuration data, intra prediction information and coded residue (CAVLD only).

Intra prediction data is fed to an intra_pred FIFO connected to the intra module, and coded residues are written into a cdd_res FIFO to be delivered to the inverse quantization and transform module. The intra prediction module then, uses the data in the intra_pred memory block and the neighbors of the current macroblock to generate intra predicted samples. The information in the cdd_res FIFO first goes through an inverse quantization process to then be de-transformed generating uncoded residue. The residues are then added to the predicted samples, and written to an output buffer. As the now decoded video information is created in a macroblock order, and the video interface needs a line by line output, the descrambler unit will control read and write address of the output buffer, reordering the output video samples.

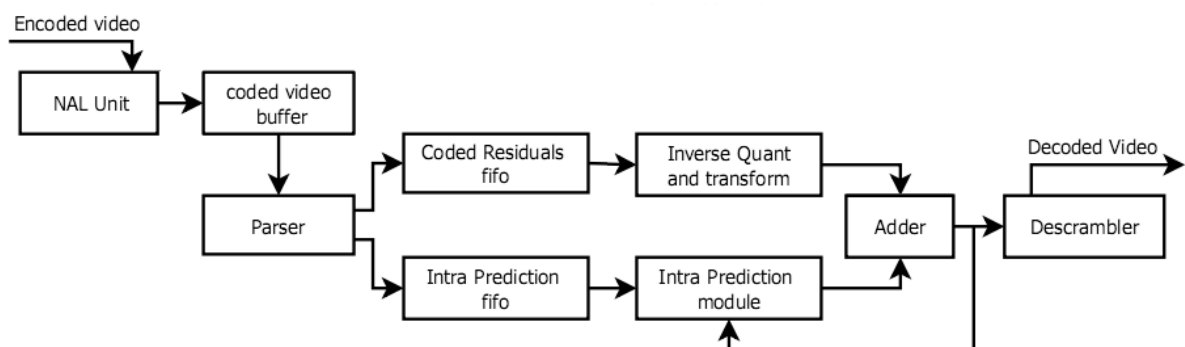


Fig. 1 – Pre-prototype View

Inter frame prediction, CABAC entropy decoding and deblocking filter will not be supported in the pre-prototype. These modules will not be integrated, and the parser is currently unable to decode information needed by them. On the other hand, real H.264 frames that do not use those algorithms will be fully processed in hardware by the pre-prototype decoder. During the integration of absent modules, some unsupported features will be performed in software, requiring the capability to run the PRH.264 software in the embedded processor. Some of these features will be a complete parser and some global control. When the inter prediction is integrated, the software is also likely to manage the reference pictures list.

The control is fully based on FIFO status, the modules will be enabled if they have valid data on their input and have space to write to the output, thus eliminating the need of a centralized global control. Immutable control data, like the screen resolution are saved to global registers and can be used by different modules.

Frequently mutable control data, like the quantization parameter Q_p , or prediction block size, are written to the FIFOs along with the input data needed by the algorithm. This is important because we do not have to know the state of each module, and when to update the control inputs, as the timing will be set by the module itself.

4.2. Validation

The validation of each new module (or sub-module) was made using the reference program PRH.264. The inputs and outputs were extracted from the software, and then a test bench would create the input stimulus using the generated input file. The test bench, created with Xilinx Modelsim, reads and compares the output from the module and the expected output from the software, signaling if a result mismatch was found. The validation scheme is shown in fig. 2. The bug fixing was made and all the hardware blocks are now validated. The maximum clock of each module after synthesizing assures the capability to decode high definition videos at the required 1920x1080 pixels resolution.

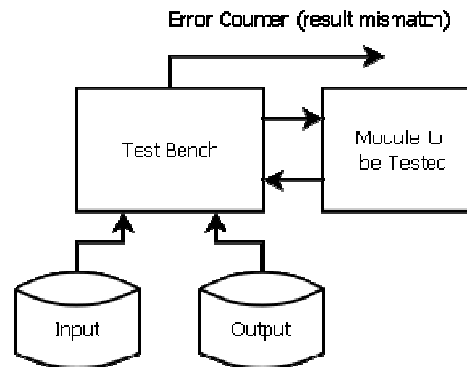


Fig. 2 – Validation Scheme

5. Conclusions and Future Works

This paper presented some of the efforts in the development of an FPGA based H.264 decoder for the SBTVD. In a first moment, some details that improved code portability of the software were explained. The use of this software as a test vector generator is a useful feature, since it is being used now to validate the previous coded modules in HDL. The next step is the integration of these HDL modules in FPGA, in order to obtain the pre-prototype architecture, using the embedded PRH.264 software to execute some tasks that might not be supported in hardware in this first prototype. Some possibilities are the NAL unit decoder, the parser module and a minor global control. This pre-prototype will require the definition of the interfaces between neighbor hardware modules, hardware and processor, and also the video decoder and the set-top box main part. This interface definition will help the developers to have a global timing and data flow vision of the decoder.

Also, the software will support more features of H.264, achieving a better coverage of the decoder profiles. In the embedded version, the serial communication will be replaced by an Ethernet connection to the host computer, using one PowerPC processor to control the data flow, and the other one to decode the video. The Ethernet connection will allow a streaming decoding process, permitting the use of bigger video sequences in a faster way.

6. References

- [1] Video Coding Experts Group, "ITU-T Recommendation H.264 (03/05): Advanced video coding for generic audiovisual services", International Telecommunication Union, 2005.
- [2] I.E.G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for the Next-generation Multimedia", John Wiley and Sons, England, 2003.
- [3] "Televisão Digital Terrestre – Codificação de vídeo, áudio e multiplexação", ABNT, Rio de Janeiro-RJ, 2007.
- [4] M.A. Lorencetti, W.T. Staehler and A.A. Susin, "Reference C Software H.264/AVC Decoder for Hardware Debug and Validation", XXIII South Symposium on Microelectronics, SBC, Bento Gonçalves-RS, pp 127-130, 2008.
- [5] M. Fiedler, "Implementation of a basis H.264/AVC Decoder", Seminar Paper, 2004.
- [6] "H.264 Reference Software", <http://iphom.hhi.de/suehring/ttml/>, 2009.

Design, Synthesis and Validation of an Upsampling Architecture for the H.264 Scalable Extension

¹Thaís Leal da Silva, ²Fabiane Rediess, ²Guilherme Corrêa, ²Luciano Agostini,

¹Altamiro Susin, ¹Sergio Bampi

{tlsilva, bampi}@inf.ufrgs.br, Altamiro.Susin@ufrgs.br

{frediess_ifm, gcorrea_ifm, agostini}@ufpel.edu.br

¹Grupo de Microeletrônica (GME) – Instituto de Informática – UFRGS

²Grupo de Arquiteturas e Circuitos Integrados (GACI) – DINFO – UFPel

Abstract

This paper presents the design, synthesis and validation of an architecture for the upsampling method of the scalable extension of the H.264 video coding standard. This hardware module is used between spatial layers (resolution) in scalability. The filter was designed to work in the context of an encoder / decoder with two dyadic spatial layers, which would be QVGA resolution for the base layer and VGA resolution for the enhancement layer. The architectures were described in VHDL and synthesized targeting Stratix II and Stratix IV Altera FPGAs devices and they were validated with the ModelSim tool. The results obtained through the synthesis of the complete upsampling architecture shows that this architecture is able to achieve processing rates between 384 and 409 frames per second when the synthesis is targeted to Stratix IV. The estimative for higher resolutions videos indicate processing rates between 96 and 103 frames per second for 4VGA resolution, and for the 16VGA resolution, the processing rate was estimated between 24 and 25 frames per second. With these results, the designed architecture reached enough processing rates to process input videos in real time.

1. Introduction

The popularization of different types of devices that are able to handle digital videos, ranging from cell phones till high-definition televisions (HDTV), creates a problem to code and to transmit a single video stream to be used for all of these types of devices. For example, in cell phones, which have a restricted size, weight and power consumption, it is difficult to justify the implementation of a decoder that is able to decode a HDTV video resolution and then subsample it to display this information on the cell display.

The H.264/AVC [1] standard is the newest and most efficient video coding standard, reaching compression rates twice higher than the previous standards (like MPEG-2, for example) [2]. But the increasing number of applications that are able to handle digital video with different capabilities and needs stimulated the development of an extension of the H.264/AVC standard. This extension is called H.264/SVC - Scalable Video Coding [3]. The concept of scalability is related to enhancement layers, that is, there is a base layer and adding enhancement layers it is possible to increase the video quality and/or the video spatial resolution and/or the video temporal resolution. The objective of H.264/SVC standardization was to allow the encoding of a high quality video bitstream with one or more subset bitstreams, which can be independently decoded with similar complexity and quality of the H.264/AVC [4].

This paper was organized as follow. In section 2 is presented the scalability and the upsampling method concept in the SVC standard. Section 3 shown the architecture that was designed in this work. Section 4 presents the synthesis results and finally, in section 5, the conclusions are presented.

2. Scalability and Upsampling in H.264/SVC Standard

The SVC Standard explores three types of scalability: temporal, spatial and quality. In order to allow the spatial scalability, the SVC follows a traditional approach based on multilayer coding, which was also used in the previous standards like H.262/MPEG-2, H.263 and MPEG-4 [4]. Each layer corresponds to a spatial, a temporal and/or a quality resolution supported.

The frames of each layer are coded with prediction parameters similar to that used in a single layer coding. In order to improve the coding efficiency comparing to simulcast (one complete bitstream is generated for each supported resolution), some inter layer prediction mechanisms were introduced. A coder can freely choose which information of the reference layer will be explored. In SVC, there is three inter layer prediction mechanisms:

Inter layer motion prediction: in order to use the motion information of a lower layer in the enhancement layers, it was created a new macroblock type. This new type is signaled by the base mode flag. When this

macroblock type is used, only a residual signal is transmitted and other information like motion parameters are not transmitted.

Inter layer residual prediction: this prediction can be applied to any macroblock that was coded with the inter prediction (the new macroblock type and the conventional types are supported). It is probable that subsequent spatial layers have similar motion information with a strong co-relation between the residual signals.

Inter layer intra prediction: when a macroblock of the enhancement layer is coded with base mode flag equal to 1 and the corresponding submacroblock of the reference layer was coded with intra frame prediction, the prediction signal of the enhancement layer is obtained by upsampling the corresponding intra signal of the reference layer [4]. This process of upsampling is the focus of this work and it is responsible to adapt the coding information in the lower resolution layer to the higher layer resolution.

The upsampling is applied when the prediction mode of a block is inter layer and the corresponding block in the reference layer has been encoded using intra prediction. The upsampling is also used in the inter layer residual prediction [4].

A multiphase 4-tap filter is applied to the luma components to implement the upsampling. A bilinear filter is applied to the chroma elements. The filters are applied first horizontally and after vertically. The use of different filters for luma and chroma is motivated by complexity issues. In SVC, the upsampling is composed by a set of 16 filters, where the filter to be applied is selected according to the upsampling scale factor and the sample position. For the luma elements, the filters are represented by equations (1) and (2). And the filters for chroma, by equations (3) and (4):

$$\begin{aligned} S4 &= (-3.A + 28.B + 8.C - D) \gg 5 & (1) & \quad S12 = (-A + 8.B + 28.C - 3.D) \gg 5 & (2) \\ S4 &= (24.A + 8.B) \gg 5 & (3) & \quad S12 = (8.A + 24.B) \gg 5 & (4) \end{aligned}$$

3. Designed Architecture

The architecture designed in this work uses the YCbCr color space and the 4:2:0 color sub-sampling model. The QVGA (320x240 pixels) and VGA (640x489 pixels) resolutions were adopted for the base and enhancement layers, respectively. Thus, the implementation was made considering just the dyadic case that occurs when the spatial resolution is doubled horizontally and vertically between subsequent layers.

