25th South Symposium on Microelectronics

May 10th to 15th, 2010
Porto Alegre – RS – Brazil

Proceedings

Edited by
Luciano Volcan Agostini
Cesar Albenes Zeferino
Fernando Gehm Moraes

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

Organized by
Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS)
Universidade Federal de Pelotas (UFPel)

Published by
Brazilian Computer Society (SBC)

Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Av. Ipiranga, 6681, Prédio 32
Porto Alegre - RS - Brasil

Dados Internacionais de Catalogação na Publicação (CIP)

C726p South Symposium on Microelectronics (25. : 2010 : Porto Alegre, RS)
Proceedings / 25. SIM ; ed. César Albenes Zeferino, Fernando Gehm Moraes, Luciano Volcan Agostini. – Porto Alegre: EDIPUCRS, 2010.
220 p.

1. Microeletrônica – Simpósios. 2. Engenharia Eletrônica. I. Zeferino, César Albenes. II. Moraes, Fernando Gehm. III. Agostini, Luciano Volcan. IV. Título.

CDD 621.3817

Ficha catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

ISSN 2177-5176

Printed in Porto Alegre, Brazil.

Cover: Leomar Soares da Rosa Junior (UFPel)
Cover Picture: Alexandre de Morais Amory (PUCRS)
Edition Production: Marcel Moscarelli Corrêa (UFPel)
Mateus Thurow Schoenknecht (UFPel)

Foreword

Welcome to the 25th 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 Porto Alegre together with the 12th edition of the regional Microelectronics School (EMICRO). A series of basic, advanced, hands-on and demonstrative short courses were provided by invited speakers.

These proceedings include 46 papers organized in eight sessions: Design Automation Tools 1, Video Coding 1, Design Automation Tools 2, Video Coding 2, NOCs and MPSoCs, Arithmetic and Digital Signal Processing, Devices and Analog Design and Digital Design and Embedded Systems. These papers came from 9 different institutions: PUCRS, UFRGS, UFPel, FURG, UFSM, UFSC, IF Sul-rio-grandense, UCPel and UNIPAMPA.

We would finally like to thank all individuals and organizations that helped to make this event possible. SIM 2010 was co-organized among PUCRS and UFPel, promoted by the Brazilian Computer Society (SBC), the Brazilian Microelectronics Society (SBMicro) and IEEE CAS Region 9, receiving financial support from CAPES and FAPERGS Brazilian agencies and DLP-CAS program. 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.

Porto Alegre, May 10, 2009

Fernando Ghem Moraes
Cesar Albenes Zeferino
Luciano Volcan Agostini

SIM 2010 - 25th South Symposium on Microelectronics

May 10th to 15th, 2010
Porto Alegre – RS – Brazil

General Chair

Prof. Fernando Gehm Moares (PUCRS)

SIM Program Chairs

Prof. Cesar Albenes Zeferino (UNIVALI)
Prof. Luciano Volcan Agostini (UFPel)

EMICRO Program Chair

Profa. Lisane Brisolara de Brisolara (UFPel)

IEEE CAS Liaison

Prof. Ricardo Augusto da Luz Reis (UFRGS)

Local Arrangements Committee

Alexandre Amory (PUCRS)
Edson Moreno (PUCRS)
Luciano Ost (PUCRS)
Guilherme Guindani (PUCRS)
Leonel Tedesco (PUCRS)
Marcel Corrêa (UFPel)
Mateus Schoenknecht (UFPel)

List of Reviewers

Adriel Ziesemer (UFRGS)
Alexandre Amory (PUCRS)
Antonio Carlos Beck Filho (UFSM)
Caio Alegretti (UFRGS)
Caroline Concatto (UFRGS)
Cesar Zeferino (UNIVALI)
Cláudio Diniz (UFRGS)
Cristiano Lazzari (INESC-ID)
Cristina Meinhardt (UFRGS)
Daniel Palomino (UFPel)
Denis Franco (FURG)
Digeorgia da Silva (UFRGS)
Douglas Rossi de Melo (UNIVALI)
Eduardo da Costa (UCPel)
Felipe S. Marques (UFRGS)
Felipe Sampaio (UFPel)
Fernando Moraes (PUCRS)
Gustavo Girão (UFRGS)
Gustavo Wilke (UFRGS)
Helen Franck (UFRGS)
José Rodrigo Azambuja (UFRGS)
Julio C.B. Mattos (UFPel)
Leomar Rosa Jr (UFPel)
Leonel Tedesco (PUCRS)
Lisane Brisolara (UFPel)
Lucas Brusamarello (UFRGS)
Luciano Agostini (UFPel)
Luciano Ost (PUCRS)
Marcelo Porto (UFPel)
Marcio Oyamada (UNIOESTE)
Marco Wehrmeister (UFRGS)
Marcos Hervé (UFRGS)
Mateus Beck Rutzig (UFRGS)
Maurício Lima Pilla (UFPel)
Monica Magalhães Pereira (UFRGS)
Osvaldo Martinello Jr (UFRGS)
Paulo F. Butzen (UFRGS)
Reginaldo Tavares (UNIPAMPA)
Renato Hentschke (INTEL)
Roger Porto (UFPel)
Sandro Sawicki (UNIJUI)
Thiago Assis (UFRGS)
Thiago Felski Pereira (UNIVALI)
Tomás Garcia Moreira (UFRGS)
Ulisses Brisolara Corrêa (UFRGS)
Vagner Rosa (FURG)
Vinicius Dal Bem (UFRGS)

Table of Contents

Session 4: VIDEO CODING 2

Session 5: NOCS AND MPSOCS

Session 6: ARITHMETIC AND DIGITAL SIGNAL PROCESSING

Design Automation Tools 1

KL-Cuts: Logic Synthesis Targeting Multiple Output Blocks

Osvaldo Martinello Jr, Felipe S. Marques, Renato P. Ribas, André I. Reis
{omjunior, felipem, rpribas, andreis}@inf.ufrgs.br

Institute of Informatics, Federal University of Rio Grande do Sul - UFRGS

Abstract

This paper introduces the concept of l-feasible backcuts, by extending the concept of k-feasible cuts to work with outputs. Then, the combination of k-feasible cuts and the new l-feasible backcuts originates the kl-cuts, which are circuit cuts with controlled number of inputs and outputs. An algorithm for computing this kind of cuts is presented and results are shown. These cuts introduce a new methodology for working with multiple output sub-circuits, and some applications to it are discussed.

1. Introduction

Some recent advances on logic synthesis are based on And Inverter Graphs – AIGs, for scalability reasons [1, 2]. Part of these advances is based on the concept of *k*-feasible cuts [3, 4], including algorithms for re-synthesis based on AIG rewriting [5]. Scalability is obtained by keeping the value of *k* small so that logic functions can be manipulated as vectors of integers. For instance, in [6] scalability is achieved by using functions of 16 or less inputs represented as binary truth-tables.

Algorithms for efficient cut computation are well known for single output cuts. Particularly, algorithms for exhaustive computation of *k*-feasible cuts were introduced by Cong [3] and Pan [4]. Chatterjee [7] introduced the concept of factor cuts, where exhaustive enumeration is avoided by making a separation between dag nodes and tree nodes in the AIG. All these algorithms for cut enumeration are only able to take the number *k* of inputs into account, not contemplating the benefits of multiple output reasoning. For example, in technology mapping using *k*-feasible cuts, logic duplication may occur during the step of covering, which is likely a problem on a design flow.

In this paper, we introduce the idea of controlling the number of outputs *l* in *k*-feasible cuts. Applications of *kl*-cuts may include peephole optimization [8], regularity extraction [9] and technology mapping. The use of *kl*-feasible cuts in peephole optimization is justified as an arbitrary portion of the circuit, potentially having multiple outputs, can be exchanged by another one by taking into account all signals which it affects at once. Its use in regularity extraction can be justified as many regular (logic) patterns are composed of multiple output circuits. This is especially true for arithmetic circuits, e.g. full adder and half adder library cells.

The remainder of this paper is organized as follows. Section 2 presents a state-of-the-art review. Section 3 discusses the concept of backcuts and *l*-feasible backcuts. Section 4 shows the *kl*-cuts definition, and an algorithm to enumerate *kl*-cuts on an AIG. Section 5 presents the results of the implemented algorithm. Section 6 discusses some possible applications for the proposed *kl*-cuts, and Section 7 concludes the paper.

2. Background

2.1. AIG

And-Inverter-Graph (AIG), $\mathcal{G}$, is a specific type of Directed Acyclic Graph (DAG), where each node has either 0 incoming edges – *primary inputs* (PI) – or 2 incoming edges – AND nodes. Each edge can be complemented or not. Some nodes are marked as *primary outputs* (PO).

2.2. K-Feasible Cuts

A *cut* of a node *n* is a set of nodes *c* such that every path between a PI and *n* contains a node in *c*. A cut of *n* is *irredundant* if no subset of it is a cut. A *k*-feasible cut is an irredundant cut containing *k* or fewer nodes.

Let A and B to be two sets of cuts. Let the auxiliary operation $\bowtie$ to be:

$$A \bowtie B \equiv \{a \cup b | a \in A, b \in B, |a \cup b| < k\}$$

Let $\Phi_{\mathcal{K}}(n)$ to be the set of *k*-feasible cuts of $n \in \mathcal{G}$, and if *n* is an AND node, let n_1 and n_2 to be its inputs. Then, $\Phi_{\mathcal{K}}(n)$ is defined recursively as follows:

$$\Phi_{\mathcal{K}}(n) \equiv \begin{cases} \{\{n\}\} & : n \text{ is a PI} \\ \{\{n\}\} \cup \left(\Phi_{\mathcal{K}}(n_1) \bowtie \Phi_{\mathcal{K}}(n_2)\right) & : otherwise \end{cases}$$

For instance, in Fig. 1, $\Phi_{\mathcal{K}}(r) = \{\{r\}, \{a, p\}, \{a, b, c\}\}$.

The $\bowtie$ operation can also easily remove the redundant cuts, by comparing the cuts with one another, or by making use of signatures [10].

3. L-Feasible Backcuts

The algorithms for computing cuts work from inputs to outputs. Computing *kl*-feasible cuts involves computing backward cuts – or *backcuts* – from outputs to inputs. The proposed backcuts are quite similar to cuts. However, instead of representing a set of nodes that can generate *n*, they represent a set of nodes that are influenced by *n*.

A backcut of a node *n* is a set of nodes *c* such that every path between *n* and a PO contains a node in *c*. A backcut is irredundant if no subset of it is a backcut. An *l*-feasible backcut is an irredundant backcut containing *l* or lesser nodes.

For convenience, let us define another operator, the operation $\bowtie_{i=m}^{n} x_i \equiv x_m \bowtie \cdots \bowtie x_n$. This attribution can be made since the $\bowtie$ operation is commutative.

Let $\Phi_{\mathcal{L}}(n)$ to be the set of *l*-feasible backcuts of *n*, and let n_i to be the *i*-th node connected to its output. We define $\Phi_{\mathcal{L}}(n)$ as:

$$\Phi_{\mathcal{L}}(n) \equiv \begin{cases} \{\{n\}\} & : n \text{ is a PO} \\ \{\{n\}\} \cup \left(\bowtie_{i} \Phi_{\mathcal{L}}(n_i)\right) & : otherwise \end{cases}$$

As an example, in Fig. 1, $\Phi_{\mathcal{L}}(p) = \{\{p\}, \{r, s\}, \{s, t\}\}$.

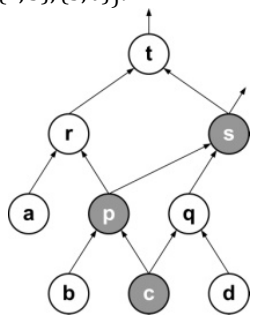


Fig. 1 – AIG demonstrating backcut enumeration.. Nodes *a*, *b*, *c* and *d* are primary inputs. Nodes *s* and *t* are primary outputs.

4. KL-Feasible Cuts

Cuts are an efficient way of representing a region of an AIG regarding one signal generation. However, when it comes to multiple output regions multiple cuts would be needed.

A *kl-cut* defines a sub-graph $\mathcal{G}_{kl}$ of $\mathcal{G}$ which has no more than *k* inputs and no more than *l* outputs. It is represented as two sets of nodes $\{\mathcal{G}_k, \mathcal{G}_l\}$: being $\mathcal{G}_k$ the inputs set and $\mathcal{G}_l$ the outputs set. If a node *n* belongs to a path between $n_k \in \mathcal{G}_k$ and $n_l \in \mathcal{G}_l$, and $n \notin \mathcal{G}_k$, then *n* is contained in $\mathcal{G}_{kl}$. Notice that all nodes in $\mathcal{G}_l$ are contained in $\mathcal{G}_{kl}$. However, $\mathcal{G}_{kl}$ does not contain any node of $\mathcal{G}_k$.

A *kl*-cut is said to be complete when all the following conditions are met:

- c1: Every path between a PI and a node $n_l \in \mathcal{G}_l$ contains a node in $\mathcal{G}_k$;
- c2: Every path between a node contained in $\mathcal{G}_{kl}$ and a PO contains a node in $\mathcal{G}_l$;
- c3: No *kl*-cut defined by a subset of $\mathcal{G}_k$ and the same $\mathcal{G}_l$ is complete;
- c4: No *kl*-cut defined by the same $\mathcal{G}_k$ and a subset of $\mathcal{G}_l$ is complete.

4.1. KL-Cuts Generation Algorithm

The objective of this algorithm is to find *kl*-cuts that have shared nodes on the generation of more than one output, that is, nodes that belong to *k*-feasible cuts of more than one output.

```
1. compute_klcuts(k, l, aig) {
2.   kcuts = compute_kcuts(aig, k)
3.   lcuts = compute_lcuts(aig, l)
4.   for (each lcut in lcuts) {
5.     p = combine_kcuts(lcut)
6.     for (each pi in p) {
7.       klcut = create_klcut(pi, lcut)
8.       if (check_and_fix(klcut))
9.         klcuts.add(klcut)
10.    }
11.  }
12.  return klcut
13.}
```

Fig. 2 – Pseudo-code for *kl*-cuts calculation.

Fig. 2 shows a pseudo-code for *kl*-cuts enumeration. It starts enumerating all *k*-feasible cuts and all *l*-feasible backcuts on the circuit. Each computed backcut generates a set of *kl*-cuts. The function on line 5 combines the *k*-feasible cuts of the nodes belonging to the current backcut *d*. Let d_i to be a node of *d*. Let *p* to

be $p = \bowtie_i \Phi_{\mathcal{K}}(d_i)$. This way, p is a set of input groups p_i, and each one defines a *kl*-cut $\{p_i, d\}$. Nevertheless, not every resulting *kl*-cut is complete, because condition c2 is not assured. So, the function on line 8 adds nodes to the outputs set in order to make the *kl*-cut complete, or else discards the *kl*-cut. If a node n_{kl} belonging to $\mathcal{G}_{kl}$ has as output a node that does not belong to $\mathcal{G}_{kl}$, then n_{kl} must be added to $\mathcal{G}_l$. If $\mathcal{G}_l$ still have no more than l nodes, $\mathcal{G}_{kl}$ is a complete *kl*-cut. On this implementation the *check_and_fix()* function also discards *kl*-cuts that are not connected.

For instance, in Fig. 3, starting by the backcut $\{u, v\}$ generated by the node *s*, the cuts $\{a, b, s\}$ from *u* and $\{s, g, h\}$ from *v* are combined, generating the incomplete *kl*-cut $\{\{a, b, s, g, h\}, \{u, v\}\}$. The last step adds the node *r* to the outputs set, resulting on the complete *kl*-cut $\{\{a, b, s, g, h\}, \{r, u, v\}\}$ containing the nodes *u*, *v*, *r* and *t*.

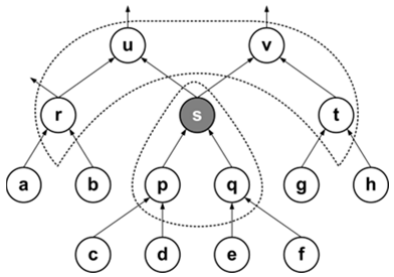


Fig. 3 – AIG illustrating a possible covering using 5-2-cuts. Nodes *a*, *b*, *c*, *d*, *e*, *f*, *g* and *h* are primary inputs. Nodes *u* and *v* are primary outputs.

5. Results

The algorithm was implemented in Java language and all results were obtained by execution on a 2.4GHz Intel Pentium IV with 1GB of RAM. The implemented program reads BAF files as inputs, which were generated from EQN files using the ABC tool [11], after running the *'dc2 –l'* command twice.

Table I shows results for *l*-feasible backcuts enumeration. Notice that for the circuit C3540 the time taken to compute the backcuts was particularly larger than the others. This is due to the fact that some nodes have a huge number of backcuts. The utilization of a limit factor of backcuts per node would significantly reduce this time in despite of the quality of results. Table II shows the results for constructing *kl*-cuts from cuts and backcuts enumeration.

Although a huge number of cuts can be enumerated from circuits, and therefore a big time to compute them, the applications can calculate *kl*-cuts on-the-fly, and more than that, compute only a small subset of them, which provides scalability to the applications.

Tab. 1 - Comparison between all l-feasible backcuts enumeration and factor backcuts enumeration.

Names	Nodes		l = 2		l = 4	
	Total	*% Dag*	*Total*	*Time (s)*	*Total*	*Time (s)*
C1355	433	38.11	1707	0.032	4457	0.344
C1908	393	41.98	1805	0.015	7350	0.421
C2670	777	22.52	4535	0.078	26286	1.578
C3540	975	27.18	6986	0.078	64779	334.703
C5315	1522	26.87	9960	0.109	29807	4.047
C6288	1902	73.87	7682	0.046	140157	3.875
C7552	1625	38.34	16790	0.188	71183	5.391
s13207	2757	27.64	10699	0.078	20605	22.422
s15850	3374	29.02	13391	0.125	54001	4.578
s35932	10841	30.84	90148	1.250	122938	9.547
s38417	9795	25.88	41122	0.250	255430	15.485
s38584	11306	27.56	39848	0.312	135238	10.594
Avg.	3808	34.15	20389	0.213	77686	34.415

Tab. 2 - Comparison between full kl-cuts search and global kl-cuts search.

Name	4-2-cuts			6-4-cuts		
	Total	*Size*	*Time (s)*	*Total*	*Size*	*Time (s)*
C1355	2778	4.19	0.578	36186	8.76	20.969
C1908	2300	3.59	0.422	24838	7.38	20.266
C2670	3313	3.18	0.797	21847	6.35	35.938
C3540	5392	2.61	1.063	56782	5.47	413.547
C5315	7680	3.00	2.297	59529	6.39	63.422
C6288	10836	3.21	2.797	345069	9.12	1083.828
C7552	9624	3.31	3.296	111160	7.10	224.500
s13207	8862	2.41	1.438	58607	5.11	38.266
s15850	13928	2.62	2.563	113296	5.48	70.219
s35932	54015	3.60	44.907	391684	6.75	817.985
s38417	44322	2.61	6.062	351974	5.38	345.703
s38584	44425	2.40	20.515	369003	5.22	294.844
Avg.	17290	3.06	7.228	161665	6.54	285.791

6. Applications

The step of technology mapping could make good use of *kl*-cuts, since most of standard cell libraries have multiple output cells, e.g. full adders. Similarly, library free technology mappers could use multiple output cells to significantly reduce area of a circuit. Moreover, current FPGAs have multiple output LUTs [12, 13].

Regularity extraction is important for improving the yield of fabrication [9]. Maintaining a hash table of *kl*-cuts structure and/or functions could help on an implementation of an algorithm to perform regularity extraction on a circuit.

Another application for *kl*-cuts is to perform peephole optimizations. By defining a sub-graph on an AIG, one can replace it by any other sub-graph that implements the same outputs, but minimizing a given cost function.

7. Conclusions and Future Work

The concept of *kl*-feasible cuts was introduced, which allows controlling both the number *k* of inputs and the number *l* of outputs in the computation of circuit cuts. Algorithms for computing *kl*-feasible cuts were presented and results have shown the usefulness of the method. Further work is necessary in the use of *kl*-feasible cuts in peephole optimization, AIG rewriting, regularity extraction, and technology mapping.

8. Acknowledgements

Research partially funded by Nangate Inc. under a Nangate/UFRGS research agreement, by CNPq Brazilian funding agency, and by the European Community's Seventh Framework Programme under grant 248538-Synaptic.

9. References

[1] Ling, A. C., Zhu, J., "Scalable Synthesis and Clustering Techniques Using Decision Diagrams", IEEE Trans. on CAD. 2008.

[2] Mishchenko, A., Brayton, R., "Scalable Logic Synthesis using a Simple Circuit Structure", Int'l Workshop on Logic & Synthesis, 2006.

[3] Cong J., Wu, C., Ding, Y., "Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution", Int'l Symp. on FPGA, 1999.

[4] Pan, P., Lin C., "A New Retiming-based Technology Mapping Algorithm for LUT-based FPGAs", Int'l Symp. on FPGA, 1998.

[5] Mishchenko, A., Chatterjee, S., Brayton, R., "DAG-Aware AIG Rewriting: A Fresh Look at Combinational Logic Synthesis", Design Automation Conference, 2006.

[6] Mishchenko, A., Brayton, R., Chatterjee, S., "Boolean Factoring and Decomposition of Logic Networks", Int'l Conf. on CAD, 2008.

[7] Chatterjee, S., Mishchenko, A. and Brayton, R., "Factor Cuts", Int'l Conf. on CAD, 2006.

[8] Werber, J., Rautenbach, D., Szegedy, C., "Timing Optimization by Restructuring Long Combinatorial Paths", Int'l Conf. on CAD, 2007.

[9] Rosiello, A. P. E., Ferrandi, F., Pandini, D., Sciuto, D., "A Hash-based Approach for Functional Regularity Extraction During Logic Synthesis", IEEE Comp. Soc. Annual Symp. on VLSI, 2007.

[10] Mishchenko, A., Chatterjee, S., Brayton, R., "Improvements to Technology Mapping for LUT-Based FPGAs", Int'l Symp. on FPGA, 2006.

[11] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification". http://www.eecs.berkeley.edu/~alanmi/abc

[12] Xilinx, "Achieving Higher System Performance with the Virtex-5 Family of FPGAs", White Paper, 2006. http://www.xilinx.com/

[13] Altera, "Improving FPGA Performance and Area Using an Adaptive Logic Module", White Paper, 2004. http://www.altera.com/

A Tool for the Automatic Generation of the Ordering and Partitioning of Coefficients in FIR Filters

[1]Angelo G. da Luz, [1]Eduardo A.C da Costa, [2]Marilton S. de Aguiar
angelogl@gmail.com, ecosta@ucpel.tche.br, marilton@ufpel.tche.br

[1]Catholic University of Pelotas – UCPel – Pelotas, Brazil
[2]Federal University of Pelotas – UFPel – Pelotas, Brazil

Abstract

This paper proposes a tool for the automatic generation of the ordering and partitioning of coefficients in Finite Impulse Response (FIR) filters, which are based on heuristic algorithms. Due to the characteristics of the FIR filter algorithms, which involve multiplications of input data with appropriate coefficients, the best ordering and partitioning of these operations can contribute for the reduction of the switching activity, what leads to the minimization of power consumption in the filters. Thus, two algorithms were implemented for the ordering and partitioning of the coefficients and all of them are based on some heuristic approach. The results are presented in terms of the Hamming distance between the consecutive coefficients.

1. Introduction

In this paper, we propose a tool that searches for the best ordering and partitioning of the sequential and semi-parallel FIR filters with the use of some heuristic-based algorithms. In the FIR filter operation, each tap involves getting the appropriate coefficient and data values and performing a multiply-accumulate computation. The larger the number of taps, the more accuracy we can have on the transfer function implemented by the filter. However, the hardwired implementation presents a large amount of arithmetic operation of addition and multiplication, which contributes for a higher power consumption in the FIR filter, mainly due to the multipliers.

In FIR filter computation, the summation operation is both commutative and associative, and the filter output is independent of the order of computing the coefficient product. Thus, coefficient ordering has been used as a technique for low power, where all coefficients are ordered in a Fully-Sequential circuit so as to minimize the transitions in the multiplier input and data bus [1]. In [2], this technique was extended-up by using the ordering and partitioning of the coefficients for the Semi-Parallel FIR filter architecture, where the hardware is duplicated and coefficients are partitioned into groups of coefficients. Thus, the problem in both cases is related to finding the best partition for each coefficient by calculating the minimum Hamming distance between the coefficients into each group. Although the techniques of [1] and [2] optimize the partitioning and ordering of the coefficients, the cost function of the algorithms is calculated for all the combinations over the coefficients, which is limited to a small group of coefficients, where the total number of permutations is calculated in a still reasonable time. However, for a higher number of coefficients these exhaustive algorithms are less attractive due to the time necessary to process the large number of combinations while using heuristics the results is obtained almost instantly.

In our work, we have implemented two heuristic-based algorithms named Nearest neighbor and Bellmore and Nemhauser [3,4], to get as near as possible to the optimal solution for the ordering and partitioning of larger filter instances. We have tested the use of the algorithms in a combination of different number of taps and different coefficient bit-width coefficients that were obtained by Remez algorithm in Matlab tool. A proposed tool calculates the Hamming distance between the coefficients automatically.

2. Heuristic-based algorithms

There are many heuristics in the literature, which can be classified as constructive [3,4] or refinement [3,4]. In this work only the constructive heuristics are taken into account. Constructive heuristics have as main feature the construction element by element solutions making local optimal choices in search for global optimum. Among the constructive heuristics in the literature we may include: Nearest Neighbor Heuristic, Bellmore and Nemhauser Heuristic and Cheapest Insertion Heuristic. This work uses the first two heuristics for the problem under study.

2.1. Nearest Neighbor Heuristic

This heuristic starts from an initial element and the next element choose to be added to the solution is that considered as the best neighbor (for a given cost function). Tab. 1 presents the symmetric cost to add each element to the sequence of decisions.

Tab. 1 Costs for the insertion of elements in the solution

Element	E1	E2	E3	E4	E5	E6
E1	0	2	1	4	9	1
E2	2	0	5	9	7	2
E3	1	5	0	3	8	6
E4	4	9	3	0	2	5
E5	9	7	8	2	0	2
E6	1	2	6	5	2	0

Applying the Nearest Neighbor Heuristic to the elements of tab. 1, we would find the following choice of items with the lowest cost ([E1] [E3] [E4] [E5] [E6] [E2]), with the following steps (considering E1 the initial element):

Step 1*: Insert E3 as the successor of E1, because it is the element with the lowest cost in the vicinity of E1. Thus, the current list consists of ([E1] [E3]).*

Step 2: *Insert the element E4, because E1 is less expensive but it has already been inserted. Current list is ([E1] [E3] [E4]).*

Step 3*: Insert E5, because it is the best available neighbor of E4. Current list: ([E1] [E3] [E4] [E5]).*

Step 4*: Insert E6, since it is the best available neighbor of E5. Current list: ([E1] [E3] [E4] [E5] [E6]).*

Step 5*: Insert E2, which is the only remaining element. Final list: ([E1] [E3] [E4] [E5] [E6] [E2]).*

The total cost of the elements of the list is given by:

Total Cost = cost (E1, E3) + cost (E3, E4) + cost (E4, E5) + cost (E5, E6) + cost (E6, E2) = 12.

2.2. Bellmore and Nemhauser Heuristic

This heuristic starts from an initial element and makes the insertion of its best neighbor. From there, the two ends of the list are analyzed in order to discover the best cost. Each new insertion should be confronted with both ends. The new element must be inserted at the end with lower cost.