Fig. 1 shows the architecture for the complete upsampling module proposed in this work. The fig. shows the two luminance filters (horizontal and vertical – Luma H Filter and Luma V Filter, respectively) and the two chrominance filters (horizontal and vertical – Chroma H Filter and Chroma V Filter, respectively). Also, the figure shows the memories (MEM IN) which work as input buffers for the luminance and chrominance filters and the memories (MEM H1 and MEM H2) used as ping-pong transpose buffers between the horizontal and vertical filters of luminance and chrominance. The clipping operators (Clip, in Fig. 1) are also important, since they are used to adjust the outputs to the same dynamic range of the inputs. Finally, the control is also represented by the dotted lines in Fig. 1. The memory address registers and their respective multiplexers were omitted in the figure for simplification, as well as their control signals.

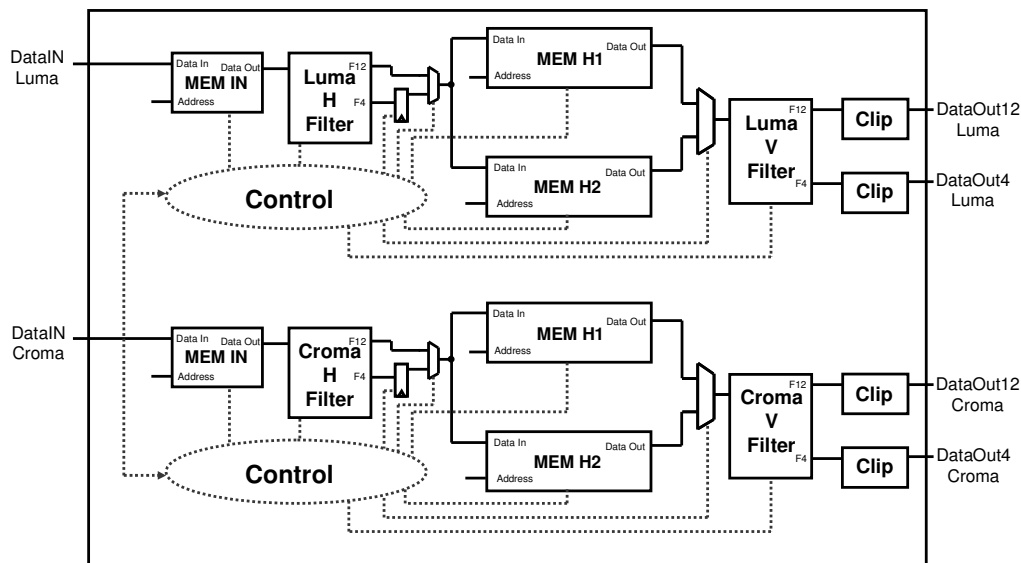


Fig. 1 – Complete upsampling architecture

.During the design, the possibility of using the same input data by both filters was analyzed and confirmed, leading to the architecture presented in Fig. 2. This architecture presents four registers which are serially connected in such a way that the input data is stored by a D register in the first clock cycle and transferred to a C

register in the second cycle, in order to allow the storage of a new input data in the D register. This happens in the same way for the other registers (i.e., the data is transferred from D to C, from C to B and from B to A). When the four registers are filled with valid data, the filters start to perform the calculations.

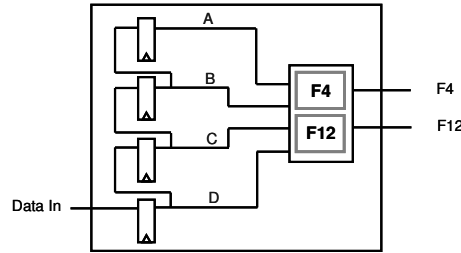


Fig. 2 – Filter core of upsampling

Each filter for the luminance upsampling (F4 and F12, in Fig. 2) was designed through algebraic manipulations over the equations showed in (1) and (2) to transform all multiplications in single shifts. The final equations, after the manipulations, are presented in (5) and (6). The equations (7) and (8) were obtained through similar manipulations over the equations (3) and (4), which correspond to the chrominance information.

$$S4 = (-2.A - A + 16.B + 8.B + 4.B + 8.C - D) \gg 5 \quad (5)$$

$$S12 = (-A + 8.B + 16.C + 8.C + 4.C - 2.D - D) \gg 5 \quad (6)$$

$$S4 = (16.A + 8.A + 8.B) \gg 5 \quad (7)$$

$$S12 = (8.A + 16.B + 8.B) \gg 5 \quad (8)$$

The architectures of both filters (luminance and chrominance) were described in VHDL and completely validated with the ModelSim tool. However, the architecture of the complete upsampling module was designed and implemented, but its validation was not concluded yet. The input stimuli used in the testbench file were extracted from the H.264/SVC reference software. The testbench file created for the luminance validation writes the input frames in a memory “MEM IN” and reads the output data, which is obtained after the clipping operations. This testbench was designed with some steps which work in parallel in order to allow the simulation in the same way that the architecture was designed in hardware, implementing all the designed parallelisms. The testbench is composed by one process which writes the input frames in the input memory, one process that starts the horizontal filter processing, one process that starts the vertical filter processing and one process that extracts the output data from the architecture.

Comparing the data generated by the architecture (extracted by the testbench) with the data extracted from the reference software, it was possible to verify that the architecture worked as expected.

The chrominance architecture was designed following the specifications of Segall [5], who stated that the chrominance data is filtered by a set of 16 filters, as the luminance data. Besides, [5] states that the filters of indexes 4 and 12 must be used alternately for the dyadic case. However, when the data was extracted from the reference software and compared during the validation, some differences between the expected and generated values were observed. An investigation in the reference software has shown that, for chrominance, the filters of indexes 2 and 10 are being used, instead of the filters of indexes 4 and 12. As the reference software is in constant development and, in many aspects, allows the use of non-standardized tools, and as the statements of [5] are in disagreement with the reference software results, we have decided, in this work, to consider the information presented in [5] to implement the chrominance filter architecture. Besides, the statements of [5] concerning the luminance filtering are completely in agreement with the reference software, reinforcing the reliability of the presented information.

By these reasons, it was not possible to validate the complete architecture with the data extracted from the reference software. Thus, some data were generated by a software implemented in C for validation. The developed software performs the same processing executed by the reference software, but uses different indexes values.

4. Synthesis Results

Initially, the cores of the architectures were synthesized to Altera Stratix II FPGAs. However, as was not possible to synthesize the complete architecture of the upsampling (luminance + chrominance) for neither Stratix II device, latter the architectures were synthesized targeting devices of the Stratix IV family.

In the two first columns of tab.1 are presented the synthesis results of the cores architectures developed for luminance and chrominance. The results of the luminance and chrominance complete architectures synthesized independently are showed in the fourth and fifth columns and, finally in the last column of the table, the results of the upsampling complete architecture are presented. In the Quartus II tool is not possible to generate totally

reliable timing analyses results for Stratix IV FPGA devices, because the timing models are still in development. The first frequency result presented in the tab.1 comes from the model called “Slow 900mV 85C Model” and the second result from the model called “Slow 900mV 0C Model”.

Tab. 1 – Synthesis results for the proposed architecture for upsampling block.

	Luma Core	Chroma Core	Luma	Chroma	Complete Upsampling
Model 1 Frequency (MHz)	151.42	381.68	137.61	190.99	119.5
Model 2 Frequency (MHz)	161.39	406.01	146.76	202.59	127.42
ALUTs	154	42	1,267	759	2,024
Dedicated Logic Registers	66	40	577	454	1,032
Memory Bits	-	-	5,222,400	1,288,800	6,451,200

Selected Device: Stratix IV EP45SGX530HH35C3

From these results was possible to estimate the processing rates in frames per second. The processing rates for the complete architectures of luminance and chrominance and for the complete upsampling architecture designed in this work are presented in the line “VGA Frames” of tab.2. Tab. 2 also presents the estimates for frame rates for higher resolutions. This is the first design related in the literature for the upsampling filter of the H.264/SVC standard, hence not was possible to accomplish comparisons with other works.

Tab. 2 – Estimated processing rates

	Slow 900mV 85C Model			Slow 900mV 0C Model		
	Luma	Chroma	Complete	Luma	Chroma	Complete
VGA Frames	442	1,233	384	471	1,308	409
4VGA Frames	*111	*309	*96	*118	*328	*103
16VGA Frames	*27	*77	*24	*29	*82	*25

*estimative

5. Conclusions

This work presented the design of an architecture for the upsampling module used in the scalable video coding according to H.264/SVC standard. This module was designed in the context of a codec with two spatial dyadic layers with resolutions QVGA (for the base layer) and VGA (for the enhancement layer).

The architectures were described in VHDL and synthesized to the Altera Stratix II and Stratix IV FPGA devices. The validation was made through of the Modelsim. The synthesis results presented an operation frequency of 119.5 MHz in the worst case for the complete architecture when the syntheses were directed to Stratix IV devices. With this frequencies was possible to arriving processing rates between 384 and 409 frames per second.

An estimate of processing rates for higher resolutions was also presented. When the 4VGA resolution was used in the enhancement layer, the processing rate was of 96 frames per second in the worst case and when the 16VGA resolution was used, the processing rate was of 24 frames per second in the worst case.

The validation of the complete upsampling architecture designed in this work is a future work. The prototyping of this architecture is also another future work. Other future investigations are the upsampling implementation with more than two spatial layers and the upsampling implementation for the non-dyadic cases.

6. References

- [1] JVT Editors (T. Wiegand, G. Sullivan, A. Luthra), Draft ITU-T Recommendation and final draft international standard of joint video specification (ITU-T Rec.H.264|ISO/IEC 14496-10 AVC), 2003.
- [2] I. Richardson, “H.264/AVC and MPEG-4 Video Compression – Video Coding for Next-Generation,” Multimedia. Chichester: John Wiley and Sons, 2003.
- [3] T. Wiegand, G. Sullivan, H. Schwarz, and M. Wien, “Joint Draft 11 of SVC Amendment,” Joint Video Team, Doc. JVT-X201, Jul. 2007.
- [4] H. Schwarz, D. Marpe, T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard,” *Proc. IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, No. 9, pp. 1103-1120, Sept. 2007, invited paper.
- [5] C. Segall and, G. Sullivan, “Spatial Scalability Within the H.264/AVC Scalable Video Coding,” *Proc. Extension. IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, No. 9, pp. 1121-1135. Set. 2007.