Considering the same element values from tab. 1, we would have the following solution ([E2] [E6] [E1] [E3] [E4] [E5]), which would be formed by the following steps (starting from the initial element E1):

Step 1*: Insert E3, since it is the best neighbor of E1. Current list: ([E1] [E3]).*

Step 2*: Insert E6 because it is among the elements not yet added. Choosing the best neighbors to each end of the list, E6 appears as the best neighbor of E1, with a cost of 1. At the other end of the list, E4 is the best neighbor of E3 with a cost of 3. Then, E6 is added to the extreme left of the route. Current list: ([E6] [E1] [E3]).*

Step 3*: Insert E2 at the left end of the list. From the elements which are not added yet, the best neighbor of E6 is E2 with a cost of 2, and the element with lower cost from E3 is E4 with cost 3. Current list: ([E2] [E6] [E1] [E3]).*

Step 4*: Insert E4 in the far right of the list, because the best neighbor of E2 is E5 at a cost of 7, and the element with lower cost compared to E3 is E4 with a cost of 3. Current list: ([E2] [E6] [E1] [E3] [E4]).*

Step 5*: As the only remaining element is the E5, we just must see which end their inclusion has lower cost. If E5 is inserted at the end of E2, then the cost is equal to 7. On the other hand, if inserted as a neighbor of E4 then the cost is 2. Final list: ([E2] [E6] [E1] [E3] [E4] [E5]).*

The total cost of the elements of the list is given by:

TotalCost = cost(E2,E6) + cost(E6, E1) + cost(E1, E3) + cost(E3, E4) + cost(E4, E5) = 16.

2.3. Proposed Tool

In order to include the two heuristic-based algorithms and the coding schemes used in this work, and generates the ordering and partitioning of the coefficients automatically, we have implemented a tool. The tool receives as input the FIR filter coefficients (in decimal or binary representation) and they are ordered and/or partitioned according to the selected technique and according to the selected encoding scheme. The output of the tool is the Hamming distance of the coefficients based on the best cost function that is found by the heuristic-based algorithms. Fig 1 shows the interface of the proposed automatic tool.

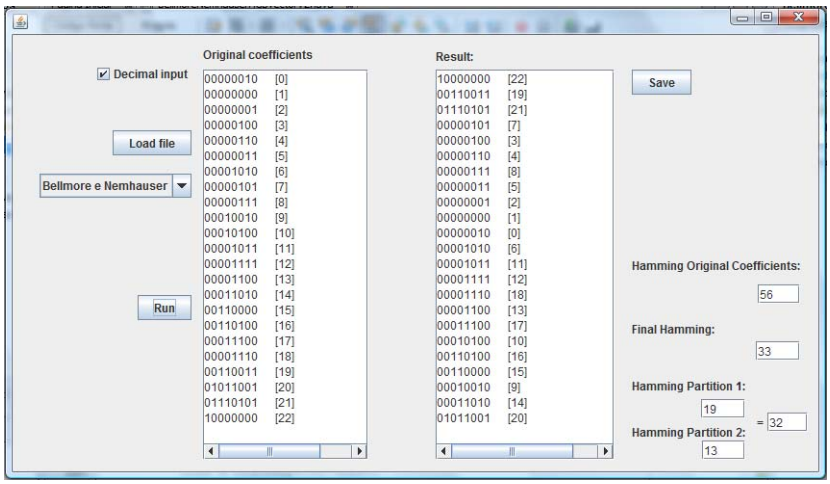


Fig. 1. Proposed tool interface

3. Results

In this section, we present the results obtained with the two heuristic-based commented previously. We have applied these algorithms to the reduction of the Hamming distance between consecutive coefficients, where the Hamming distance represents the different bits between two consecutive words. The filter coefficients were computed with the MATLAB using the Remez algorithm. We used the same FIR filter instances of [5]. The five columns of tab. 2 present the filters specification: *filter* is just an index for each example, *passband* and *stopband* are normalized frequencies, *#tap* is the number of coefficients of the filters and *width* is the bit-width of each coefficient. The results are obtained by the cost function that is automatically calculated by the proposed tool.

Tab. 2 Fir filters specification

Filter	Passband	stopband	#tap	width
1	0.10	0.25	27	10
2	0.15	0.25	18	12
3	0.10	0.15	25	14
4	0.15	0.25	29	14
5	0.10	0.15	48	16
6	0.15	0.20	30	14
7	0.15	0.20	49	16
8	0.10	0.12	49	16
9	0.10	0.12	60	18
10	0.24	0.25	53	12
11	0.20	0.25	23	8
12	0.20	0.25	36	12

Tab. 3 shows the results that were obtained, in terms of number of transitions, from the original coefficients and after using the heuristic-based Bellmore and Nemhauser and Nearest Neighbor algorithms for the ordered and partitioned coefficients. The partitioned coefficients were grouped into two columns of half of the coefficients.

Although the application of the heuristic-based algorithms reduces the number of the transitions of the coefficients significantly, the Bellmore and Nemhauser algorithm is more efficient for all the filter instances. This occurs because this algorithm looks for the best ordering of the coefficients in a larger search area. As also can be observed in tab. 3, the partitioning of the coefficients also reduces the number of transitions in all the filter instances, because the search for a best ordering occurs in a specified smaller group of coefficients. This is important for the implementation of a semi-parallel filter, because this it can operate faster (the processing of two samples at a time) with a reduced number of transitions between the coefficients (probably less power consumption).

Tab. 3 Number of transitions for the original coefficients

	Number of transitions between the coefficients (Hamming distance)		
		Bellmore and Nemhauser	Nearest Neighbor
Filter	Original Coefficients	Ordering/ Partitioning	Ordering/ Partitioning
1	85	53/52	56/53
2	67	44/42	47/45
3	102	71/69	74/71
4	139	79/76	81/78
5	249	149/146	157/155
6	150	89/87	93/92
7	276	147/144	152/151
8	294	177/174	182/179
9	398	231/228	242/239
10	204	95/93	98/96
11	56	33/32	34/33
12	137	83/81	87/86

4. Conclusions

In this work two heuristic-based algorithms named Nearest Neighbor and Bellmore and Nemhauser were applied to the ordering and partitioning of FIR filters coefficients. The results showed that the algorithms can find the best cost function for the ordering and partitioning of the coefficients, so that the Hamming distance between consecutive coefficients could be reduced significantly. An automatic tool was proposed in order to include the heuristic-based algorithms and generates the Hamming distance results from the ordered and partitioned coefficients. As future work we intend to implement the filters with original and ordered (for a sequential FIR filter) and partitioned (for a semi-parallel FIR filter) coefficients in order to observe the impact on power reduction of the filters with the reduction of the number of the transitions of the coefficients.

5. References

[1] M. Mehendale, S. Sherlekar, G. Venkatesh. Techniques for Low Power Realization of FIR Filters. In: Asia and South Pacific Design Automation Conference, ASPDAC, 1995. Proceedings… New York: ACM, 1995. p.447-450.

[2] E. Costa, J. Monteiro , and S. Bampi. Gray Encoded Arithmetic Operators Applied to FFT and FIR Dedicated Datapaths. In: 11th IEEE/IFIP International Conference on Very Large Scale Integration. VLSI-SoC. 2003.

[3] Ansari, Nirwan and Hou, E. Computational Intelligence for Optimization. Kluwer Academic Publishers, 1997.

[4] Reeves, C.R. Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientif Publications, 1993.

[5] E. Costa, P. Flores, and J. Monteiro. Maximal Sharing of Partial Terms in MCM under Minimal Signed Digit Representation. In: European Conference on Circuit Theory and Design. pp. 221-224. 2005.

AIG Rewriting Considering Multiple Objectives

Thiago Figueiro, Renato Ribas, Andre Reis
{trfigueiro,rpribas,andreis}@inf.ufrgs.br

Av. Bento Gonçalves, 9500, Bloco IV, Porto Alegre/RS, Brazil
Instituto de Informática – Universidade Federal do Rio Grande do Sul

Abstract

This work presents a technology independent logic synthesis algorithm that works on top of an AND-INV graph (AIG) data structure. The final goal is to minimize a cost function taking into account both the number of nodes and the depth of the AIG. In this paper we analyze several cases for AIG rewriting that affect the depth and number of nodes of final AIG. This initial study is proposed to guide an algorithm for constructive synthesis of AIGs considering a trade-off between area and depth.

1. Introduction

Algorithmic logic synthesis is usually performed in two steps, one performed over Boolean equations (regardless any physical property) and another where the resulting logic is mapped into a physical cell library or other physical implementation. The first step is composed of several logic operations such as Decomposition Extraction, Factoring, Substitution and Elimination [1]. These operations may be either explicitly performed (SIS [2]) or implicitly performed by other methods such as AIG rewriting (ABC [3]).

The main goal of technology independent synthesis is to compute an equation that represents a given Boolean function with a minimum number of literals. This type of optimization aims to yield a minimum area of the circuit implementing this function [4]. Some heuristic techniques have been proposed, achieving high commercial success, such as quick_factor (QF) and good_factor (GF), both algorithms available in SIS tool [2]. Recent works on this field indicate that it is not only possible to reduce area by a good technology independent synthesis but also it is feasible to enhance other characteristics of the resulting circuit by focusing on goals other than minimum number of literals. Recently, a function composition algorithm was proposed in order to build equations in a bottom-up approach, where characteristics other than literal count are taken into consideration [5].

The second step of algorithmic logic synthesis is to map the function description into a physical implementation. This is usually implemented as a graph covering problem. The most common approach is to use the resulting equations from technology independent synthesis to build a Boolean network, which is a directed acyclic graph (DAG), with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates inputs and outputs [2]. Recent works are using AND-INV graphs (AIGs) as a special case of a DAG to represent the Boolean network on the covering step [3]. AIGs are graphs whose nodes are limited to two-input ANDs while inverters are indicated by a special attribute on the edges of the network.

A Boolean function has many possible AIG representations, according to the order which the variables are processed and how the nodes of the AIG graph are created. However, it is possible to build AIGs ensuring that functional equivalent structures are represented by the same sub-graph. This idea leads to the integration of the functional reduction process to the AIG construction, generating what is called Functionally Reduced AND-INV Graph (FRAIG) [6].

A function may have different FRAIG representations, according to the characteristics of the equation used to build the FRAIG. It is known that two functions with the same number of literals may produce FRAIGs with different sizes and that equations with more literals may, in some cases, produce a smaller FRAIG. Therefore, this paper intend to explore a large set of equations trying to identify characteristics that allow equations with a larger number of literals to result in better FRAIGs (either regarding the number of nodes or the logical depth). This work is part of a project that intends to explore the different characteristics of equations in the context of AIG rewriting.

This paper is organized as follows. Section 2 presents basic concepts regarding technology independent synthesis algorithms and FRAIGs. Section 3 presents and discusses examples of equations and their corresponding FRAIGs. The final section discusses the conclusions of this paper.

2. Basic Concepts

A Boolean function describes how to determine a Boolean value output based on some logical calculation performed over Boolean inputs. An equation is one form to represent a function, which may also be described

as a Binary Decision Diagram (BDD) or as a Truth Table (TT), for instance. Every representation of a function may be classified as canonical or non-canonical. A representation is said to be canonical if every function will always be described exactly in the same way. Examples of canonical representations are BDDs and TTs (as long as the variable orderings are the same). Equations are non-canonical representations of a function. Therefore, the same function may be described by different equations. For instance, equations (1), (2) and (3) represent exactly the same function. An equation is composed of literals. A literal is an instance of a variable (positive literal, for instance "A") or its complement (negative literal, for instance "!A").

*F=A*B+C* *(1)*

*F=A*B+A*C* *(2)*

F=(A+C)(B+C)* *(3)*

AND-INV graphs (AIGs) are another way to represent Boolean functions. An AIG is a graph composed exclusively of two-input AND gates and inverters. It is frequent to represent the inverters as a special flag on graph edges. This way, all nodes on the graph represent two-input ANDs. Figure 1 present one AIG for the equation F=A*!B*C. Different AIGs may represent the same Boolean function, as AIGs are not a canonical representation. Figure 2 presents three different AIGs for the same function (they implement the equations presented previously).

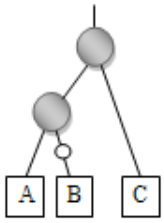
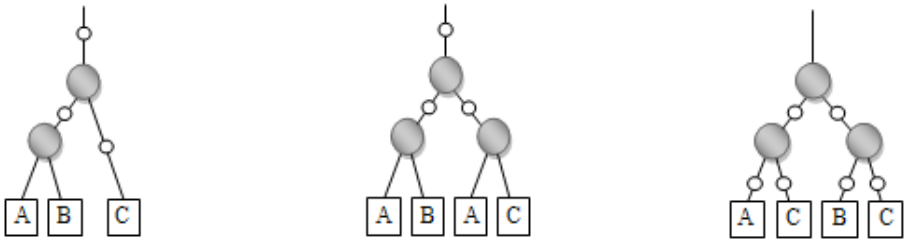


Fig. 1 - Sample of AIG for the equation F=A*!B*C



(a) ***(b)*** ***(c)***

Fig. 2 - Sample of AIG representing the same function but described as (a) F= A*B+C, (b) F= A*B+A*C and (c) F=(A+C)*(B+C)

3. Examples

This section presents two examples. The first example consists on the result of two different factoring algorithms and the demonstration that an equation with less literals may result in a larger AIG graph. The second example presents different equations generating graphs with different logical depths.

3.1. Example 1

This example intends to demonstrate that assuring the smallest equation possible for a function will not result in the smallest AIG graph. Consider the function represented by the truth table in figure 3. Considering two possible factoring results of this function represented in equations (4) and (5). Equation (4) is the factoring result generated by the algorithm described in [4] and equation (5) the result from ESPRESSO command in SIS [3].

Q=(!B(!D+!C))* (!A*(D+C))* *(4)*

Q=!A!B*C*!D+!A*!B*!C*D* *(5)*

A	B	C	D	Q
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Fig. 3 - Truth table for the function in example 1.

The resulting FRAIGs when using equations (4) and (5) to build them are presented in figures 4 and 5, respectively.

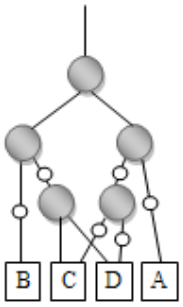


Fig. 4 - FRAIG built using equation (4) as input.

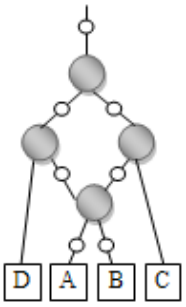


Fig. 5 - FRAIG built using equation (5) as input.

Although equation (4) present 6 literals, while equation (5) present 8 literals, the resulting FRAIG for equation (4) has one extra node, since it is more difficult to reuse intermediate nodes (notice that one node in the FRAIG generated for equation (5) presents a fanout equal to 2).

3.2. Example 2

This example intends to demonstrate that two different equations may generate graphs with different heights. This can potentially lead to circuits with different logical depths. Figure 6 presents the truth table for the function used in example 2.

A	B	C	D	Q
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Fig. 6 - Truth table for the function in example 2.

Considering two possible factoring results of this function represented in equations (6) and (7).

$Q=(a*b+c)+d$ (6)

$Q=(a*b)+(c+d)$ (7)

The resulting FRAIGs when using equations (6) and (7) to build them are presented in figures 7 and 8, respectively.

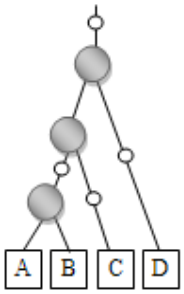


Fig. 7 - FRAIG built using equation (6) as input.

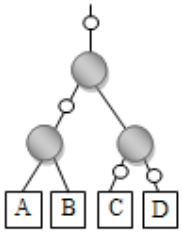


Fig. 8 - FRAIG built using equation (7) as input.

Although equation (6) presents the same number of literals than equation (7), the resulting FRAIGs present different heights since the associations are performed in a different order.

4. Conclusions

The use of AIGs for representing a network to be mapped is appropriate since its characteristics directly reflect the characteristics of the resulting circuit. There are several techniques to reduce the AIGs size, both available in building time (such as the FRAIG) and after construction enhancements (techniques for AIG rewriting). In this paper we highlight the fact that features of an AIG associated to the quality of the final implementation can vary for the same logic function. This initial study will be used to produce a constructive algorithm to produce AIGs while minimizing a cost function expressing a trade-off between the depth of the AIG and the final number of nodes.

5. Acknowledgements

Research partially funded by Nangate Inc under a Nangate/UFRGS research agreement, by CNPq Brazilian funding agency, and by the European Community's Seventh Framework Programme under grant 248538-Synaptic.

6. References

[1] G.D. Hachtel and F. Somenzi, Logic Synthesis and Verification Algorithms, Kluver Academic Publishers (1996).

[2] E. Sentovich, K. Singh, L.Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, SIS: A system for sequential circuit synthesis. Tech. Rep. UCB/ERL M92/41. UC Berkeley, Berkely, 1992.

[3] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[4] M.C. Golumbic and A. Mintz, Factoring Logic Functions Using Graph Partitioning, ICCAD '99. IEE Press, Piscataway, NJ, 195-199.

[5] A. Reis, A. Rasmussen, L. Rosa, R. Ribas., Fast Boolean Factoring with Multi-Objective Goals, International Workshop on Logic & Synthesis, IWLS 2009.

[6] A. Mishchenko, S. Chatterjee, R. Jiang, R. Brayton, "FRAIGs: A Unifying Representation for Logic Synthesis and Verification", ERL Technical Report, EECS Dept., UC Berkeley, March 2005.

Automatic Cell Layouts Generation Using the ASTRAN Tool

Gracieli Posser, Daniel Guimarães Jr., Adriel Ziesemer, Gustavo Wilke, Ricardo Reis
{gposser,dsgjunior,amziesemerj,wilke,reis}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul (UFRGS)
Instituto de Informática – PPGC/PGMicro
Av. Bento Gonçalves 9500 Porto Alegre, RS - Brazil

Abstract

In this work, a cell library in a 0.35 µm technology was automatically generated using the ASTRAN tool. These cells are then inserted in an automatic synthesis flow to generate some test circuits. Results are compared to results obtained using the same flow with a commercial standard cell library. In the first step cells are generated and then characterized so that they can be used in the standard synthesis flow. This process it was done using Cadence's Encounter Library Characterizer tool. Using the two liberty files was possible to compare a design mapped using commercial standard cell library and the automatic generated library. Our results in the synthesis flow indicate that the automatically generated cells saved up to 24% of power and the timing is on average 12% lower. The increase in the total circuit area was only 12%.

1. Introduction

New products are needed in the market in the shortest time possible due to time-to-market, and these devices should have more functionalities and higher performance. This increases the need for the development of CAD (Computer Aided Design) tools that enables productivity increasing the designer's ability of designing higher complexity projects in a shorter time.

To achieve an efficient area and performance tradeoff in current designs, a cell-based methodology with large standard cell libraries is usually employed. To address this problem, cells synthesis tools can be used to quickly generate physical layouts for a given transistor-level netlist, accordingly to the design rules and constrains specified by the designer. The ASTRAN [1] tool is an example of a cells synthesis tool.

Using a tool for automatic cell generation reduces the need to fabricate integrated circuit layouts for the full-custom methodology and enabled the circuits generation on the fly using a smaller number of transistor than standard cells methodology. On this paper ASTRAN was used to generate a large standard cell library. The synthesis results of two circuits, one synthesized using a standard cell library generated by ASTRAN and the other using a typical standard cell library, were compared.

The remaining of this paper is organized as follows: in section 2, we present the ASTRAN tool that is used to generate the layouts. Section 3 discusses the approach used for cells characterization and the tool ELC (Encounter Library Characterizer) used in this work to make the characterization. The cells generation and characterization to the 0.35 µm technology are explained in Section 4. In section 5, a comparison is made by mapping a set of automatically generated cells and a set of standard cells from a commercial library to circuits found in the set of benchmarks ISCAS/89. In this section, is also made a comparison between the two sets of cells characterization values. Finally, we present conclusions in section 6.

2. ASTRAN Tool

In this work, the academic netlist-to-layout tool ASTRAN [1] is used to synthesize the cells being designed. This tool was developed by Ziesemer [2] in order to automate the process of developing circuit layouts.

The ASTRAN generates the cells under a linear matrix (1D) layout style and is able to support unrestricted circuit structures, and continuous transistor sizing. Also, the generator supports transistor folding and uses Threshold Accept algorithm [3] to determine a placement that maximizes the diffusion sharing and minimizes the interconnection length. The cell nets are routed using a negotiation-based algorithm similar to Path-Finder [4]. An Integer Linear Programming (ILP) solver [5] is used for compaction. The compaction step produces the final layout, according to the result provided by the placement/routing steps and regarding the technology design rules [1].

The tool receives as input a file containing a SPICE netlist of the cells (with their respective and individually sized transistors and interconnections), a configuration file (which defines the layout topology and control parameters to the generator), and a technology file, which contains a description of the design rules [1].

For a given transistor network, the tool objective is to place and route the transistors using the proposed layout style in such a way that the cell width and interconnections length are minimized. At the end, the circuit is compacted to produce a design error-free layout in CIF (Caltech Intermediate Format) and LEF (Library Exchange Format) formats [1].

The automatically generated cells follow a pattern that allows creation of a cell library and subsequently they can be used to integrate a generation flow with automatic placement and routing, compatible to the standard cells flow [2].

3. Cells Characterization

Cell characterization generates a liberty file (.lib) as a result of the characterization process. The liberty file contains power, timing and noise values for each cell after its fabrication. The characterization is made using Cadence ELC (Encounter Library Characterizer) [6]. This tool, generate ECSM models (Effective Current Source Model) that increase the accuracy in the delay values, where fluctuations in voltage, process variation and noise are problematic.

The ELC requires three input files to perform the characterization of the cell library:

- Setup file: this is the cells configuration file, where the values in which the characterization will be based are reported. The values are: corners conditions (voltage, temperature), the standard cells signals (maximum output slew rate, threshold voltage), data for standard cells simulation (transient, resistance), slew rate and load capacitance for each cell or group of cell (for instance, if a set of cells have the same slew and load values, then they are in the same group and require only one specification), process characteristics (simulation corner).
- SPICE subcircuits file: contains the subcircuit of each cell to be characterized. The subcircuit is taken from the SPICE netlist obtained through the layout parasitic extraction using Cadence's Virtuoso.
- Transistor model: is also taken from the SPICE netlist, but, as all cells have the same technology, it is used only one template file. It has the electrical characteristics of the transistor.

In addition to these three files, scripts are required to automate the cell characterization process. ELC output is the liberty file (.lib).

The parameters used to characterize the library generated by ASTRAN have to be in accordance to the parameters used to characterize the commercial standard cell library to which synthesis results are going to be compared. Input slew range, loading capacitance range, timing characteristic have all to be matched to guarantee a fair comparison between the two libraries.

4. Methodology

This work follows the flow shown in Figure 1. First, we select 14 basic cells used for the comparison between automatic generated cells and standard cells. The selected cells are: ADD21, BUF2, CLKBU2, DF1, INV1, JK1, NAND20, NOR20, DFC1, XOR20, INV6, MUX21, AOI211 and OAI210. We extract the SPICE netlist of each cell by using Cadence Virtuoso tool.

This netlist is used as input to ASTRAN to automatically generate cell layouts. The CIF format file is then imported into Virtuoso where we set the pins, verify the design rules and extract the layout parasitics to a SPICE file. Next, cell characterization is done using the Encounter Library Characterize tool from Cadence and the output is the Liberty file (.lib).

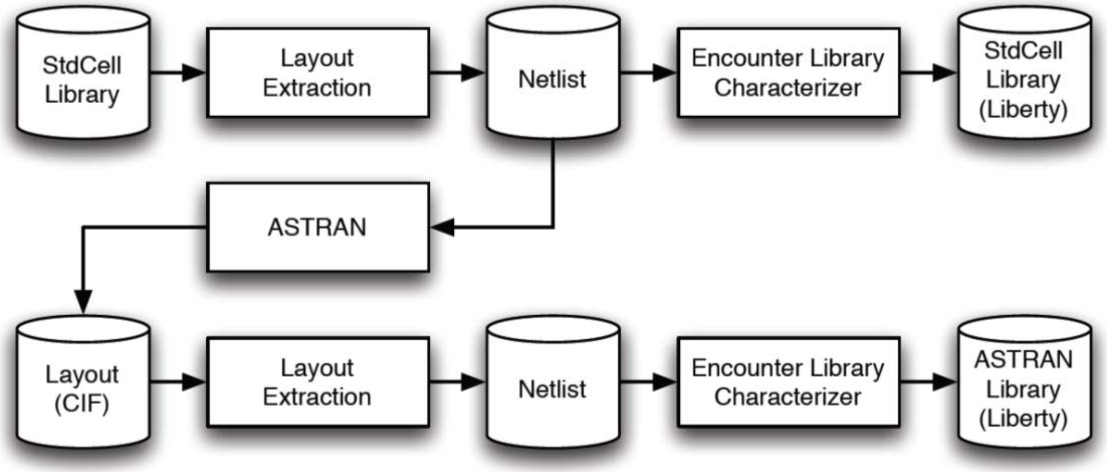


Fig. 1 – Design Flow to generate the set of cells

Following the flow shown in Figure 1, it was possible to obtain a file with a set of cells to be instantiated by a logic synthesis tool. To obtain a fair comparison between this set of automatically generated cells and standard cells, we had re-characterized the standard cells. This step is necessary, because the two set of cell should have the same characterization data and a same setup file. In this paper, we used the characterization process typical, where the corner conditions are: 25^0 C temperature and voltage of 3.3V.

5. Comparison between Standard Cells and Automatic Generated Cells

The comparison was made using eight ISCAS/89 [7] benchmark circuits: s1196, s1238, s15850, s9234, s35932, s38417, s38584 and s13207. Table 1 presents the power (mW), timing (ns) and area (μm^2) values for the benchmarks mapping using two different standard cell libraries, the commercial standard cell re-characterized library (SC) and automatic generated one (AG). Negative values indicate that the commercial standard cell library yields a better result. The mapping was obtained by running the logic synthesis tool, RTL Compiler. The different technology mapping was used for mapping the circuits using both libraries.

Table 1 – Comparison between standard cells (SC) and automatic generated cells (AG)

	Power (mW)			**Timing (ns)**			**Cell area (μm^2)**		
	SC	**AG**	**Gain(%)**	**SC**	**AG**	**Gain(%)**	**SC**	**AG**	**Gain(%)**
s1196	29.40	23.34	25.96	2.144	2.146	-0.09	38929	40385	-3.61
s1238	29.23	23.55	24.12	2.310	2.012	14.81	37765	44226	-14.61
s15850	190.22	153.47	23.95	2.779	2.368	17.36	62517	72982	-14.34
s9234	208.94	167.75	24.55	3.538	2.971	19.08	91655	107543	-14.77
s35932	2527.74	2040.05	23.91	2.520	2.328	8.25	957627	1097041	-12.71
s38417	2322.22	1871.97	24.05	4.432	4.069	8.92	950604	1074728	-11.55
s38584	1707.61	1376.91	24.02	2.976	2.68	11.04	738101	853434	-13.51
s13207	466.39	373.23	24.96	2.434	2.16	12.69	161761	184821	-12.48
Avg.	**935.22**	**753.78**	**24.44**	**2.890**	**2.59**	**11.51**	**379870**	**434395**	**-12.20**

According to Table 1, the automatically generated cells saved up to 25% of power and the design timing is 12% average faster than the circuits using standard cells. The standard cells achieve a better result only in area, obtaining a 12% lower, because the automatic generated cells layout is less dense, therefore having larger cell areas.

In addition to the results obtained by logic synthesis, we compared the characterization values obtained for the two sets of cells. The results are showed in Table 2 in percentage. Positive values mean that automatic generated cells had a lower value, in this case, a better result. Two aspects have significant difference: automatic generated cells have produced an average 13.44% smaller input capacitance and commercial standard cells presented 16.4% smaller area.