SystemC Modeling of an H.264/AVC Intra Frame Video Encoder

¹Bruno Zatt, ¹Cláudio Machado Diniz, ²Luciano Volcan Agostini, ¹Altamiro Susin,
¹Sergio Bampi

{bzatt,cmdiniz,agostini,bampi}@inf.ufrgs.br, altamiro.susin@ufrgs.br

¹GME – PGMICRO – PPGC – Instituto de Informática - UFRGS
Porto Alegre, RS, Brasil

²Grupo de Arquiteturas e Circuitos Integrados – GACI – DINFO - UFPel
Pelotas, RS, Brasil

Abstract

This work presents a SystemC modeling of an H.264/AVC intra frame encoder architecture. The model was used to evaluate the architecture in terms of hardware amount of memory, interfaces bandwidth and the characterization of the encoder timing. This evaluation provides the information to choose between architectural alternatives and to correct possible errors before the hardware description reducing the developing time and the cost of errors correction. The modeled architecture is able to encode H.264/AVC HDTV 1080p video sequences in real time at 30 fps.

1. Introduction

H.264/AVC [1] is the latest video coding standard developed by the Joint Video Team (JVT), assembled by ITU Video Coding Experts Group (VCEG) and ISO/IEC Moving Pictures Experts Group (MPEG). This standard achieves significant improvements over the previous standards in terms of compression rates [2].

Intra frame prediction is a feature introduced in H.264/AVC which is not present in previous standards. It explores the spatial redundancy in a frame, by producing a predicted block based on spatial neighbor samples already processed in this frame. This prediction mainly acts where the motion estimation, which explores the temporal redundancy, does not produce a good match for P and B macroblocks and in intra frames (I frames), which contain only intra macroblocks and the motion estimation cannot be applied. Intra frames typically occur at a lower rate than P and B frames, however, the efficient prediction on these frames are determinant to coding efficiency and video quality.

The complexity of intra frame encoder is based on the following steps: i) to generate all candidate prediction blocks; ii) to decide which prediction block produces the best match compared to original block; iii) to encode the chosen block and iv) to reconstruct neighbor samples for the next prediction. In a real-time intra frame encoder hardware design targeting high definition videos, the design space exploration is huge. A high-level hardware-based simulation model is appropriate to explore such design space, which real-time is the main design restriction.

In this work an H.264/AVC intra frame encoder model is proposed, which is described in SystemC language. The model contains an intra frame prediction module and the reconstruction path, which is composed by transform/quantization and inverse transform/inverse quantization (scaling).

The paper is organized as follows. H.264/AVC intra frame prediction concepts are presented in section II. The SystemC model is presented in details in section III. Section IV shows the results, section V shows the conclusions and section VI presents the paper references.

2. Intra Frame Prediction

In H.264/AVC a 4:2:0 color ratio is used when considering baseline, main and extended profiles. In this case, for each four luma samples are associated one blue chrominance (Cb) and one red chrominance (Cr) samples. This standard works over image partitions, called macroblocks (MB), composed by 16x16 luma samples, 8x8 Cb and 8x8 Cr samples. Considering the aforementioned profiles, an intra MB can be predicted using one of two intra frame prediction types: Intra 16x16 and Intra 4x4. In this work these types will be referred to as I16MB and I4MB respectively. The I16MB is applied to blocks with 16x16 luma samples and the I4MB is applied to blocks of 4x4 luma samples. The chroma prediction is always applied to blocks of 8x8 chroma samples, and the same prediction type is applied for both Cb and Cr. Each of these two types of luma prediction contains some prediction modes to generate a candidate block that matches some specific pattern contained in the frame.

Spatial neighbor samples already processed in the frame are used as reference to prediction. When spatial neighbors used to intra predict a specific block are not available (if they are out of frame limits, for example) this mode is not applied for this block. The prediction stage can produce as high as 9 modes for each 4x4 MB

partition and 4 modes for the MB. The mode decision stage decides which MB partitioning will be performed and which modes cause the best match compared to the original MB.

After the mode decision, the residual information (the difference of the prediction block and the original block) is transformed, quantized and sent to the entropy encoder. The quantized block is also inverse quantized (scaling) and inverse transformed to generate the reconstructed block. Some samples of the reconstructed block are used as neighbor samples for the next blocks to be predicted. This process forms a data dependency between neighbors 4x4 blocks for the intra prediction. This data dependency difficult an MB-level pipeline scheme.

3. H.264/AVC Intra Frame Encoder SystemC Model

The intra frame encoder architecture presented in [3] was modeled using a Transaction Level Modeling (TLM) methodology [4]. The TLM allows separating the communication details from the computation. In this implementation the abstraction used was Cycle-Accurate (CA) computation model. In this model, the computation is pin-accurate and executes cycle-accurate while the communication between the modules is done by approximate timed abstract channels.

This model was designed to characterize the intra frame encoder hardware to reach performance to real time encoding for H.264/AVC at 30fps for HDTV 1080p video sequences. The model process four samples (one line of 4x4 samples block) in parallel.

As presented in Fig. 4 the model is composed by five main SystemC modules: MB input buffer, intra encoder, mode decision, T/Q and IT/IQ modules. Inside each module the computation and communication is cycle-accurate but the communication between modules is implemented by abstract channels, in this case FIFOs were used. This approach was adopted to permit experiments with different buses topologies.

In Fig. 1 the boxes represent SC_MODULES and ellipses represent SC_METHODs or SC_THREADS. Hierarchical channels (sc_channel) are represented using cylinders and are linked to modules through little boxes which represent interfaces (sc_interface).

The objective of this model is to analyze the proposed H.264/AVC intra encoder architecture considering the communication between the modules and with the external memory, the number of operators, the amount of internal memory and the timing.

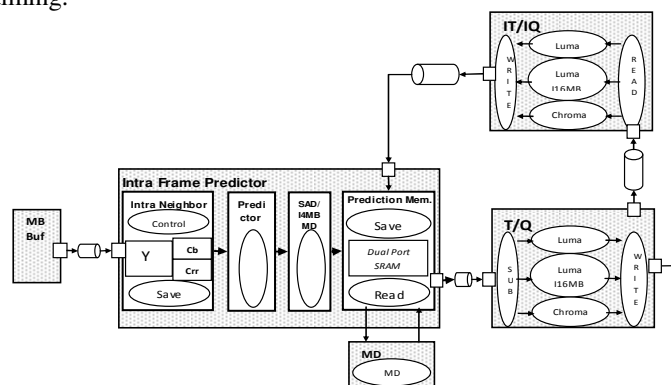


Fig. 1 – Intra frame encoder SystemC model schematic.

3.1. Intra Frame Encoder

This is the module responsible for all the intra frame prediction process, from the neighbor samples fetching to the decision between the I4MB prediction modes. It is represented in Fig. 1 by the largest gray box.

The intra encoder was modeled as a hierarchical pipeline and it is composed by four modules: Intra Neighbor, Sample Predictor, SAD/I4MB Mode Decision and Prediction Memory. Each SC_MODULE is composed by SC_METHODs and/or SC_THREADS which implement the computation logic.

Intra Neighbor: This module is composed by three SRAM memory blocks and two SC_METHODs. Each memory block stores the neighbor samples of the upper line of one color channel. These memories were sized to fit with one HDTV 1080p samples line, so “Y Memory”, which stores the luminance channel, is composed by 1920 samples while “Cb Memory” and “Cr Memory” keep 960 samples each.

Each SC_METHOD implement one FSM (Finite State Machine). The ‘control’ (Fig. 4) method implements a FSM with 44 states and it is responsible for reading the neighbor samples from the memories, calculating the DC parameters and coefficients for the plane prediction mode, sending the correct neighbor samples to the Sample Predictor module and controlling the intra prediction process. This FSM dispatches the processing intercalating the different kinds of blocks in the following order: I4MB / I16MB + Cb / I4MB / I16MB + Cr.

‘Neighbor Save’ method waits the reconstructed samples from the T/Q and IT/IQ loop to update the neighbor memories. This method describes an FSM with 13 states. The methods communicate each other through events (sc_event) and with the memory blocks through signals (sc_signal).

Sample Predictor: The Sample Predictor is composed by 31 PEs (processing elements) controlled by one FSM with 13 states, described by an SC_METHOD. The PEs use the neighbor samples received from the Intra Neighbor module to generate the prediction for all the nine I4MB modes or the four I16MB and the four chroma modes in parallel.

There are four different kinds of PEs. Twenty three PEs are used to apply the interpolation operations required for the directional samples predictions. Other eight PEs are used to apply the plane mode operation, four of them to I16MB prediction and four to chroma prediction. The PEs outputs generate all possible sample predictions and are multiplexed to generate each mode prediction. Four predicted samples are generated at each clock cycle. The FSM controls the multiplexers.

SAD / I4MB Mode Decision: This module is composed by nine trees of adders. Each adder tree accumulates the SAD of one different prediction mode, intercalating between the different block kinds in the same manner described earlier in Intra Neighbor module section. The control is performed by a five-state FSM implemented using an SC_METHOD. A tree of comparators selects the prediction mode, which generates the smaller SAD value for the I4MB.

Prediction Memory: After generating the prediction and calculating the SAD, the mode decision chooses the best MB type to be encoded. Once choose is done, the residues are used to be entropy encoded and then the predicted samples are used to reconstruct the coded frame. To provide this data to the next encoding steps is necessary to generate the prediction again, or to store it in a memory block, as implemented in this model.

The prediction memory module has an internal dual port SRAM with 68 lines and 36 samples wide. This memory is organized as follows: the first four lines store all the nine prediction modes of the I4MB, the even lines from 4 to 35 store I6MB and Cb prediction, the odd lines store I6MB and Cr prediction and the other lines store just I6MB prediction.

Two FSM are responsible for memory write and read. The first FSM controls the memory write and is formed by 11 states and the second FSM have 6 states. Both are implemented using SC_METHODs and communicate each other through sc_signals.

3.2. Transform and Quantization

Once the video reconstructed in the decoder must be exactly equal to the one reconstructed in the encoder, the prediction is done based in the reconstructed neighbor samples instead of using the original image as reference. To reconstruct the neighbor samples, the residues resultant from the difference between original and predicted frames must be processed by forward and inverse transforms and quantization.

To complete the intra frame encoder process it was necessary to implement the forward and inverse transforms and quantization. To best fit with the intra encoder datapath, these modules were implemented to process four samples (one 4x4 block line) per clock cycle. The forward transform and quantization were unified and implemented using a single SC_MODULE. The same approach was used to implement the inverse ones.

These modules were designed using five SC_METHODs. In the T/Q module, one SC_METHOD is used to subtract the predicted and original samples generating the residue. Three other methods process the different kinds of blocks (Luma, Luma I16MB and Chroma). One more SC_METHOD is responsible to write the processed data in the output FIFO.

3.3. Mode Decision

The mode decision algorithm is not in the scope of this work and is not defined yet. Some experiments are been realized to determine the best hardware oriented mode decision for this architecture. However the module interface was defined and a very simple decision algorithm was implemented using a SC_METHOD to permits the entire model verification for every encoding possibilities.

4. Results

The SystemC modeling of an H.264/AVC intra encoder was developed to easily obtain information about the proposed architecture considering the timing, amount of memory and data dependency.

The timing information extracted from the model is represented by Fig. 2. The 'y' axis represents the processing units and 'x' axis details the used clock cycles. To calculate an I4MB block prediction and determine the SAD 10 cycles are spent but the data dependency between the neighbor blocks forces the architecture to wait the T/Q and IT/IQ processing of the previous block, spending 25 cycles per block. It generates clock cycles of idle time for the predictor and SAD calculator, so the intercalation between I4MB and I16MB, detailed in previous section, takes advantage using the datapath during this idle time, providing a better utilization of hardware resources.

One macroblock is formed by sixteen 4x4 blocks, considering 25 cycles to predict and calculate SAD of each block, 400 cycles are used to this task for each MB. Adding this number to the initial 4 cycles for DC calculation and the necessary Cb and Cr transformation and quantization 471 cycles are used to process a MB if the I4MB mode is chosen. If the mode decision decides for the I16MB prediction, the entire Y channel of the MB must be transformed, spending more 149 cycles, totalizing 620 cycles to predict one MB in the worst case.

With this number of cycles it is possible to determine that the HW implementation must reach 150MHz to attend H.264/AVC real time encoding for HDTV 1080p at 30 fps. Note that this frequency does not depend on the target technology, it just determine the minimum frequency the HW implementation needs to reach for real time encoding. The mode decision between the different MB prediction types, I4MB and I16MB, is not added because it is calculated in parallel with the Cb and Cr transformation.

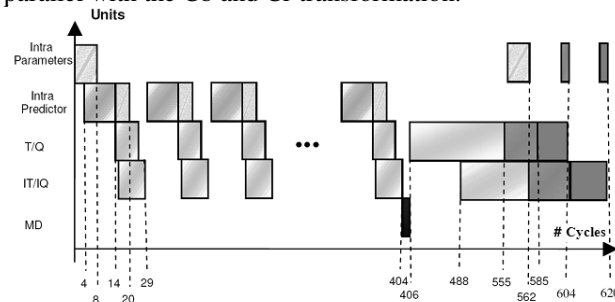


Fig. 2 – Timing Diagram.

The traffic in the communication channels between the modules was characterized. The input channel requires a bandwidth of 93,3 MB/s (MBytes per second) for HDTV 1080p encoding. The bandwidth of the other channels, which connects the T/Q, IT/IQ, intra encoder and the output to entropy encoding depends on the mode decision logic and the input video, so it cannot be determined. However, when encoding HDTV 1080p at 30fps, this bandwidth can vary between 93,3 MB/s if just I4MB is used and 155,5 MB/s when I16MB only prediction is used.

5. Conclusion

This work presented a SystemC model to analyze an H.264/AVC intra frame encoder architecture. The proposed architecture was presented and the relationship between the modules and its internal implementation were detailed.

The model is composed by five main modules: intra encoder, mode decision, T/Q and IT/IQ modules designed as a hierarchical pipeline. It was modeled to process four samples in parallel to reach the performance to real time encode H.264/AVC at 30fps for HDTV 1080p video sequences.

Results of hardware amount of memory, interfaces bandwidth, video encoding quality and the characterization of the encoder timing were quantized and presented. These results represent important information to HW development and functional verification.

No other similar work of modeling was found in the literature up to the current moment. Furthermore, the published implementations cannot reach the necessary performance to 1080p real time encoding. The work in [5] presents an intra frame codec in UMC 0.18 μ m technology. The encoder part is able to process 720p 4:2:0 (1280x720) at 30 fps, working at 117 MHz. The work in [6] presents some optimizations on the intra frame algorithm and a hardware architecture for intra frame encoding. It is also implemented in UMC 0.18 μ m and it is able to encode 720p 4:2:0 at 33 fps, operating at 120 MHz. The architecture modeled in this work, after implemented will encode 1080p at 30 fps running at 150 MHz.

As future work it is planned to model the entropy encoder, to evaluate the tradeoff between quality and amount of hardware, to use SATD as similarity metric and to model the inter frame encoding modules. The hardware implementation is also planned to future work.

6. References

- [1] JVT Editors. ITU-T H.264/AVC Recommendation (03/2005), ITU-T, 2005.
- [2] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard", IEEE Transactions on Circuits and Systems For Video Technology, IEEE, July 2003.
- [3] C. Diniz, B. Zatt, L. Agostini, S. Bampi and A. Susin, "A Real Time H.264/AVC Main Profile Intra Frame Prediction Hardware Architecture for High Definition Video Coding", 24th South Symposium on Microelectronics, May 2009,
- [4] L. Cai, D. Gajski, "Transaction Level Modeling: An Overview", 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, 2003.
- [5] C.-W. Ku; C.-C. Cheng, G.-S. Yu, M.-C. Tsai, T.-S. Chang, "A High-Definition H.264/AVC Intra-Frame Codec IP for Digital Video and Still Camera Applications", IEEE Transactions on circuits and systems for video technology, IEEE, August 2006, pp. 917-928.
- [6] C.-H. Tsai, Y.-W. Huang, L.-G. Chen, "Algorithm and Architecture Optimization for Full-mode Encoding of H.264/AVC Intra Prediction", IEEE Midwest Symposium on Circuits and Systems, IEEE, August 2005, pp. 47-50.

A High Performance H.264 Deblocking Filter Hardware Architecture

^{1,2}Vagner S. Rosa, ¹Altamiro Susin, ¹Sergio Bampi
vsrosa@inf.ufrgs.br, Altamiro.Susin@ufrgs.br, bampi@inf.ufrgs.br

¹Instituto de Informática - Universidade Federal do Rio Grande do Sul
²Centro de Ciências Computacionais - Universidade Federal do Rio Grande

Abstract

Although the H.264 Deblocking Filter process is a relatively small piece of code in a software implementation, profile results shows it cost about a third of the total CPU time in the decoder. This work presents a high performance architecture for implementing a H.264 Deblocking Filter IP that can be used either in the decoder or in the encoder as a hardware accelerator for a processor or embedded in a full-hardware codec. A developed IP using the proposed architecture support multiple high definition processing flows in real-time.

1. Introduction

The standard developed by the ISO/IEC MPEG-4 Advanced Video Coding (AVC) and ITU-T H.264 [1] experts set new levels of video quality for a given bit-rate. In fact, H.264 (from this point the standard will be referred only by its ITU-T denomination) outperforms previous standards in bit-rate reduction. In H.264 an Adaptive Deblocking Filter [2, 3] is included in the standard to reduce blocking artifacts, very common in very high compressed video streams. In fact, most video codecs use some filtering as a pre/post-processing task. The main goal of the inclusion of this kind of filter as a part of the standard was to put it inside the feedback loop in the encoding process. As a consequence, a standardized, well tuned, and inside the encoding loop filter could be designed, achieving better objective and subjective image quality for the same bit-rate. The H.264 Deblocking Filter is located in the DPCM loop as shown in Fig 1 for the decoder (it is also called Loop Filter by this reason). Exactly the same filter is used in the encoder and the decoder. The Deblocking Filter in the H.264 standard is not only a single low pass filter, but a complex decision algorithm and filtering process with 5 different filter strengths. Its objective is to maintain the sharpness of the real images while eliminating the artifacts introduced by intra-frame and inter-frame predictions at low bit-rates, while mitigating the characteristic “blurring” effect of the filter at high bit-rates.

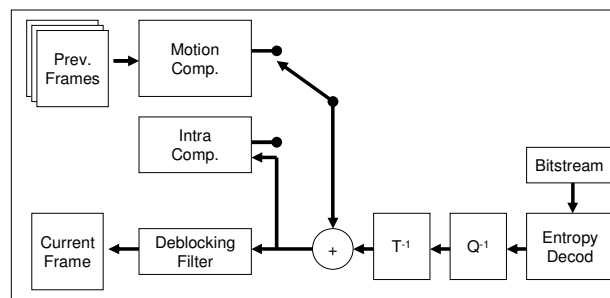


Fig. 1 – H.264 Decoder.

The filter itself is very optimized, with very small kernels of simple coefficients, but the complexity of the decision logic and filtering process makes the H.264 Deblocking Filter responsible for about one third [4] of the processing power needed in the decoder process.

In this paper an architecture for the H.264 deblocking filter is proposed. The architecture was developed focusing FPGA, aiming making a balanced use of the resources (logic, registers, and memory) available in FPGA architectures, although it can be synthesized using an ASIC flow. The performance goal was to exceed 1080p requirements when synthesized to a XILINX Virtex II FPGA. A Xilinx Virtex II pro FPGA was used to validate the developed IP.

2. Proposed architecture

In the Deblocking Filter process, the edge filter is the heart of the process. It is responsible for all filter functionality, including the thresholds and BS (Boundary Strength) calculation and filtering itself. The

remaining of the process is only sequence control and memory for block ordering. Fig. 2 presents the filter elements and the processing order of H.264 Deblocking Filter.

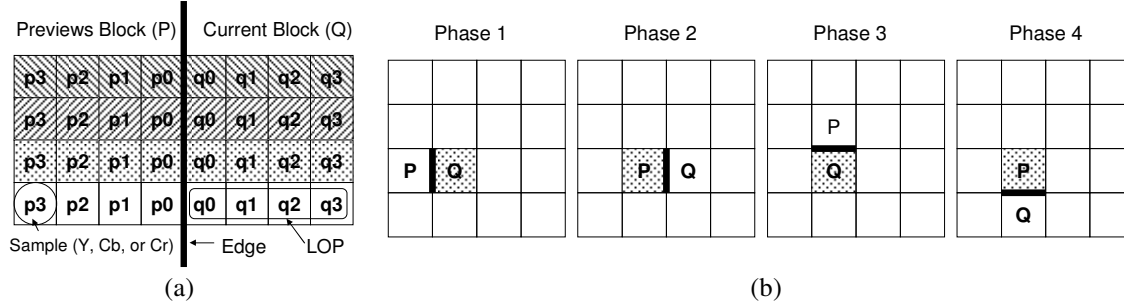


Fig. 2 – (a) 4x4 Block Conventions; (b) Border Processing Order.

The Edge Filter architecture can accept one LOP per cycle for Q and P blocks and produce the filtered Q' and P' LOP. This process is illustrated in fig. 3 (parameter lines are omitted for simplicity). Using this scheme, an entire block border will enter in the Edge Filter each four.



Fig. 3 – Edge Filter.

Based on the diagram presented in Fig. 3, a pipelined architecture for the Edge Filter was designed. As stated before, the procedure for calculation of all parameters needed to decide whether to filter or not and what BS to use requires a lot of sequential calculations (data dependencies). The pipelined architecture was designed so that only one arithmetic or logic operation is to be done every stage of the pipeline. This lead to an 11-stage pipeline only to calculate the BS, used to select the correct filter. All the filters could be done in parallel to BS calculation, so a 12 stage pipeline would be enough to achieve the maximum possible operation frequency. However, if an entire column of luma blocks (for vertical edges) that consist of four blocks stacked are processed before the next one, the first LOP of the first Q block (the uppermost) will be the first LOP of the first P block after 16 LOP cycles. If an Edge Filter architecture with a 16 stage pipeline can be designed, the output P' could be connected directly to the input Q. The chroma Cb and Cr, which are half the height (only 2 blocks tall), can be stacked so they can be processed as a luma block. This approach makes the implementation of the control logic much simpler. A 16-stage pipelined Edge Filter, presented in fig. 4, was then designed to meet the above criteria.