Table 2 - Comparison cell to cell from Liberty values

Cells	**Timing (ns)**	**Power**	**Leakage**	**Transition**	**Area**	**Input Cap.**
	Gain (%)	Gain (%)	Gain (%)	Gain (%)	Gain(%)	Gain(%)
ADD21	2.86	5.88	0.02	0.93	-20	18.07
AOI211	1.59	7.08	0.43	1.34	-20	12.95
BUF2	0.35	1.11	-0.02	0.26	-25	-1.98
CLKBU2	2.48	3.27	-0.04	-2.23	-25	12.72
DFC1	4.22	1.01	-0.16	0.61	-10.52	15.11
DF1	9.76	-1.33	0.10	1.20	-11.76	54.85
INV6	-2.33	2.60	-42.68	-2.31	-40	-6.57
INV1	2.05	8.93	-0.42	1.10	0	42.39
JK1	1.94	-6.59	-0.04	0.54	-20.83	44.59
MUX21	1.60	3.54	-0.03	0.50	-14.28	19.34
NAND20	1.27	6.06	0.04	1.63	0	8.32
NOR20	-0.52	2.47	-0.19	0.77	-25	9.68
OAI210	-0.70	-1.26	0.29	-0.16	-20	15.67
XOR20	-1.85	-3.81	-0.19	-0.80	0	25.68
Average	**2.26**	**0.30**	**-2.20**	**0.23**	**-16.37**	**13.44**

The timing and power gain for the automatic generated cells presented in Table 1 is much higher than the gain compared vis-to-vis in Table 2. This is explained by the smaller input capacitance presented by the automatic generated cells. The smaller input capacitance has allowed the logic synthesis tool to reduce the total number of gates in the circuit and the number of gates in the longest path, improving timing and power while keeping transition times under control.

Table 2 shows that larger cell areas represent smaller input capacitances. This happen because the automatic generation tool has designed cells where the polysilicon used to create the transistor gates presents a smaller coupling capacitance with metal 1 lines used to route internal signal. The commercial standard cell library was designed minimizing area, reducing the available space for internal interconnects, consequently there are more metal lines crossing over the polysilicon lines increasing the gate input capacitance.

The main advantage of automatically generated cell is to produce cell versions different from the existing in standard libraries, this comparison doesn't fully take advantage of the benefits of the automatic layout

generation approach, better timing and power results could be achieved by using optimized netlist produced by logic synthesis and transistor sizing tools.

6. Conclusion

In this work, we have shown that automatic generated cells are able to present the same overall quality than standard cells, or even better. By using a set of cells automatically generated we could verify the efficiency of the ASTRAN tool, reducing the time needed for a layout, and generating layouts with smaller input capacitance. This characteristic results in a design with a smaller power consumption and timing than the commercial standard cells. This work enabled us to learn about cells characterization and use the tool Encounter Library Characterizer. Furthermore, this work will be available for other colleagues to build circuits, or parts there, using cells layout which may reduce design timing and power consumption.

As future work, we intend to generate more cells to be included in the set generated in this work. These cells will have different logic and sizes than the ones available in the commercial standard cell libraries. This will allow a higher quality project. The final goal is to design a synthesis flow, where the first step is to use a logic synthesis tool that makes logical optimizations to reduce the number of transistors; an example is the ELIS (Environment for Logic Synthesis) tool [8]. The output of the logic synthesis tool is then inserted into the ASTRAN allowing the circuit layout generation on the fly.

7. Acknowledgment

This work is partially supported by Brazilian National Council for Scientific and Technological Development (CNPq – Brazil) and Coordination for the Improvement of Higher Education Personnel (CAPES).

8. References

[1] A. Ziesemer; C. Lazzari, R. Reis. “Transistor Level Automatic Layout Generator for non-Complementary CMOS Cells,” In: IFIP/CEDA VLSI-SoC2007, International Conference on Very Large Scale Integration, Atlanta, USA, October 15-17, 2007. pp. 116-121.

[2] A. M. Ziesemer Jr. “Geração Automática de Partes Operativas de Circuitos VLSI”. master`s thesis, PPGC, UFRGS, Porto Alegre, 2007.

[3] G. Dueck et al., “Threshold accepting: A general purpose optimization algorithm appear superior to simulated annealing”, Journal of Computational Physics, 1990.

[4] L. MCMURCHIE; C. EBELING. “PathFinder: a negotiation-based performance-driven router for FPGAs,” In: ACM International Symposium on Field-Programmable Gate Arrays, FPGA, 3., 1995, Monterey, California, United States, Proceedings… New York: ACM Press, 1995. p.111-117.

[5] Lpsolve, “Mixed Integer Linear Programming (MILP) Solver”, http://lpsolve.sourceforge.net/5.5/

[6] ENCOUNTER LIBRARY CHARACTERIZER – ELC. Cadence. Available at: http://www.cadence.com/products/di/library_characterizer/pages/default.aspx.

[7] ISCAS89, “Iscas89 benchmark circuits,” March 2009. [Online]. Available: http://courses.ece.illinois.edu/ece543/iscas89.html

[8] ELIS (Environment for Logic Synthesis). Available at: http://www.inf.ufrgs.br/nangate.

SwitchCraft - A Tool for Generating Switch Networks for Digital Cells

[1]Vinicius Callegaro, [1]Felipe de Souza Marques, [1]Carlos Eduardo Klock,
[2]Leomar S. da Rosa Jr, [1]Renato P. Ribas, [1]André I. Reis
{vcallegaro, felipem, ceklock, rpribas, andreis}@inf.ufrgs.br, leomarjr@ufpel.edu.br

[1]Instituto de Informática – Nangate/UFRGS Research Lab, Porto Alegre, Brazil
[2]Departamento de Informática – IFM – UFPel, Pelotas, Brazil

Abstract

This paper presents a new CAD tool for switch network synthesis. SwitchCraft environment provides a set of tools for switch network (and logic gate) generation. Transistor networks corresponding to target logic functions can be generated from Boolean equations and/or from BDD graphs. Logically and topologically complementary networks can be derived through dual-graphs. Different CMOS logic styles can be obtained, e.g. single- and dual-rail, static and dynamic topologies, with disjoint planes or in a non-disjoint PTL-like structure (with shared pull-up/pull-down). Estimators for delay propagation, layout area and power dissipation (dynamic and leakage) are available. The switch network profile can also be extracted, providing the logic function behavior, switch/transistor count, number of connections in internal nodes, the longest and shortest paths, and other characteristics.

1. Introduction

Currently, VLSI design has established a dominant role in the electronics industry. CAD tools have enabled designers to manipulate a huge number of transistors on a single die and shorten time-to-market. In particular, logic synthesis tools have contributed significantly to reduce the design cycle time. In this sense, it becomes also comfortable to have a transistor network synthesis tool to help digital circuit designers to generate and evaluate potential logic gate implementations that respects the project needs.

There are several algorithms to derive transistor networks. Some algorithms have as main goal to reduce the total number of transistors in the network [1]. Other ones reach transistor arrangements that respect the minimum theoretical transistor stack for a target Boolean function [2]. There is a large variety of transistor network arrangements that represent the same logical functionality, presenting naturally different behaviors in terms of area, delay and power consumption. In this way, it became necessary to provide a tool that is able to generate and to compare different transistor network arrangements. A tool that could be used to perform these tasks is not available neither in the industry, nor in the academia until now.

This paper introduces a new CAD tool called SwitchCraft, which presents some features allowing the user to generate transistor networks using several different algorithms, and to evaluate electrical and physical characteristics through estimations. Also, it is possible to generate logic gates in different logic styles, like standard CMOS [3], pass-transistor logic (PTL) [3] and others. It is a fast and easy way to investigate the behavior of possible transistor networks and logic gate implementations for a given target logic function.

The remaining of this paper is organized as follows. Section 2 introduces basic concepts. Section 3 describes the proposed framework and conclusions are outlined in Section 4.

2. Background

There are several different methods for implementing switch networks proposed in the literature. The resulting networks may present different properties, including physical and electrical characteristics. Different logic styles are also available to implement digital circuits. In this sense, the first question is: what is the best transistor arrangement that should be used to implement a target logic gate or digital circuit? Several different methods and algorithms suggest to deliver the best solution for specific cases, and they must be exploited to ensure the best solution. In the same way, a second question becomes: considering different logic styles, what is the best one to achieve the design needs and constraints? Again, this answer is not obvious, since it depends of several aspects regarding the project. These aspects include the definition of the input function that needs to be implemented and the process technology targeted. To help designers to evaluate alternative solutions, several estimation methods are described by the scientific community. The main idea behind these methods is to offer means to analyze transistor network and logic gate behaviors without the need to perform simulation and characterization. The use of estimators accelerates the design flow and increases the exploitation of the solutions space.

2.1. Switch Networks

Switch networks are composed of an arrangement of switches that implement a more complex switching function. The switches, as elements, may be either a direct switch (when it conducts by applying a logic '1' value in the control terminal) or a complementary switch (when it conducts by applying a logic '0' value in the control terminal). By interconnecting several switches, it is possible to build different arrangements, for switch networks, such that the resulting network performs the interconnection between two different terminals according to a given logic function. Fig. 1 illustrates some switch arrangements.

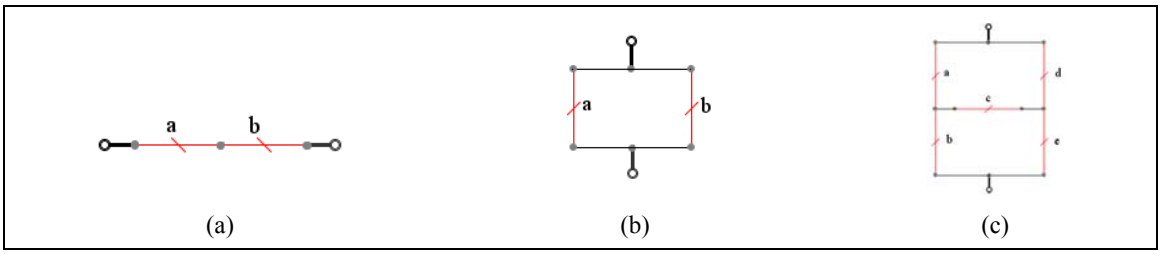


Figure 1 – Switch arrangements: (a) in series (b) in parallel (c) non-series-parallel.

Depending on the technology used, these switches can be implemented as different physical devices. In the current CMOS technology, NMOS transistors are direct switches, while PMOS transistors are complementary switches.

2.2. CMOS Logic Styles

Logic styles are basically classified as being static and dynamic topologies. Static styles guarantee that, under fixed input vectors, each gate output is connected to power rail supplies (either 'Vdd' or 'Gnd') via a low resistance path. Logic gate outputs assume at all times the value of the Boolean function implemented by the circuit, meaning that the circuit does not need to be pre-charged or pre-discharged. Some of the most common static logic styles are static CMOS, pseudo-NMOS, DCVSL and PTL [3]. Dynamic styles rely on temporary storage of signal values on the capacitance of high-impedance circuit nodes [4]. The implementation approach of dynamic circuits may result in faster circuits, but their design and operation are more prone to failure because of the increased sensitivity to noise. The most common dynamic logic styles are Domino and its variants dual domino, multiple-output domino, NORA domino and zipper domino [5]. Some logic styles are depicted in Fig. 2. The boxes labeled N and P represent NMOS and PMOS switch networks, respectively.

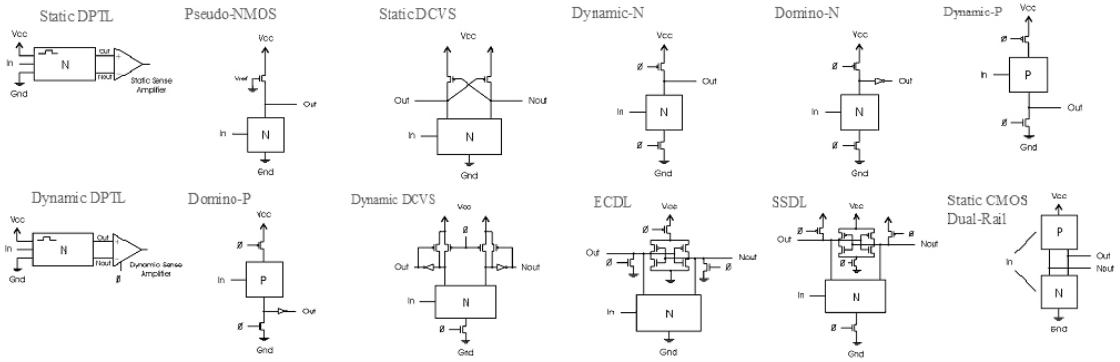


Figure 2 – Some logic styles.

3. SwitchCraft Framework

The main goal of SwitchCraft framework is to provide a set of algorithms and methods to help designers to generate and evaluate transistors networks and logic gates. As data input, the framework accepts several descriptions like Boolean expressions, BDD, truth table, BLIF and Spice netlist format. This information is used by the transistor network generation module, which implements several algorithms found in the literature to generate transistor networks, for example: Direct generation from SOP equation, generation from factorized expression [6], BDD-based [7], with unateness optimization [8], respecting minimum stacks [2], Mux-based [2], Dual-rail [2], Topologically Complementary network (dual-graph) [9], and so on.

As these methods generate transistor networks with different arrangements and electrical behaviors, it is important to provide methods for analysis and evaluation of the resulting networks. For this purpose, this tool provides a set of estimation methods also available in the open literature: Elmore delay model [10], Dynamic

power consumption [2], Static power consumption (Subthreshold leakage [11] and Subthreshold and gate leakage [12]), and Area [2].

All these methods and algorithms offer a complete solution to designers that need to go from a logic description to a transistor-level implementation at the cell level. In this sense, SwitchCraft provides the possibility to investigate several switch networks and logic gate alternatives, and the means to choose the best solution that meets the design constraints.

Finally, this tool provides a transistor network visualization module. This is a useful feature, delivering a fast and efficient way to visualize all the different networks and gates generated in the framework. Spice netlist descriptions can also be easily imported and translated to transistor-based schematic. An example is shown in Fig. 3, which also includes transistor network profile information.