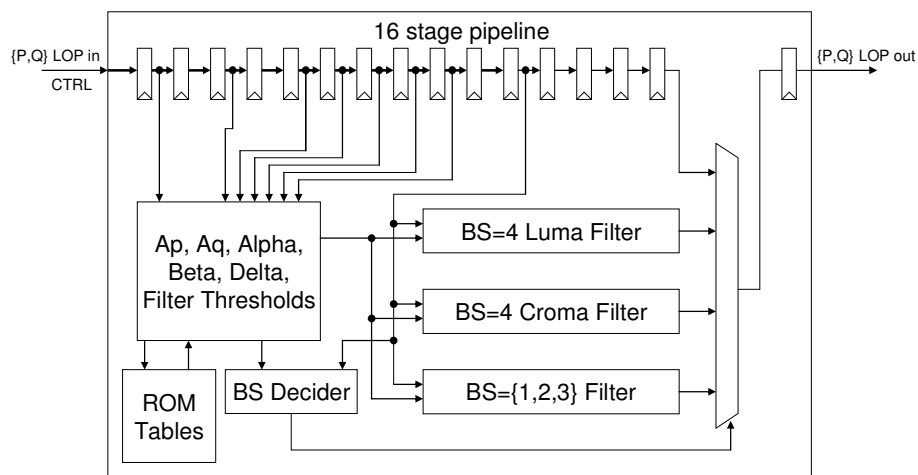


Fig. 4 – Edge Filter Architecture.

The architecture presented in Fig. 4 consumes two LOPs (one for Q and other for P blocks) and their respective parameters (QP, offsets, prediction type and neighborhood information) every clock cycle, producing two filtered LOPs (Q' and P') 16 clock cycles later.

This pipelined edge filter can be then encapsulated in such way only the Q input and P' outputs are visible, as illustrated in Fig. 5.

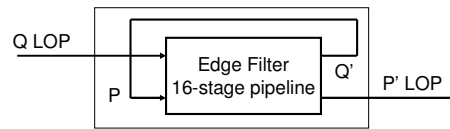


Fig. 5 – Filter encapsulation.

Using this encapsulation turns the control logic simpler, but at a cost of a small overhead: P data can only be fed by the Q input, so the first P data need to be fed 16 cycles before the filtering process can start. During this 16 cycles, the BS should be forced to 0, so that no filtering is applied to P data while they pass through the Q datapath. After finishing the processing of a macroblock, the Edge filter must be emptied, so another 16 cycle have to be spent. Fortunately, the emptying and filling phases stages can be overlapped, so the overhead is much lower.

Finally, the architecture of the proposed Deblocking Filter is presented in Fig 6. The filter has to be applied to both vertical and horizontal edges. In the proposed architecture, a single Edge Filter filters both horizontal and vertical edges. A transpose block is employed to convert vertically aligned samples in the block into horizontally aligned ones, so that horizontal edges can be filtered by the same Edge Filter.

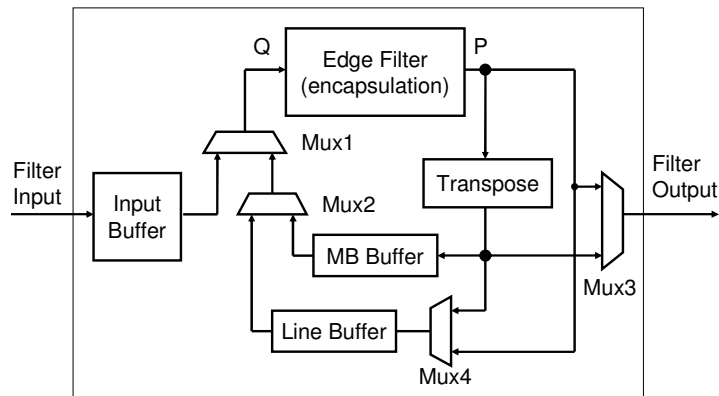


Fig. 6 – Proposed Deblocking Filter architecture.

The input buffer is needed only for data reordering, as input blocks arrive in a different order they are consumed in the deblocking process.

The MB and Line buffers are used to store blocks and block information data (QP, type, filter offset, state) which are not completely filtered (horizontal or vertical filtering is missing). The size of MB buffer is 32 blocks (512 samples plus 32 block information data) and the line buffer depends on the maximum frame size the filter is supposed to process (7680 samples, plus 480 block information data for a 1080p HDTV frame).

3. Implementation and Results

The architecture presented in Section 2 was described in VHDL. About 3,500 lines of code were written. The design behavior was validated by simulation using some testbench files and data extracted from the JVT reference software using some public domain video sequences. The validated behavioral design was then synthesized, the post place and route was validated and performance results were obtained for a Xilinx Virtex2-pro FPGA.

Using the reference software CODEC, extracted data before and after the Deblocking Filter process was used to ensure the correctness of the implemented architecture. Tab. 1 presents the number of Xilinx LUTs and BRAMs used to synthesize the developed Deblocking Filter. Observe the balance between the amount of logic (LUTs) and memory employed, related to the total amount available in the target device.

Tab. 1 - Synthesis results.

Device	XC2VP30	XC5VLX30
LUTs	4008/27392 (14%)	4275/19200 (21%)
BRAMs	20/136 (14%)	7/32 (21%)
Fmax (MHz)	148	197
FPS@1080p	71	95

Running at 148MHz in the Virtex II Pro device, this implementation is 2.36 times faster than the requirement for HDTV (1080p). For the Virtex-5, running at 197MHz, it is 3.14 times the requirement for HDTV. This IP can be used to build an encoder or decoder for a system with the following characteristics:

- Ultra-high definition (4K x 2K pixel @ 24fps);
- High definition stereo video (two HDTV streams);
- Multi stream surveillance (up to 2K CIF streams);
- Scalable high definition;
- Low-power HDTV, where the device can operate at lower frequency, lower voltages and still achieve HDTV requirements.

Tab. 2 presents some performance comparison. The IP implemented with the developed architecture only loses to [5], but [5] do not include the external memory access penalty needed to obtain the upper MB data.

Tab. 1 - Literature comparison.

	Cycles/MB	Memory type	fps@1080p
our	256	Dual-port	95
[4]	variable	Two-port	45
[5]	96	Two-port	100
[6]	192	Dual-port	30

The maximum resolution achievable by the proposed architecture is only limited to the size of the line buffer (Fig. 6) implemented. This buffer represents a significant amount of the total memory employed by this design and impacts the number of Block RAM (BRAM) used by the IP. The results presented in Tab 1 is for a 2048 pixel wide frame, including the memory necessary to store block parameters, needed by the BS decision process. The maximum picture width is determined by a parameter in the synthesizable code and the height is unlimited.

4. Conclusion

This work presented a high performance architecture for H.264 Deblocking Filter IP targeted to exceed HDTV requirements in FPGA. The primary contribution of this work was the high performance deep pipeline architecture employed to improve the speed in the Boundary Strength decision and at the same time reducing the control logic. The proposed architecture stores all intermediate information in its own memory, differently from most works in literature that rely on external memory to store some blocks not completely filtered in a line of macroblocks. The developed IP based on the proposed architecture was synthesized to a Virtex II Pro and for a Virtex 5 device and prototyped in a XUP-V2Pro (Virtex II-Pro XC2VP30 device). Results showed its capability to exceed the processing rate for HDTV, reaching 71 frames per second in the Virtex II Pro device and 95 frames per second in the Virtex 5 device at 1080p (1920x1080) resolution.

Future work will address the support for high profiles, scalability and multi-view amendments of the H.264 standard, which require small modification in the BS decision logic and in the data width for pixel values.

5. References

- [1] Draft ITU-T Recommendation and Final Draft international Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC), March 2003.
- [2] P. List, A. Joch, J. Lainema, G. Bjotergaard, and M. Karczewicz, Adaptive deblocking filter, IEE trans. Circuits Syst. Video Technol., Vol. 13, pp. 614-619, July 2003.
- [3] A. Puri, X. Chen, A. Luthra, Video coding using the H.264/MPEG-4 AVC compression standard, Signal Processing: Image Communication, Elsevier, n. 19, pp. 793-849, 2004.
- [4] M. Sima, Y. Zhou, W. Zhang. An Efficient Architecture for Adaptive Deblocking Filter of H.264/AVC Video Coding. Trans. On Consumer Electronics, IEEE, Vol. 50, n. 1, pp. 292-296. Feb. 2004.
- [5] H-Y Lin; J-J Yang; B-D Liu; J-F Yang. Efficient deblocking filter architecture for H.264 video coders. 2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006, 21-24 May 2006.
- [6] G. Khurana; A.A. Kassim; T-P. Chua; M.B.A Mi Pipelined hardware implementation of in-loop deblocking filter in H.264/AVC IEEE Transactions on Consumer Electronics. V.52, I. 2, May 2006. pp.536-540.

A Hardware Architecture for the New SDS-DIC Motion Estimation Algorithm

¹Marcelo Porto, ²Luciano Agostini, ¹Sergio Bampi
{msporto,bampi}@inf.ufrgs.br, agostini@ufpel.edu.br

¹Instituto de Informática - Grupo de Microeletrônica (GME) – UFRGS
²Grupo de Arquiteturas e Circuitos Integrados (GACI) – DINFO – UFPel

Abstract

This paper presents a hardware architecture for motion estimation using the new Sub-sampled Diamond Search algorithm with Dynamic Iteration Control (SDS-DIC). SDS-DIC is a new algorithm proposed in this paper, which was developed to provide an efficient hardware design for motion estimation (ME). SDS-DIC is based on the well known Diamond Search (DS) algorithm and on the sub-sampling technique. In addition, it has dynamic iteration control (DIC) that was developed to allow better synchronization and quality. The hardware architecture implementing this algorithm was described in VHDL and mapped to a Xilinx Virtex-4 FPGA. This architecture can reach real time for HDTV 1080p in the worst case, and it can process 120 HDTV frames per second in the average case.

1. Introduction

Motion estimation (ME) is the most important task in the current standards of video compression. Full Search (FS) is the most used block matching algorithm for hardware implementation of ME, due to its regularity, easy parallelization and good performance. However, hardware architectures for FS algorithm require tens of processing elements and lots of resources (of the order of hundred thousand gates) to achieve real time, mainly for high resolution videos. The Diamond Search (DS) [1] algorithm can drastically reduce the number of SAD calculations when compared to FS algorithm. Using DS algorithm it is possible to increase significantly the search area for matching, generating quality results close to the FS results. Another strategy to reduce the use of hardware resources and to increase the throughput is the Pixel Sub-sampling (PS) technique [1]. Using PS at a 2:1 rate, it is possible to reduce the number of SAD calculations by 50% and to speed up the architecture.

This paper presents a high throughput architecture for ME using a new algorithm named Sub-sampled Diamond Search with Dynamic Iteration Control (SDS-DIC), which is based on the DS algorithm and uses PS 2:1 and DIC. This architecture was described in VHDL and mapped to Xilinx Virtex-4 FPGA, and to TSMC 0.18 μ m CMOS digital standard cells. This architecture can reach real time for HDTV 1080p videos with a low hardware cost.

2. SDS-DIC Search Algorithm

This paper proposes SDS-DIC, a new algorithm for motion estimation, based on our previous work [2]. This algorithm is based on three main principles: (1) Diamond Search algorithm; (2) 2:1 Pixel-Subsampling technique and (3) Dynamic Iteration Control. The two first principles are well known, thus only the third one will be detailed herein.

The Dynamic Iteration Control (DIC) was designed focusing in the hardware design of ME. This feature allows a good tradeoff among hardware resources consumption, throughput, synchronization and quality. Through the software analysis of 10 video samples, it was possible to conclude that the DS algorithm uses an average number of 3 iterations while, in the worst case, DS may use up to 89 iterations. This analysis considered a search area with 208x208 video samples and blocks of 16x16 samples. This number of iterations must be restricted to guarantee the performance requirements for real time. This characteristic of number of iterations was explored in this paper through DIC design.

The restriction in the number of iterations is important to make it easier the synchronization of this module with other video coder modules. Doing it dynamically at the same time allows the number of cycles used to generate a motion vector to be optimized to reach results close to the longest time (worst case) whenever possible. This restriction is also important to allow the best compromise between high throughput and a low hardware cost. However a static restriction can cause an important degradation in the quality of the results. Then a dynamic restriction is considered in DIC. A variable stores the number of iterations used by the generation of the 16 last motion vectors. Each motion vector has a limited number of 10 iterations to get calculated, much more than the average 3. When a motion vector uses less iteration than the maximum available for it, the saved iterations are accumulated and this extra slack is available to be used in the estimation of the next motion vector.

3. Software Evaluation

This section presents a software evaluation of several algorithms: FS, FS with 2:1 PS, DS, QSDS-DIC [2] and SDS-DIC. The main results of the software analysis are shown in tab. 1. The search algorithms were developed in C and the results for quality and computational cost were generated. The search area used was 64x64 pixels (except for DIC versions that have a dynamic search area) with block size of 16x16 pixels. The SDS-DIC algorithm was restricted to 160 iterations (maximum), the number of iterations used to generate 16 motion vectors and this means that, in the best case, the SDS-DIC algorithm can reach a search area of 660x660 samples. The algorithms were applied to the first 100 frames of 10 real SDTV video sequences at a 720x480 pixels resolution.

The quality results were evaluated through the averages of PSNR and percentage of error reduction (PER). The PER is measured comparing the results generated by the ME process with the results generated by the simple subtraction between the reference and the current frames. Tab. 1 also presents the average number of SAD operations used by each algorithm.

Tab.1 – Results for Software Analysis

Search Algorithm	PER (%)	PSNR (dB)	# of SADs ($\times 10^9$)
FS	54.66	28.483	82.98
2:1 FS	54.32	28.425	41.49
DS	48.63	27.233	0.85
QSDS-DIC	43.64	26.030	0.21
SDS-DIC	48.58	27.195	0.42

SDS-DIC presents an error reduction 0.05% lower than the DS algorithm. The PSNR is roughly the same (only 0.038dB lower), while the SAD operations are reduced by a factor of two. This is a large reduction at a negligible penalty. Compared to the previous QSDS-DIC algorithm [2], the SDS-DIC increases the PER by almost 5%, and the PSNR by more than 1 dB. SDS-DIC algorithm causes a loss of 1.288dB in PSNR when compared to the FS results, however, it reduces the number of SAD operations by more than 197 times – a considerable savings.

4. Designed Architecture

The designed architecture for the SDS-DIC algorithm uses SAD [1] as a distortion criterion. The architectural block diagram is shown in fig. 1. The architecture has nine processing unities (PU). The PU can process eight samples in parallel and this means that one line of the 16x16 2:1 sub-sampled block is processed in parallel. So, sixteen accumulations must be used to generate the final SAD result of each block, one accumulation per line.

The nine candidate blocks of the Large Diamond Search Pattern (LDSP) [1] are calculated in parallel and the results are sent to the comparators. Each candidate block receives an index, to identify the position of this block in the LDSP. The comparator finds the lowest SAD and sends the index of the chosen candidate block to the control unit. The control unit analyses the block index and decides the next step of the algorithm. If the chosen block was found in the center of the LDSP, the control unit starts the final refinement with the Small Diamond Search Pattern (SDSP). In the SDSP calculation, four additional candidate blocks are evaluated. The lowest SAD is identified and the control unit generates the motion vector for this block.

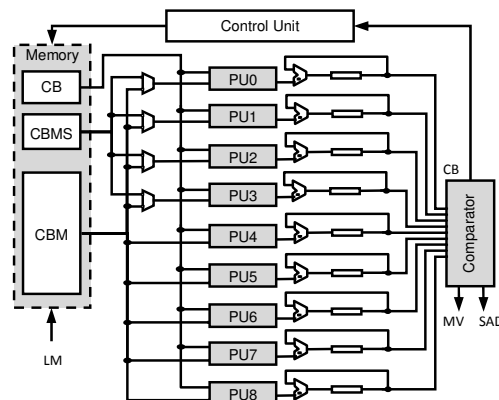


Fig. 1 – SDS-DIC block diagram architecture.

When the chosen block is not in the center of LDSP, the control unit starts the second step of the algorithm with a search for vertex (five additional candidate blocks) or for edge (three more candidate blocks) [1]. The second step can be repeated n times, where n is controlled dynamically.

4.1. Dynamic Iteration Control

As mentioned before, it is clear that a restriction in the number of iterations, available in the architecture, must be implemented to ensure the desired throughput. To solve this problem DIC was developed. The architecture designed to implement DIC is shown in fig. 2.

The number of iterations used is sent to a shift register with 16 positions. A set of add/accumulator generates the total number of iterations used in the last 16 motion vectors. The DIC was developed allowing a maximum of 10 iteration per motion vector, so the used iterations for the last 16 vectors are subtracted by 160 (maximum value for a set of 16 motion vectors), and the result is the number of available iterations for the next motion vector generation. The SDS-DIC can use a maximum search area of 660x660 pixels.

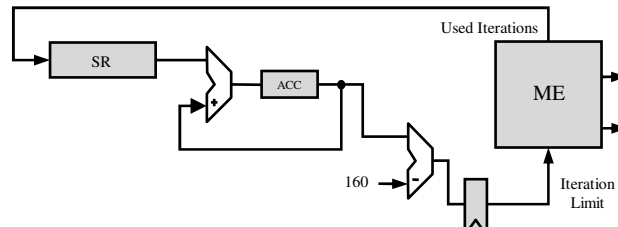


Fig. 2 – Block diagram of DIC.

4.2. Performance Evaluation

SDS-DIC algorithm stops the search when the best match is found at the center of the LDSP. This condition can occur in the first application of the LDSP. Thus, no searches for edge or vertex are done. This is the best case, when only thirteen candidate blocks are evaluated: nine from the first LDSP and four from the SDSP. This architecture uses 36 clock cycles to fill all the memories and to start the SAD calculation. The PUs have a latency of four cycles and they need fifteen extra cycles to calculate the SAD for one block. The comparator uses five cycles to choose the lowest SAD. Thus, 60 clock cycles are necessary to process the first LDSP. The SDSP needs only 28 cycles to calculate the SAD for the four candidates block. So, in the best case, this architecture can generate a motion vector in 88 clock cycles.

In the cases where the LDSP is repeated, for an edge or vertex search, 10 cycles are used to write in the CBMs. The same 24 cycles are used by the PUs and the comparator. Both edge and vertex search use the same 34 cycles. SDS-DIC algorithm can perform 160 LDSP repetitions (iterations) to generate 16 motion vectors, in an average of 10 iterations per motion vector. For each iteration, other 34 cycles are used.

5. Synthesis Results

The proposed hardware architecture was described in VHDL. ISE 8.1i tool was used to synthesize the architecture to the Virtex-4 XC4VLX15 device and ModelSim 6.1 tool was used to validate the architecture design. Tab. 2 presents the synthesis results. The device resources utilization is small, as percentage of used resources in the FPGA. The synthesis results show the high frequency achieved by the SDS-DIC architecture.

Tab. 2 also shows the architecture performance considering HDTV 1080p videos. SDS-DIC architecture can process, in the worst case, 53 HDTV 1080p frames per second. Considering the DIC with 10 iterations per vector, the architecture can use, in the worst case, 6848 clock cycles to generate 16 motion vectors, which represents an average of 428 cycles per motion vector.

Tab.2 – Synthesis Results for SDS-DIC architecture

Parameter	SDS Results
Slices	2190 (35%)
Slice FF	2379 (19%)
LUTs	3890 (31%)
Frequency (MHz)	184.19
HDTV fps (worst case)	53
HDTV fps (average case)	120
Device: Virtex-4 XC4VLX15	

The average case was obtained from the software implementation. The algorithm uses an average of three iterations in the second step of iterations. Then, the SDS-DIC architecture needs 190 clock cycles to generate a motion vector, in the average case. This implies a processing rate of 120 HDTV 1080p frames per second.

A standard cells version of SDS-DIC architecture was also generated to allow a comparison with other published designs. Leonardo Spectrum was used to generate the standard-cells version for TSMC 0.18 μ m technology.

SDS-DIC results, for FPGA and standard-cells (SC) implementations, were compared in tab.3 to QSDS-DIC [2], FTS (Flexible Triangle Search) [3], FS+PS4:1 and early termination architecture [4], PVMFAST [5]

and FS [6] and [7]. Architecture [2] has a variable search area, as the SDS-DIC. The architectures presented in [3] and [6] do not present the search area width. The search area used by [4] is 32x32 samples, and [7] uses a search area of 46x46 samples. The chip area comparison is presented in number of CLBs (FPGAs) or number of gates (standard-cells). The processing rate of HDTV 1080p frames is also presented.

SDS-DIC architecture is the fastest one among the compared designs for FPGA, except for our previous work [2]. However, the algorithm used in [2] presents quality degradation close to 5% compared to the SDS-DIC algorithm. Due to the efficiency of the SDS-DIC algorithm, the designed architecture uses less hardware resources than other solutions, reaching the highest throughput among then, keeping quality results close to the FS algorithm.

Tab.3 - Comparison between architectures

Solution	CLBs	Gates (K)	Frequency (MHz)	HDTV (fps)
Porto, 2008 [2]	492	-	213.3	*188
Rehan, 2006 [3]	939	-	109.8	1.66
Larkin, 2006 [4]	-	10.1	120.0	*24
Li, 2005 [5]	6142		74.0	45
Gaedke, 2006 [6]	-	1200	334	25
He, 2007 [7]	-	1020	100	46
SDS-DIC (FPGA)	547	-	184.1	*120
SDS-DIC (SC)	-	30.7	259	*168
*Average throughput				

The SDS-DIC architecture uses 39 and 33 times less gates than [6] and [7] respectively. And it achieves a higher throughput. SDS-DIC architecture presents gate utilization higher than [4], which is based in fast algorithms and considers small search areas. The search area used in SDS-DIC architecture can be 425 times higher than that of [4]. SDS-DIC presented the best tradeoff between hardware usage, quality and throughput among all compared works.

6. Conclusions

This paper presented a hardware architecture for the Sub-sampled Diamond Search algorithm with Dynamic Iteration Control, named SDS-DIC. Software analyses show that this algorithm can reduce drastically the number of SAD operations with a very small degradation in quality when compared with the Full Search algorithm.

The developed architecture is able to run at 184.1 MHz in a Xilinx Virtex 4 FPGA. In the worst case, the architecture can work in real time at 53 fps for HDTV 1080p videos. In the average case, it is able to process 120 HDTV fps. The synthesis results showed that our design architecture can achieve high throughput with less hardware, keeping the quality of the estimated motion vectors.