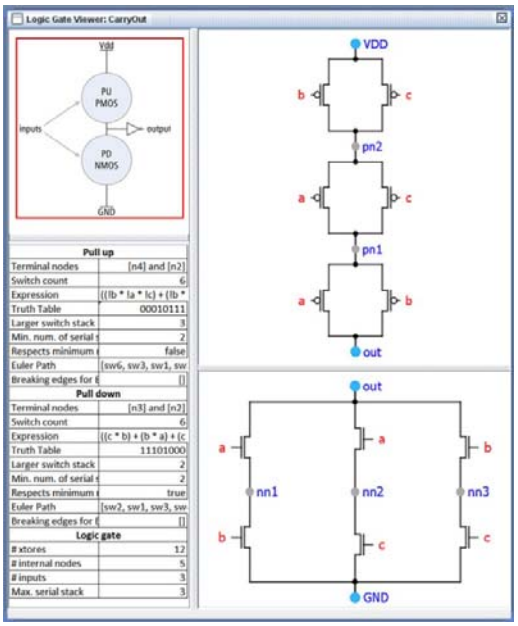


Figure 3 – Visualization and profile window for a carry-out full-adder logic gate.

A friendly graphical user interface (GUI) is available, as seen in Fig. 4. Each action performed in the graphical interface is linked to a prompt command. In this way, the user is also able to create script files using these commands in order to run tasks in batch mode. This feature makes possible to use SwitchCraft in a text only interface (prompt mode), allowing that other tools may be used together with SwitchCraft in an integrated design flow.

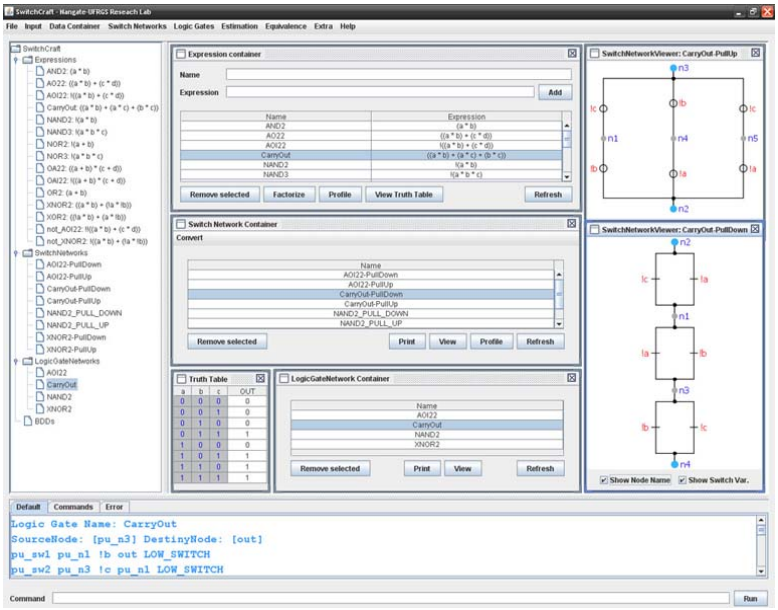


Figure 4 – SwitchCraft GUI.

4. Conclusions and Future Work

This paper presented a framework to automatically generate and evaluate transistor networks and logic gates. In the current version, SwitchCraft provides several methods and algorithms available in the literature. The tool can be used as a fast and easy way to investigate how different topologies may be used and explored to improve a target design. As future works, we intend to implement other alternative methods and algorithms, adding and providing extra capabilities for designers that want to extract the most optimized solution for digital circuit design. Also, the native didactic characteristics of this tool will be improved, giving more flexibility for students that want to learn about this topic.

5. Acknowledgments

Research partially funded by Nangate Inc. under a Nangate/UFRGS research agreement, by CNPq and Capes Brazilian funding agencies, and by the European Community's Seventh Framework Programme under grant 248538-Synaptic.

6. References

[1] D. Kagaris and T. Haniotakis, "A Methodology for Transistor-Efficient Supergate Design" IEEE Transactions on VLSI Systems, New York, USA, v.15, n.4. Apr. 2007. p. 488-492.

[2] L. S. Da Rosa Junior, "Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles" Tese (Programa de Pós-Graduação em Microeletrônica) – Instituto de Informática, UFRGS, Porto Alegre. 2008. 147 f.

[3] J. M, Rabaey, A. Chandrakasan and B. Nikolic, **"**Digital Integrated Circuits: A Design Perspective", 2nd ed. Upper Saddle River: Prentice Hall, 2005.

[4] T. J. Thorp, G. S Yee, C. M Sechen, "Design and synthesis of dynamic circuits". IEEE Transactions on VLSI Systems, New York, v. 11, n. 1, Feb. 2003, p. 141-149.

[5] N. H. E. Weste, D. Harris, "CMOS VLSI Design: A Circuits and Systems Perspective", 3rd ed. Boston: Pearson/Addison Wesley, 2005.

[6] V. Callegaro, L.S Rosa, A. I. Reis, R. P. Ribas, "A Kernel-based Approach for Factoring Logic Functions" Microelectronics Students Forum (2009 Aug.: Natal, BR-RN). SForum 2009 [recurso eletrônico]. CD-ROM [S.l.], Setembro 2009.

[7] L.S Rosa, R. P. Ribas, A. I. Reis, "Fast Transistor Networks from BDDs", Symposium On Integrated Circuits And System Design, SBCCI, 19., 2006, Ouro Preto, Brasil. Proceedings… New York: ACM, 2006. p. 137-142

[8] R. E. B. Poli, R. P. Ribas, A. I. Reis, "Unified Theory to Build Cell-Level Transistor Networks from BDDs", Symposium On Integrated Circuits And System Design, SBCCI, São Paulo, Brasil. Proceedings Los Alamitos: IEEE, 2003. p. 199–204.

[9] V. Callegaro, L.S Rosa, A. I. Reis, R. P. Ribas, "A Graph-based Solution for Dual Transistor Network Generation", VIII Student Forum on Microelectronics, 2008, Gramado. VIII Student Forum on Microelectronics Porto Alegre : SBC, 2008. CDROM.

[10] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers", Journal Applied Physics, Woodbury, USA, v. 19, n. 1, Jan. 1948, p. 55- 63.

[11] P. F. Butzen and Et Al, "Modeling Subthreshold Leakage Current in General Transistor Networks", ISVLSI. 2007

[12] P. F. Butzen and Et Al, "Simple and accurate method for fast static current estimation in CMOS complex gates with interaction of leakage mechanisms." 18th ACM Great Lakes Symposium on VLSI (GLSVLSI), 2008, Orlando. Proceedings of the 18th ACM Great Lakes Symposium on VLSI. New York : ACM, 2008, p. 407-410.

Clock Mesh Size for Wirelength and Capacitance Minimization

Guilherme Flach, Gustavo Wilke, Marcelo Johann, Ricardo Reis
{gaflach, wilke, johann, reis}@inf.ufrgs.br

UFRGS - Universidade Federal do Rio Grande do Sul

Abstract

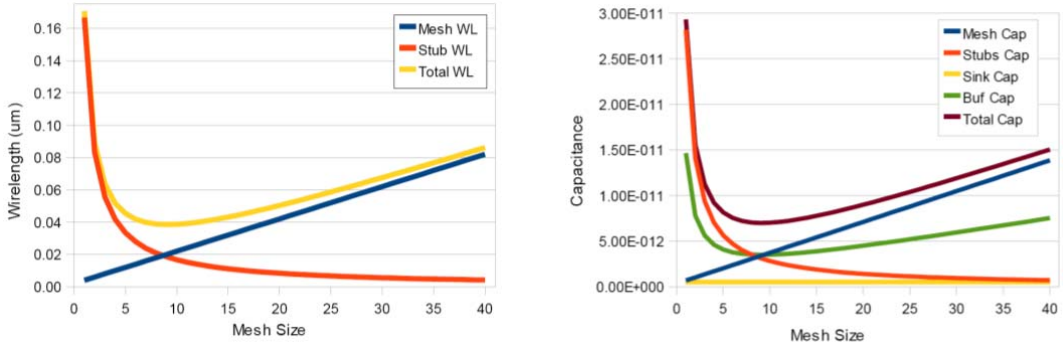
In this work we present some guidelines for mesh size selection aiming mesh wirelength or mesh capacitance reduction. We show that the both goals depend basically only on the number of sinks that mus be driven by the clock mesh. Also a study is presented analyzing how the clock skew changes as we move further away from the optimum mesh size.

1. Introduction and Motivation

Clock meshes are effective architectures for clock skew reduction mainly in sub-micron technologies where process and environmental variations become more aggressive. High-end microprocessors are usually designed using clock meshes to smooth out variability effects on clock delays, e.g. [1-3].

When designing a clock mesh the designer trades off skew improvements by higher power consumption. Denser is the clock mesh lower is the clock skew. By contrast power is not a monotonic function of the mesh size and in fact an optimum mesh size for power reduction can be found.

Let's take a look at the Figure 1(a), which shows the mesh total wirelength and its components as a function of the mesh size. The data were obtained from a *1000umx1000um* mesh driving *1200* sinks randomly placed on the circuit area. Although it is a snapshot for a given mesh, different meshes present the same behavior as the mesh size changes.



(a) (b)

Figure 1 – Mesh wirelength (a) and capacitance (b) as a function of the mesh size.

As it can be seen in Figure 1(a) the total mesh wirelength has a global minimum, which means that we waste routing resources if the appropriate mesh size is not chosen. This global minimum emerges due to the reduction on the average stub length as the mesh size increases in contrast with the mesh wirelength that increases linearly as the mesh size increases.

Similarly, the total mesh capacitance has a global minimum as it can be noticed in Figure 1 (b). For this chart, the buffers are sized to match the fanout-of-*2*, e.g. the total input capacitance of the buffers is equal to the total mesh capacitance divided by two. However, as we shall see, the fanout rule does not affect the optimum mesh size for total mesh capacitance reduction.

2. Assumptions and Definitions

In our analysis we suppose randomly placed clock sinks over an *LxL* die area. All clock sinks have the same capacitance. A clock mesh is built over this region with *m+1* rows and *m+1* columns evenly spaced and it is referred as having an *m* size. The stubs connect sinks to the nearest grid edges. Figure 2 shows an example of a *4x4* clock mesh.

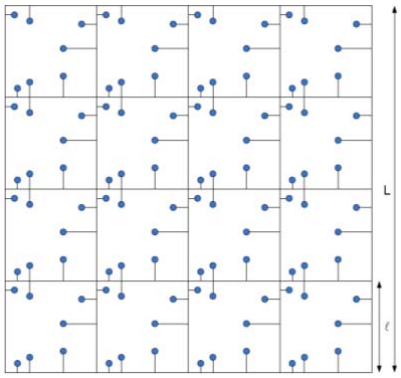


Figure 2 - A *4x4* clock mesh.

Although the buffer placement affects the clock skew, it can be ignored for the following discussion since we are interested only in the total mesh wirelength and capacitance. Later, for experiments, we shall describe the placement and buffer sizing strategy. For now, with no loss of generality, consider that mesh is driven by a single buffer sized according to the fanout-of-*n* rule: the size of the buffer is such that its input capacitance is equal to the total mesh capacitance divided by *n*.

3. Optimum Mesh Size for Wirelength and Capacitance Minimization

In this section we present analytic formulas for finding the optimum mesh size w.r.t total mesh wirelength and total mesh capacitance.

3.1. Average Stub Length Estimation

Stubs connect sinks to the nearest mesh edge so that the sinks are always connected to one of the four edges, which enclose it. The average stub length can be estimated by Equation 1 [4].

$$\int_0^{\ell/2} \frac{8(\ell/2 - x)}{\ell^2} x \partial x = \frac{\ell}{6} \tag{1}$$

3.2. Mesh Size for Minimum Total Mesh Wirelength

The total mesh wirelength is composed by both mesh wirelength and stub wirelength. As seen in the Figure 2 the mesh wirelength increases linearly and the stub wirelength decreases in a 1/*m* rate. It is also clear by Figure 2 that the total mesh wirelength has a global minimum. Equation 2 gives us the estimation of the total mesh wirelength.

$$W(m) = \underbrace{2L(m+1)}_{mesh} + \underbrace{(kL/6m)}_{stubs} \tag{2}$$

Now taking derivative of W equal to zero we obtain the mesh size, which minimizes the total wirelength as in Equation 3.

$$m_{wl}^{*} = \sqrt{\frac{k}{12}} \tag{3}$$

Notice that the mesh size depends only on the number of sinks. Note also that each bin should has in average 12 sinks for minimum wirelength.

3.3. Mesh Size for Minimum Total Mesh Capacitance

We define C_{total} as the sum of the mesh capacitance (mesh wires and stubs) and buffer capacitance. The mesh capacitance, C_{mesh}, is given by Equation 4.

$$C_{\text{mesh}}(m) = \underbrace{C_{wire}\left[2L(m+1)\right]}_{\text{mesh wirecap}} + \underbrace{C_{stub}(kL/6m)}_{\text{stub cap}} \tag{4}$$

Since we size buffers based on the fanaout-of-*n* rule, the total buffer capacitance is proportional to the mesh capacitance. Finally Equation 5 gives the total mesh capacitance.

$$C_{\text{total}}(m) = \underbrace{C_{mesh}(m)}_{\text{mesh cap}} + \underbrace{C_{mesh}(m)/n}_{\text{buffer cap}} + \underbrace{kC_{sink}}_{\text{sink cap}} \quad (5)$$

Equation 6 shows that the mesh size which minimizes the total mesh capacitance, m^*_{cap}, is proportional to the square root of the number of sinks and are not affected by the fanout-of-*n* rule. This is to say that we can change the fanout used for sizing buffers without affecting the optimum mesh size and so trade off power consumption and skew reduction [5].

$$m^*_{cap} = \sqrt{\frac{k}{12}}\sqrt{\frac{C_{stub}}{C_{wire}}} \quad (6)$$

Notice also that if we have $C_{stub} = C_{wire}$ we have $m^*_{cap} = m^*_{wl}$. However generally $C_{stub} < C_{wire}$ so that $m^*_{cap} < m^*_{wl}$.

4. Experiments

We perform a bunch of simulations to analyze the behavior of the clock skew as the mesh size goes away from the optimum mesh size.

4.1. Experiment Setup

A buffer is placed at each intersection of a vertical and a horizontal grid wire. The buffers are sized based on the fanout-of-*4* rule [6] addressing the load capacitance due to direct connected edges as well as the stubs and sink's capacitance connected to those edges. Figure 3 presents an example of the region addressed by a buffer.

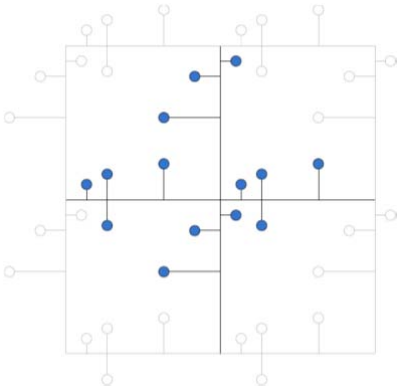


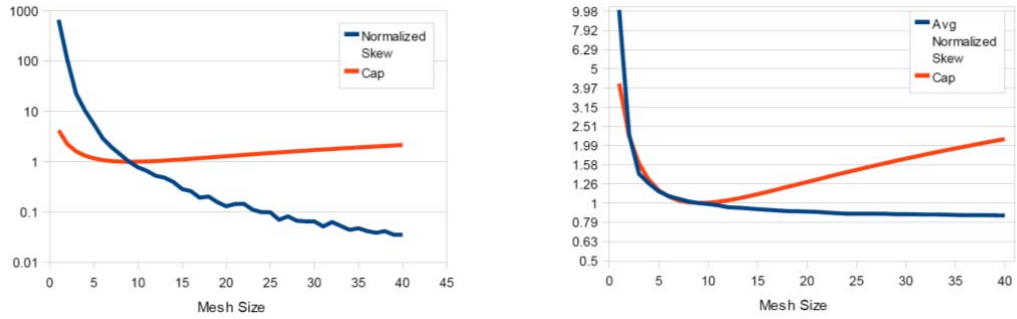
Figure 3 - The coverage region of a buffer.

The 65nm PTM [7] technology parameters are used in the electrical simulations. Wires smaller than *100um* are modeled with 1π and the remaining ones are modeled with 3π model. When variability is applied, *1000* Monte Carlo iterations are performed.

4.2. Result Analysis

First let us analyze the case where no input skew is applied to the clock mesh, that is, all mesh buffers receive the clock signal at same time. The log-plot of results is shown in Figure 4(a). As we can see, as the mesh size increases further from the optimum mesh size, we still obtain significant clock skew reduction. Although the total mesh capacitance increases and more power is dissipated, it is still worthwhile to increase the mesh size until the desired clock skew is achieved.

This scenario changes when input skew is applied to the mesh buffers. By analyzing the log-plot in Figure 4(b) we can observe that no significant skew reduction is achieve by increasing the mesh size further from the optimum size. This contrast with the rapidly increasing of the total mesh capacitance. The higher is the total mesh capacitance, the higher is the mesh power consumption. It is important to notice that the power consumption also increases by the increase of short circuit current.



(a) (b)

Figure 4 - Skew and capacitance trade-off (a) when no input is applied and (b) when input skew is applied.

5. Conclusions

In this paper we outlined analytical formulas to compute the optimum mesh size for both wirelength and capacitance reduction. As we observed the optimum mesh size for wirelength reduction is only dependent on the number of sinks. The optimum mesh size for capacitance is also dependent on the number of sinks, but it is also affected by the ratio between the mesh and the stub wire capacitances.

Another key point is to observe that we obtain little skew improvement when moving further from the optimum mesh size, which may not compensate the increasing in total mesh capacitance and by as a consequence the power consumption.

6. References

[1] S. Tam et al., "Clock generation and distribution for the first ia-64 microprocessor," IEEE Journal of Solid-State Circuits, vol. 35, no. 11, pp. 1545–1552, Nov. 2000.

[2] T. Xanthopoulos et al., "The design and analysis of the clock distribution network for a 1.2ghz alpha microprocessor," in Proceedings of the IEEE Internaional Solid State Circuits Conference, ISSCC, Feb. 2001, pp. 402–403.

[3] P. Restle et al., "The clock distribution of the power4 microprocessor," in Proceedings of the IEEE Internaional Solid State Circuits Conference, ISSCC, Feb. 2002, pp. 144–145. 476–479.

[4] G. Flach, G. Wilke, M. Johann, R. Reis. "A Study on Clock Mesh Size Selection" *Proc. First IEEE Latin American Symposium on Circuits and Systems (LASCAS)*, 2010.

[5] Shinya Abe; Hashimoto, M.; Onoye, T., "Clock Skew Evaluation Considering Manufacturing Variability in Mesh-Style Clock Distribution" in *9th International Symposium on Quality Electronic Design*, 2008. ISQED 2008. 17-19 March 2008 , pp. 520-525 .

[6] Sutherland, I.; Sproull, B.; Harris, D. Logical effort: designing fast CMOS circuits. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[7] PTM. "Predictive Tecnology Model" Apr, 2010. http://www.eas.asu.edu/~ptm

Video Coding 1

Power Efficient Motion Estimation Architecture Using QSDS Algorithm with Dynamic Iteration Control

[1,3]Marcelo Porto, [2]João Altermann, [2]Eduardo Costa, [3]Luciano Agostini, [1]Sergio Bampi
{msporto,bampi}@inf.ufrgs.br, {jaltermann, ecosta}@ucpel.tche.br
{porto, agostini}@ufpel.edu.br

[1]Institute of Informatics - Federal University of Rio Grande do Sul – UFRGS
[2]Polytechnic Institute - Catholic University of Pelotas – UCPEL
[3]GACI – Federal University of Pelotas – UFPel

Abstract

This paper presents a power efficient and low hardware cost architecture for motion estimation (ME) using a Quarter Sub-sampled Diamond Search algorithm (QSDS) with Dynamic Iteration Control (DIC) algorithm. QSDS-DIC is a new algorithm for motion estimation and it is based on the Diamond Search algorithm and the sub-sampling technique. The aspect of reducing significantly the number of Sum of Absolute Differences (SAD) calculations enables the development of an efficient hardware design for the ME. Moreover, the Dynamic Iteration Control available to the architecture, allows that the desired throughput can be achievable with a restriction in the number of iterations. This aspect enables a power efficient architecture, since the number of clock cycles can be reduced by using DIC technique. The implemented architecture uses blocks of 16x16 samples and it was described in VHDL. Synthesis results are presented for TSMC 0,18um CMOS standard cell. The architecture can reach real time for HDTV 1080p with power consumptions of 32.92mW.

1. Introduction

Motion estimation (ME) is the most important task in the current standards of video compression. However, the intensive computation required to estimate the motion vectors leads to an increase of power consumption. According to [1], typically, 60-80% of the total encoder computational power is due to the motion estimation block. Thus, for the applications where the real-time video in portable devices operated by battery, such as cell phones, wireless handheld terminals, etc., has to be considered, the low power aspect has to be taken into account.

Several techniques for reducing power consumption in ME architectures are presented in literature. The main goal is to reduce the computational complexity of the motion vector by using the most adequate search algorithm. Diamond Search (DS) [2] algorithm can be a good choice to the reduction of the computational requirements, because this algorithm can drastically reduce the number of SAD calculations, when compared to the conventional Full Search (FS) algorithm. The use of DS algorithm enables a significant increase of search area, with quality results close to the FS results. Moreover, the possibility of reducing SAD calculations can appreciably reduce the hardware resources that are necessary to achieve real time for high resolution videos. Another form to reduce the number of SAD calculations is the use of Pixel Sub-sampling (PS) technique [2]. According to [3], by using PS at a 4:1 rate, it is possible to reduce the number of SAD calculations by 75% and to speed up the architecture four times.

This paper presents a power efficient architecture for ME using the new Quarter Sub-sampled Diamond Search with Dynamic Iteration Control (QSDS-DIC), which is based on DS algorithm and uses PS 4:1 and DIC. This architecture was firstly presented in [3] and the results showed that this architecture can reach real time for HDTV 1080p videos with a low hardware cost. In this paper we show the capability of reducing power consumption of the ME of [3] by exploring the use the DIC aspect in the QSDS algorithm. In fact, by using DIC technique, the number of iterations that are used to generate the motion vectors can be controlled, and thus, the number of clock cycles can be reduced with no penalties in the desired throughput and motion vector quality. This enables a significant power reduction in the ME architecture.

The architecture was described in VHDL. Synthesis results are presented for TSMC 0.18μm CMOS standard cell technology. The architecture can reach real time for HDTV (1920x1080 pixels) in the worst case, with a power consumption of 32.92mW, saving 17.5 % when compared with QSDS without DIC architecture. The QSDS-DIC was developed for real time HDTV applications, but this architecture presents good results for power consumption even working on low resolution videos as CIF. These results are achievable due to the efficiency of the QSDS-DIC algorithm that is able to save SAD calculation and hardware resources.

The rest of paper is organized as follows: section II summarizes the QSDS-DIC algorithm. The designed architecture is presented in section III and section IV presents the hardware synthesis results and comparisons with related work. Finally, section V presents the conclusions of this paper.

2. QSDS-DIC Motion Estimation Algorithm

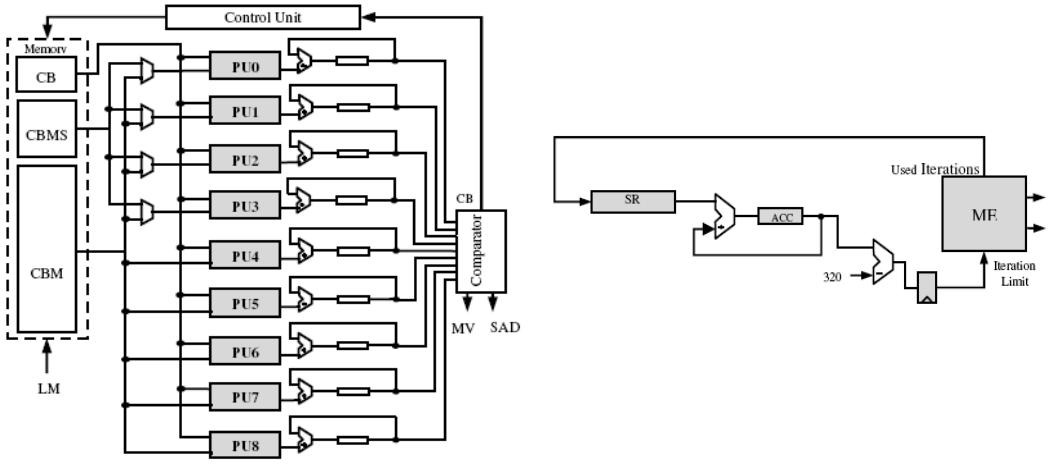
In this section we summarize the QSDS-DIC algorithm for motion estimation proposed by [3]. This new algorithm is based on three main principles: (1) Diamond Search algorithm; (2) 4:1 Pixel Sub-sampling (PS) technique and (3) dynamic iteration control. These main principles will be detailed in the next paragraphs.

Diamond Search (DS) algorithm defines two diamond patterns, the Large Diamond Search Pattern (LDSP) and the Small Diamond Search Pattern (SDSP) [3]. The LDSP consists of nine comparisons around the center of the search area. The SDSP is used in the end of the search, when four candidate blocks are evaluated around the center of the LDSP, to refine the final result. The search ends when the lower SAD is found at the center of the LDSP. So, the SDSP is applied and the search is finished. When the best match is found in a vertex, more five blocks are evaluated to form a new LDSP. If the best match was obtained in an edge, three blocks more are evaluated. The 4:1 PS technique is applied to each evaluated candidate block. Thus, each line of the block has eight samples, instead of 16 in a non sub-sampled block. The number of lines is also sub-sampled, only eight lines are considered. Thus, each candidate block with 16x16 samples, becomes a sub-sampled block with 8x8 samples. Then, by using PS it is possible to improve the throughput reducing the hardware cost, with a small loss in terms of quality.

The Dynamic Iteration Control (DIC) was designed focusing in the hardware design of ME. This feature allows a good trade off among hardware resources consumption, throughput, synchronization, quality and power consumption. The DIC explores the characteristics of the algorithm, and dynamically controls the use of iterations, contributing to power saving and reducing the hardware utilization.

3. QSDS-DIC Architecture

This section presents the QSDS-DIC architecture. The designed architecture for the QSDS-DIC algorithm uses SAD as a distortion criterion [3]. The architectural block diagram is shown in Fig. 1(a). The architecture has nine processing unities (PU). The PU can process eight samples in parallel and this means that one line of the 16x16 sub-sampled block is processed in parallel. So, eight accumulations must be used to generate the final SAD result of each block, one accumulation per line.



(a) **(b)**

Fig. 1 – (a) QSDS –DIC block diagram architecture (b) Block diagram of DIC

3.1 Memory Organization

The internal memory is organized in 15 different local memories [3]. A local memory (LM) stores the region with the nine candidate blocks of the first LDSP and all the possible blocks for the next step. Thus, when the control unit decides which blocks must be evaluated in the next step, the LM already has this data. LM memory is composed of 16 words with 128 bits. Another 13 small memories are used to store the candidates block (CBM) and one for the current block (CB). These 14 memories are composed of 8 words with 64 bits (8 samples of 8 bits) and they store one sub-sampled block with 8x8 samples [3].

LM memory is read line by line and the data is stored in the CBM memories. Nine CBMs are used to store the candidate blocks from the LDSP. Four CBMs are used to store the blocks of the SDSP. These blocks are always stored, and they are ready to be calculated if the control decides to start the SDSP. This solution speeds up the architecture and reduces the memory access latency. When the SDSP mode is active, the control unit selects a multiplex and the PUs receives the data from these memories [3].

3.2 Dynamic Iteration Control – DIC

The restriction in the number of iterations is important to make it easier to synchronize this module with other video coder modules, while at the same time it allows the number of cycles used to generate a motion

vector closer to the worst case [3]. Then, a dynamic restriction is considered in DIC. DIC allows the algorithm to use a variable number of iterations, depending on the algorithm necessity. 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 20 iterations to use (this is a variable number). When a motion vector uses less iterations 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].