7. References

- [1] P. M. Kuhn, Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Springer, June 1999.
- [2] M. Porto, L. Agostini, A. Susin and S. Bampi, "Architectural Design for the New QSDS with Dynamic Iteration Control Motion Estimation Algorithm Targeting HDTV," ACM 21st annual Symposium on Integrated Circuits and System Design, Gramado, Brazil, 2008, pp. 216-221.
- [3] M. Rehan, M. Watheq El-Kharashi, P. Agathoklis, and F. Gebali, "An FPGA Implementation of the Flexible TriangleSearch Algorithm for Block Based Motion Estimation," IEEE International Symposium on Circuits and Systems, Island of Kos, Greece, 2006. pp. 521-523.
- [4] D. Larkin, V. Muresan and N. O'Connor, "A Low Complexity Hardware Architecture for Motion Estimation," IEEE International Symposium on Circuits and Systems, Island of Kos, Greece, 2006, pp. 2677-2688.
- [5] T. Li, S. Li and C. Shen, "A Novel Configurable Motion Estimation Architecture for High-Efficiency MPEG-4/H.264 Encoding," IEEE Asia and South Pacific Design Aut. Conf., Shanghai, China, 2005, pp 1264-1267.
- [6] K. Gaedke, M. Borsum, M. Georgi, A. Kluger, J. Le Glanic and P. Bernard, "Architecture and VLSI Implementation of a programmable HD Real-Time Motion Estimator," IEEE International Symposium on Circuits and Systems, New Orleans, USA, 2007. pp. 1609-1612.
- [7] W. He and Z. Mao, "An Improved Frame-Level Pipelined Architecture for High Resolution Video Motion Estimation," IEEE Int. Symp. on Circ. and Systems, New Orleans, USA, 2007. pp. 1381-1384.

SDS-DIC Architecture with Multiple Reference Frames for HDTV Motion Estimation

¹Leandro Rosa, ¹Débora Matos, ¹Marcelo Porto, ¹Altamiro Susin, ¹Sergio Bampi,
²Luciano Agostini

{leandrozanetti.rosa, debora.matos, msporto, bampi}@inf.ufrgs.br,
altamiro.susin@ufrgs.br, agostini@ufpel.edu.br

¹Universidade Federal do Rio Grande do Sul

²Universidade Federal de Pelotas

Abstract

This paper presents the design of a motion estimation architecture with multiple reference frames, targeting HDTV (High Definition Television). The designed architecture uses SDS-DIC algorithm (Sub-sampled Diamond Search with Dynamic Iteration Control) and the architectural template designed in previous works for this algorithm. The designed solution used the first three past frames as reference frames to choose the best motion vector. The results show improvements in quality in relation to the original algorithm which uses only one reference frame. The architecture was described in VHDL and synthesized to Xilinx Virtex-4 FPGA. Synthesis results show that the architecture is able to run at 159.3 MHz and it can process up to 39 HDTV frames per second in the average case.

1. Introduction

Digital videos use a large amount of data to be stored and represented, which hinders their handling. Video compression allows the efficient use of digital videos, because it significantly reduces the amount of data that must be stored and transmitted. Thus, the video compression is essential, especially for mobile devices.

Among the video compression standards, the H.264/AVC [1] is the latest one and it offers the greatest gain in terms of compression rates. Motion estimation (ME) is an operation in video compression that is responsible to find the best matching among the current frame (frame that is being coded) blocks and the blocks of other previously coded frames, called reference frames. When the best matching is found, the ME maps the position of this block in the reference frame through motion vectors. ME uses search algorithms to find the best way to compare the blocks of a video frame.

In [2] and [3] are presented the SDS-DIC algorithm and an architectural solution for high performance for this algorithm. However, this architecture uses only one reference frame. The architecture designed in this work uses the first three past frames as reference frames, to improve the motion vectors quality in relation those previous works.

This paper is organized as follows: Section 2 presents some basic concepts about motion estimation. Section 3 presents the SDS-DIC algorithm and the fourth section shows the designed architecture. The results are described in section 5 and conclusions in section 6.

2. Motion estimation

The motion estimation operation is responsible to find, in the reference frames, the block that is most similar to the current block. When this block is found, it is generated a motion vector (MV) and a frame index, to identify the position of this block. These two identifiers are coded together with the video content. Each block of the current frame generates one MV and one frame index. The MV shows the position of the block in the reference frame and the frame index identifies the frame used as reference.

It is necessary to define a search area in the reference frames for each block of the current frame to reduce the processing time. A search algorithm is used by the ME to determine how the block is moved within the search area. This is an operation with great degree of computational complexity, however, it generates a very significant reduction in the amount of data necessary to represent the video, since all the necessary information to represent a block may be replaced by a index of three dimensions and the residual information. The residual information is the difference between the block of the current frame and block that showed highest similarity within the search area of the reference frame.

In this paper, were used blocks with 16x16 samples and a search area that can reach 660 x 660 samples.

3. SDS-DIC Algorithm

The search algorithm used to the ME architecture designed in this paper was based on Diamond Search (DS) [4], using Pixel Sub-sampling (PS) in a 2:1 proportion and a Dynamic Iteration Control (DIC) [2] [3] to limit the number of algorithm iterations.

The algorithm SDS-DIC (Sub-sampled Diamond Search with Dynamic Iteration Control) uses two diamond-shaped patterns, the LDSP (Large Diamond Search Pattern) and the SDSP (Small Diamond Search Pattern), used in the initial and final step of the algorithm, respectively. These patterns are shown in fig. 1(a). In the first step, the LDSP is applied and nine comparisons are made. If the best result is found in the center of the diamond, the SDSP is applied and the search ends. But if the best result is found in one of the edges or vertexes of the diamond, the LDSP is repeated with a search for edge or vertex [4] [5].

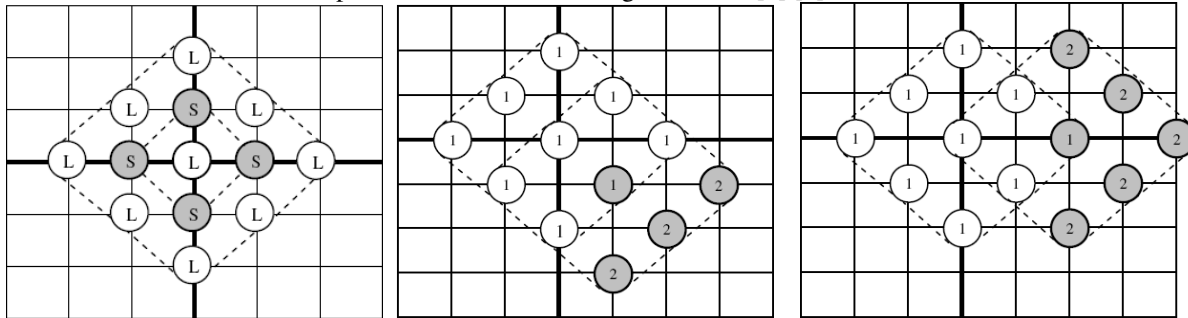


Fig. 1 – Search format: (a) LDSP and SDSP (b) edge (c) vertex.

When a search is performed by edge, more three new values will be obtained to form a new diamond on the new center, as showed in fig. 1(b). If a search is performed by vertex, more five values are required in order to form the new LDSP, as showed in fig. 1(c). The pattern replication is applied until the best result is found at the center of LDSP. Only then the SDSP is applied, comparing additional four positions around the center of the LDSP and, finally, MV is generated considering the position with the lower difference.

SDS-DIC differs from DS because SDS-DIC limits the number of LDSP repetitions. This restriction is important for limit number of clock cycles used to create the MV, avoiding the performance decrease caused by a very high number of iterations and allowing a best synchronization with other encoder architecture modules, since the maximum number of iterations are reduced and fixed. This limitation is controlled by a variable that stores the iterations number used by last 16 generations of MVs. Each vector has a limited number of 10 iterations, but when the ME uses less than 10 iterations to generate a MV, the excess of iterations are reserved to be used in the generation of the next MVs.

4. Designed Architecture

In this paper, as explained above, a new architecture was designed, based on SDS-DIC algorithm [2] [3], but using multiple reference frames. The designed architecture is shown in fig. 2 and it consists of three memory blocks, nine processing unities (UP), nine adders, nine accumulators, two comparators and a four control blocks.

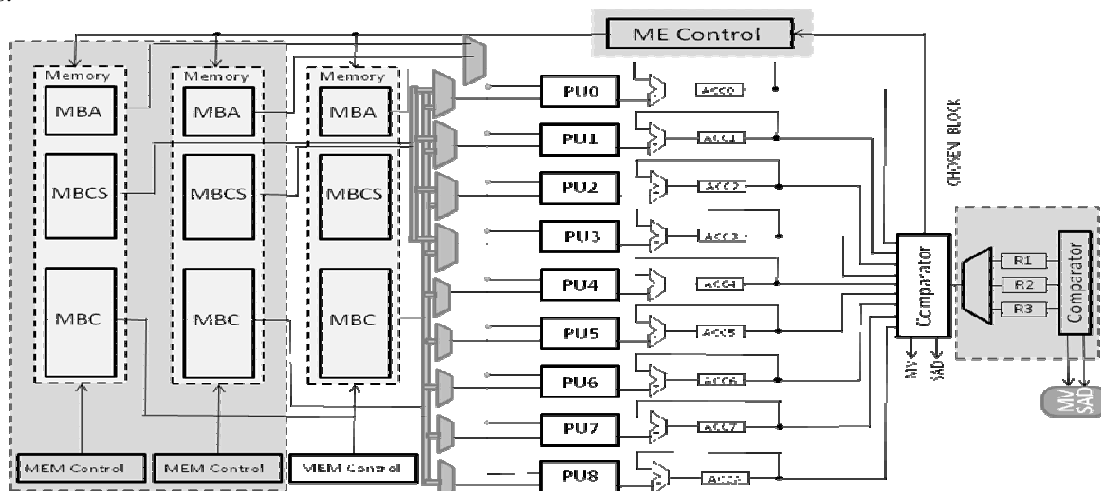


Fig. 2 – SDS-DIC architecture for multiple reference frames.

The three memory blocks are composed of an MBA (Actual Block Memory), a MBC (Candidate Block Memory) and a MBCS (SDSP Candidates Block Memory). The MBA contains the block of the current frame, in other words, the frame that is being predicted from the reference frame. The MBC stores the LDSP candidate blocks, while the MBCS stores the candidate blocks that are used in SDSP.

In the first step, the nine LDSP blocks are compared with the candidate blocks in the nine parallel PUs. When the SDSP is activated, only the first 4 PUs will receive the next candidate blocks that are stored in the MBCS memory. Initially, eight samples of the current block and the candidate block are subtracted. Then the module of the difference between these samples is added and the final result of the differences accumulation, for all samples of the line, is stored in the register output.

A set adder/accumulator is used in the PUs output to store the intermediate results of each candidate block. 16 accumulations are necessary to generate the final result of the SAD for a candidate block, one accumulation for each line block. The PU output, stored in accumulators (ACC0 to ACC8 in fig. 2) is sent to the comparator, which analyzes and finds the lowest SAD among the candidate blocks. Finally, these data are sent to the ME control. When the final refinement is performed, the chosen block is the best of that reference frame that is being analyzed and the result is stored in one of the registers: R1, for the first reference frame, R2 for the second frame, and R3 for the third reference frame. Finally, the comparator selects the best candidate block among the three registers and sends the data block for the global control of the ME, allowing the three memory to be loaded with new data, related to the next block that needs to be encoded.

5. Results

A software evaluation was done to compare the single reference frame SDS-DIC with the multiple reference frames SDS-DIC designed in this work and the Full Search algorithm considering a search area of 46x46 samples [7]. Full Search is the algorithm that obtains the optimal result for a given search area and then, is a good evaluation reference. The algorithms were developed in C language and applied to ten video sequences [6]. Tab. 1 shows the average quality results, considering the PSNR (Peak-to-Signal Noise Ratio) and the RDA (Reduction of Absolute Difference) criteria. Tab. 1 also presents the average computational costs obtained by the algorithms in terms of SAD (Sum of Absolute Differences) calculations.

The RDA is calculated comparing the difference between the neighboring frames before and after the ME application. The number of SAD operations was measured in Gops (billions of operations).

Tab. 1 – Results of quality and computational cost

Algorithms	Criteria of Quality Evaluation		
	RDA (%)	PSNR (dB)	SAD (Gops)
Multiple References SDS-DIC	53.49	28.09	1.47
Single Reference SDS-DIC [3]	48.19	27.16	0.42
Full Search 46x46 [7]	51.80	28.26	33.21

In RDA terms, the use of multiple references provides a gain of more than 5% in comparison with the previous version. The RDA of this new version also exceeds the value of the Full Search RDA in almost 2%, which shows the additional capacity generated by the use of multiple reference frames.

The second quality evaluation criterion was the PSNR, which is the most used criterion to measure the quality of image and video processing. The results show one more time improvements in relation to the single frame SDS-DIC. This improvement is of 0.9 dB, which is an important increase in the quality. But, in this criterion, the multiple references SDS-DIC does not improve the results achieved by the Full Search.

The third evaluation presented in tab. 1 is the computational cost. This criterion is important to evaluate the impact that results improvements have on the architecture size. The use of multiple reference frames greatly increases the number of SAD operations required for the generation of a MV. This value is increased from 0.42 Gops to 1.47 Gops. On the other hand, comparing this result with the Full Search, the computational cost is still more than 22 times lower, although the Full Search using one reference frame and a small search area (46 x 46 samples) [7]. After this software analysis, the architecture was designed and described in VHDL using the Xilinx ISE 8.1i. Mentor Graphics ModelSim 6.1 tool was used in the architecture validation.

Tab. 2 presents the synthesis results of the designed architecture. The synthesis was target to the Xilinx Virtex4 XC4VLX200 FPGA. Tab. 2 also presents the comparative results with the single reference SDS-DIC architecture and two other designs that use the single reference Full Search algorithm [8] [9].

The device hardware utilization is small, especially in memory terms. The reached frequency allows a processing rate of 39 HDTV frames (1920 x 1080 pixels) per second in average and 16 HDTV frames per second in the worst case. With this performance, even using almost three times more cycles, it is still possible to run more than 30 HDTV frames per second, which is enough to run a video in real time.

Tab. 2 - FPGAs comparatives results

Architectures	Synthesis Results								
	Flip-Flops	LUTs	BRAM	Slices	CLBs	Freq. (MHz)	RDA (%)	PSNR (dB)	Frames HDTV
Multiple Reference SDS-DIC	4597	7312	96	4154	1038	159.3	53.49	28.09	39
Single Reference SDS-DIC [3]	2020	3541	32	1968	492	185.7	48.19	27.16	120
Single Reference FS [8]	-	-	-	-	19194	123.4	50.20	27.25	5
Single Reference FS [9]	-	-	-	-	939	109.8	-	-	1.66

Tab. 2 also shows, an increase use of resources in relation to [3]. This was expected, because several architectural modules have been expanded. Even so, the use of resources is much lower than that used by [8], which uses 18 times more CLBs. The architecture designed in this work uses 10% more resources than [9], but our work reached a higher processing rate. Besides the expressive gain in terms of resources usage, the designed architecture achieved better results in RDA, PSNR and frame rates when compared with the FS architectures [8] and [9]. In relation to the single references SDS-DIC architecture, the increase in the hardware resources consumption is balanced by the gain in terms of the quality of the generated results.

6. Conclusions

This paper presented an architectural design for the SDS-DIC ME algorithm using multiple reference frames. This architecture can process till 39 HDTV frames per second, which is enough to process 1080p HDTV videos in real time.

The quality results showed that the use of multiple reference frames generates significant gains in comparison with the single reference version of this algorithm. The designed architecture presents high quality results, also when compared with FS solutions, using up to 18 times less FPGA hardware resources.

7. References

- [1] I. Richardson, H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia. Chichester: John Wiley and Sons, 2003.
- [2] M. Porto, et al. "Architectural design for the new QSDS with dynamic iteration control motion estimation algorithm targeting HDTV," In: 21st Symp. on Integrated Circuits and System Design, 2008, Gramado.
- [3] M. Porto, et al. "A high throughput and low cost diamond search architecture for HDTV motion estimation," In: IEEE International Conference on Multimedia & Expo, 2008, Hannover.
- [4] P. Kuhn, Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Springer, June 1999.
- [5] X. Yi and N. Ling, "Rapid block-matching motion estimation using modified diamond search algorithm," In: IEEE International Symposium on Circuits and Systems. Kobe: IEEE, 2005, p. 5489 – 5492.
- [6] VQEG. The Video Quality Experts Group Web Site. Disponível em: <www.its.bldrdoc.gov/vqeg/>. Acesso em: abr. 2007.
- [7] M. Porto, et al. "Investigation of motion estimation algorithms targeting high resolution digital video compression," In: ACM Brazilian Symposium on Multimedia and Web, WebMedia, 2007. New York: ACM, 2007. p. 111-118.
- [8] M. Porto, et al. "High throughput hardware architecture for motion estimation with 4:1 pel subsampling targeting digital television applications" In: IEEE Pacific-Rim Symposium on Image Video and Technology, 2007, Santiago do Chile. IEEE PSIVT 2007.
- [9] M. Mohammadzadeh, M. Eshghi, and M. Azdfar, "An Optimized Systolic Array Architecture for Full-Search Block Matching Algorithm and its-Implementation on FPGA chips," The 3rd International IEEE-NEWCAS Conference, 2005, pp.174-177.

Author Index

- Agostini, Luciano: 95, 203, 207, 215, 219, 223, 227, 231, 235, 239, 247, 251, 259, 263
Aguirre, Paulo César: 83
Almança, Eliano Rodrigo de O.: 75
Almeida, Sergio J. M. de: 67
Altermann, João S.: 63
Bampi, Sergio: 79, 119, 141, 203, 207, 211, 215, 227, 235, 247, 251, 255, 259, 263
Bem, Vinicius Dal: 51
Beque, Luciéli Tolfo: 183
Berejuck, Marcelo Daniel: 165
Bonatto, Alexsandro C.: 149
Braga, Matheus P.: 197
Butzen, Paulo F.: 123
Callegaro, Vinicius: 27
Camaratta, Giovano da Rosa: 119
Campos, Carlos E. de: 55, 59
Carro, Luigi: 133, 179, 187
Cassel, Dino P.: 157
Conrad Jr., Eduardo: 119
Corrêa, Guilherme: 231, 247
Costa, Eduardo A. C. da: 63, 67
Costa, Miklécio: 173
Cota, Érika: 179, 183, 187, 197
Cruz, Luís A.: 231
Deprá, Dieison Antonello: 211
Dessbesel, Gustavo F.: 83
Dias, Y.P.: 101
Diniz, Cláudio Machado: 227, 251
Dornelles, Robson: 219, 223, 239
Fachini, Guilherme J. A.: 179
Ferreira, Luis Fernando: 119
Ferreira, Valter: 193
Flach, Guilherme: 31
Foster, Douglas C.: 87
Franck, Helen: 79
Freitas, Josué Paulo de: 83
Gervini, Vitor I.: 91, 153, 157
Ghissoni, Sidinei: 129
Girardi, Alessandro: 41, 129
Gomes, Humberto V.: 179
Gomes, Sebastião C. P.: 91, 153, 157
Guimarães Jr., Daniel: 79, 137
Güntzel, José Luís: 55, 59, 79, 145
Jaccottet, Diego P.: 67
Johann, Marcelo:, 15
Kozakevicius, Alice: 87
Leonhardt, Charles Capella: 23
Lima, Carolina: 31
Lorencetti, Márlon A.: 243
Lubaszewski, Marcelo: 197
Marques, Jeferson: 129
Marques, Luís Cléber C.: 105, 109
Martinello Jr, Osvaldo: 133
Martins, João Baptista: 83
Martins, Leonardo G. L.: 87
Matos, Debora: 141, 263
Mattos, Júlio C. B.: 71, 75, 95
Matzenauer, Mônica L.: 67
Medeiros, Mariane M.: 157
Melo, Sérgio A.: 63
Mesquita, Daniel G.: 87
Monteiro, Jucemar: 55, 59
Moraes, Fernando G.: 161
Moreira, E. C.: 101
Moreira, Tomás Garcia: 137
Müller, Cristian: 83
Neves, Raphael: 129
Nihei, Gustavo Henrique: 145
Nunes, Érico de Morais: 19
Obadowski, V. N.: 101
Palomino, Daniel: 219, 223, 239
Paniz, Vitor: 109
Pereira, Carlos Eduardo: 137
Pereira, Fabio: 235, 243
Pieper, Leandro Z.: 67, 83
Pinto, Felipe: 31, 187
Pizzoni, Magnos Roberto: 169
Porto, Marcelo: 259, 263
Porto, Roger: 215
Pra, Thiago Dai: 183
Raabe, André: 45
Ramos, Fábio Luís Livi: 133
Rediess, Fabiane: 247
Reimann, Tiago: 37
Reinbrecht, Cezar R. W.: 161
Reis, André: 27, 51, 113, 123
Reis, Ricardo: 15, 23, 31, 37, 79, 137, 187
Ribas, Renato: 27, 51, 113, 123
Ribas, Robson: 55, 59
Rosa Jr, Leomar S. da: 27
Rosa, André Luís R.: 91
Rosa, Leandro: 141, 203, 263
Rosa, Thiago R. da: 161
Rosa, Vagner: 91, 153, 157, 255
Roschild, Jônatas M.: 67
Salvador, Caroline Farias: 45
Sampaio, Felipe: 219, 223, 239
Santos, Glauco: 37
Santos, Ulisses Lyra dos: 105
Sawicki, Sandro: 15
Scartezzini, Gerson: 161
Schuch, Nivea: 51
Severo, Lucas Compassi: 41
Sias, U. S.: 101
Siedler, Gabriel S.: 95
Silva, André M. C. da: 63
Silva, Digeorgia N. da: 113
Silva, Ivan: 173
Silva, Mateus Grellert da: 95
Silva, Thaísa: 207, 231, 235, 247
Soares, Andre B.: 149
Soares, Leonardo B.: 153
Susin, Altamiro: 141, 149, 193, 207, 227, 235, 243, 247, 251, 255, 263
Tadeo, Gabriel: 91
Tavares, Reginaldo da Nóbrega: 19
Teixeira, Lucas: 83
Werle, Felipe Correa: 119
Wilke, Gustavo: 15
Wirth, Gilson I.: 105, 109, 193
Zatt, Bruno: 227, 251
Zeferino, Cesar Albenes: 45, 165, 169
Ziesemer Jr., Adriel Mota: 23
Zilke, Anderson da Silva: 71