The architecture used to implement DIC is shown in Fig. 1(b). 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 20 iteration per motion vector, so the used iterations for the last 16 vectors are subtracted of 320 (maximum value for a set of 16 motion vectors), and the result is the number of available iterations for the next motion vector generation.

4. Synthesis Results

The developed architecture was described in VHDL and synthesized to TSMC 0.18um standard cells technology. The synthesis was generated using the Cadence Encounter RTL Compiler tool. The results are presented for the worst case, considering the minimum frequency for real time processing at 30 fps (frames per second). The worst case considers the maximum number of iterations available in the DIC. For the lower resolution standards, as QCIF, CIF and VGA, the DIC was set to five. In this case, each motion vector can use a maximum of 192 clock cycles. The highest resolution videos use a DIC with restriction of 20, for maximum iteration per motion vector in the worst case. Thus, in this case, each motion vector can use 582 clock cycles.

By using DIC, the motion estimator works in a dynamic search area. Considering that the maximum number of iteration per motion vector is set to five, the maximum number of iteration, for the last 16 vectors is 80 (maximum value for a set of 16 motion vectors). The maximum search area in this case is 320x320 pixels, since it is considered that all motion vectors use all iterations available of it. For the DIC with 20 iterations, the maximum search area is 1300x1300 pixels.

The synthesis results for the QSDS architecture, with and without DIC, are presented in tab. 1. The results are presented considering the minimum operational frequency for real time processing, at 30 fps, for many resolution standards. The QSDS-DIC architecture needs a very slow frequency (approximately 0.5MHz) to work with QCIF resolution. The power consumption is also small, only 12.51mW.

The QSDS-DIC architecture was developed to reach real time for HDTV 1080p resolution. This architecture can achieve this performance at 141.1MHz in the worst case. The power consumption is small, less than 33mW to process HDTV videos.

Tab. 1 – Synthesis Results

Video Standard 30fps	QSDS –DIC		QSDS	
	Frequency (MHz)	Power (mW)	Frequency (MHz)	Power (mW)
QCIF	0.57	12.51	0.57	13.92
CIF	2.28	12.55	2.28	14.58
VGA	6.91	13.26	6.91	16.07
SDTV	23.5	18.06	23.5	21.92
HD 720p	83.8	28.23	83.8	34.12
HD 1080p	141.4	32.92	141.4	39.92

The QSDS-DIC synthesis results were compared with some published solutions for low power ME as [4-7]. Tab. 2 presents the results of the QSDS-DIC architecture and some related work. Our architecture presents the best results for area resources utilization and power consumption, when HDTV video processing is taken into account. Our architecture presents the smaller area resources utilization, in terms of number of transistors (Xtor). Comparing against [4] and [7], that have performance results for HDTV, our solution can save 65 % and 51 % power consumption, respectively.

The work of [5] and [6] presents architecture solutions for low resolution videos (CIF). As should be observed, even developed to process HDTV, our solution has also lower power consumption in CIF. As can be seen in table 2, our solution presents less power when compared against [6], and it presents power consumption close to the result of [5]. However, the solution of [5] was developed especially for mobile applications, and works only with QCIF and CIF videos. Moreover, this solution uses a less power supply of 1.2V, while our solution uses a power supply of 1.8V. Even compared with solutions for low resolution videos, our architecture has the lower hardware utilization. It can prove the quality of the QSDS-DIC algorithm for hardware implementation.

Tab. 2 – Synthesis Results

	Motion Estimation Architectures					
	[10]	[5]	**Our**	[8]	[7]	**Our**
Video Standard	HD 1080p	HD 1080p	**HD 1080p**	CIF	CIF	**CIF**
Process (µm)	0.18	0.18	**0.18**	0.18	0.13	**0.18**
Transistor	620K	2.25M	**94K**	508K	1M	**92.3K**
Frames (fps)	-	30	**30**	30	30	**30**
Freq. (MHz)	155	108	**141.4**	27	13.5	**2.2**
Power (mw)	68	95	**32.92**	22.5	12	**12.55**

5. Conclusions

This paper presented a high performance and power efficient motion estimation architecture for real time HDTV encoding. The developed architecture uses the QSDS-DIC algorithm, and it presents smaller hardware resource utilization. The architecture was described in VHDL and synthesized to TSMC 0.18µm standard cells technology.

The synthesis results shows that the developed architecture can reach real time for HDTV 1080p videos with a frequency of 141.1MHz, with a power consumption of 32.92mW, for the worst case. The results also show the advantages in power consumption by using DIC. Our solution presents the best result for area resources utilization and power consumption when compared against some related work for low power ME architectures, working on HDTV 1080p videos.

6. Referências

[1] M. Panovic, and A. Demosthenous, "A Low Power Block-Matching Analog Motion Estimation Processor" IEEE International Symposium on Circuits and Systems, vol. 5, pp. 4827–4830, 2005.

[2] P. Kuhn, Algorithms, Complexity, Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Springer, June 1999.

[3] M. Porto, S. Bampi, A. Susin, and L. Agostini, "Architectural Design for the New QSDS with Dynamic Iteration Control Motion Estimation Algorithm Targeting HDTV" 21st Symposium on Integrated Circuits and Systems Design. September 2008.

[4] Y. Murachi, T. Matsuno, K. Hamano,J. Miyakoshi, M. Miyama, and M. Yoshimoto "A 95mW MPEG2 MP@HL Motion Estimation Processor Core for Portable High Resolution Video Application" Symposium on VLSI Circuits Digest of Technical Papers, 2005.

[5] M. Miyama, J. Miyakoshi, Y. Kuroda, K. Imamura, H. Hashimoto, and M. Yoshimoto, "A Sub-mW MPEG-4 Motion Estimation Processor Core for Mobile Video Application" IEEE Journal of Solid-State Circuits, vol. 30, no 9, pp. 1562–1570, September 2004.

[6] Y-Han Chen, T-Chien Chen, C-Yung Tsai, S-Fang Tsai, and L-Gee Chen, "Data Reuse Exploration for Low Power Motion Estimation Architecture Design in H.264 Encoder" Journal of Signal Processing Systems, vol. 50, pp. 1-17, January 2008.

[7] S. Warrington, W. Chan, and Subramania Sudharsanan, "Scalable High-Throughput Architecture forH.264/AVC Variable Block Size Motion Estimation**",** International Symposium on Circuits and Systems, 2006. ISCAS 2006.

A Low-Cost Hardware Architecture Design for Binarizer Defined by H.264/AVC Standard

[1]André Martins, [1]Vagner Rosa, [1]Dieison Deprá, [1]Sergio Bampi
{almmartins, vsrosa, depra, bampi}@inf.ufrgs.br

[1]UFRGS – Universidade Federal do Rio Grande do Sul

Abstract

This paper presents a hardware design of the binarizer part of the CABAC (Context-Based Adaptive Binary Arithmetic Coding) entropy encoder as defined in the H.264/AVC video compression standard. The architecture proposed in this paper is able to reach the HDTV throughput requirements for all modules of the CABAC binarization process. The proposed solution was described in VHDL and the synthesized results show that the developed architecture reaches enough performance in FPGA and ASIC to process HDTV videos in real-time and it has the lowest area among all related work in literature .

1. Introduction

The H.264/AVC standard [1], [2] is the state-of-art in digital video compression, reaching gain in compression rate of 50% compared to the previous Standards. However, the lower bit-rate to a given bitrate was achieved through sophisticated algorithms that increased significantly the computational complexity in encoding and decoding process. Therefore, many hardware architectures have been developed and proposed, mainly to process high definition videos. One of the main tools that provide high compression performance to H.264/AVC is the entropy encoder CABAC. However, developing high-performance hardware architectures to CABAC is a great challenge because their algorithms are extremely sequential having strong data dependency. These features deal very hard the exploring of parallelism being one of the bottlenecks of the system [2].

The coding process used by CABAC is composed by three basic stages: binarization, context modelling, and arithmetic encoding as shown in Fig. 1. In the binarization stage, the syntactic elements (SE) are mapped to chains of binary symbols, according to the type of SE being processed. Each symbol of these chains is called "bin", and the complete chain is called "bin string". [3]

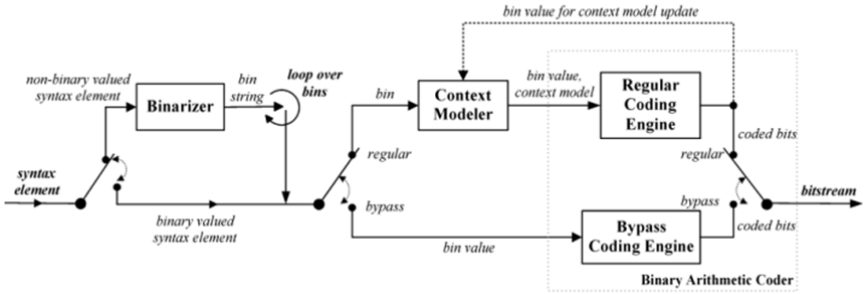


Fig. 1 – CABAC block diagram [2]

Currently, many works focusing CABAC architectures were found in literature, but only in a few works all the stages of codification are developed. The majority is restricted to the problems of the arithmetical codification and/or the context modeling, sometimes ignoring the binarization.

The previous works, which proposed hardware architectures for CABAC, usually do not focus in or simply do not present implementation synthesis data on the binarization process [6]-[8]. This happens because the CABAC data bottleneck is not in the binarizer, the other blocks of CABAC are critical in performance and in their blocks the most of authors are developed architectures. Further about 90% of the total symbols are derived from DCT coefficients and foreach one of these elements is necessary to apply the binarization process. Based on these statistics, the work presented in [3] decided to implement the techniques of binarization in hardware only for SEs that derived from the DCT coefficients, leaving the remaining binarization techniques to be processed in software. However, the first architecture solution found in the literature for CABAC [3] showed that the binarizer could demand 52% of total area of CABAC.

In the next section, will be present all the methods of binarization defined by H.264/AVC standard. The architecture proposal is discussed in section 3. Section 4 presents the results of our developed architecture. Finally, we present some conclusions in Section 5.

2. Binarization Process

The binarization process consists of mapping integer values in a sequence of bits that represents the original value. This mapping is carried through with objective to reduce the alphabet of symbols, thus simplifying the amount of elements to be modeled, minimizing the costs of the context modeling and facilitating the task of arithmetical codification. Each bit, generated through this process, is called "bin" and the set of all "bins" (bits), generated from the mapping of an input value, was called "bin string". The size of "bin string" generated for each input value is variable and depends on the type of SE that is being processed and on the current context. The goal to use the binary mapping with variable size of "bins" is, exactly, to generate the smallest possible representation for SEs that occurs more frequently, making possible compression of bitstream.

Tab.1 - Example of binarization using U, TU and FL methods

SE	**Bin**		
	Unary	**Truncated Unary** (*cMax=4*)	**Fix Length** (*cMax=4*)
0	0	0	0000
1	10	10	0001
2	110	110	0010
3	1110	1110	0011
4	11110	1111	0100

To implement the binary mapping process, the CABAC defines a set of seven binarization methods, some of them are extremely complex. The decision of which the form of binarization will be used is based on the SE type, macroblock type (or sub macroblock), slice type and in the SE value, that is being treated. The mainly binarization methods are composed by four basic techniques: Unary (U), Unary Truncated (TU), Fixed Length (FL) and Concatenated Unary/Kth order Exp-Golomb (UEGk). Beyond these, other three special binarization forms are defined for specific SEs: Macroblocks and SubMacroblocks (MB&SUB), Coded Block Pattern (CBP) and Quantized Parameter Delta (QPd). The MB&SUB method uses ROM table to map SE to bin. The CBP method is given by FL binarization (*cMax*=4) concatenated with TU binarization (*cMax*=2). The QPd method is given by U binarization of the mapped value of SE, because the SE binarized by QPd can admit negative values.

Tab.2 - Example of UEGk binarization process

SE	**UEGk Bin** (*k=0 uCoff=14*)	
	Truncated Unary	**Kth Exp-Golomb**
0	0	-
1	10	-
...	...	-
13	11111111111110	-
14	11111111111111	0
15	11111111111111	100
16	11111111111111	101
17	11111111111111	11000
18	11111111111111	11001
19	11111111111111	11010
20	11111111111111	11011
21	11111111111111	1110000
22	11111111111111	1110001
N	TU(SE) cMax=14	EGk(SE-14)

Tab.3 – The same Tab. 2 binstring but produced by equations Eq. 1; Eq. 2 and Eq. 3

SE	**UEGk Bin** (*k=0 uCoff=14*)	
	Unary	**Fixed Length**
0	0	-
1	10	-
...	...	-
13	11111111111110	-
14	111111111111110	-
15	1111111111111110	0
16	1111111111111110	1
17	11111111111111110	00
18	11111111111111110	01
19	11111111111111110	10
20	11111111111111110	11
21	111111111111111110	000
22	111111111111111110	001

The unary method translates the unsigned integer value in a chain of one's ("1") with length defined by symbol value and concatenated by zero ("0") in the end. The length of the bin for the truncated unary method is limited by the parameter *cMax,* in other words, it means the length of bin binarized by TU should not be larger than *cMax*. The fixed length method bypasses the SE's which are already in their binary representation. The FL method depends of the parameter *cMax* too. The table (Tab. 1) below shows an example about the mapping of the SE's to U, TU and FL methods.

The UEGk bin string is a concatenated of a prefix and suffix bin string and depends of parameters *k* (defines the minimum input value from which exist suffix) and *uCoff* (defines the maximum prefix length). The prefix part is defined by TU method and the suffix is defined by Exp-Golomb code as shown below at table Tab.2.

3. Designed Architecture

Known that the binarization is not the bottleneck of the system, but it can demand more than a half of total area of CABAC and the arithmetic encoder doesn't need process 1bin/cycle to be efficient, the strategy used to develop the design of this work was a multi-cycle implementation which could be able to reuse of some operators. To get this objective, we focus the efforts in optimize or replace the binarization method more complex, the UEGk. Analyzing the UEGk bins shown at Tab. 2, it was detected that the same bins could be generated by using the methods FL and U. The prefix bin is generated by method U as defined as equation Eq. 1 and the suffix method is generated by method FL as defined as equation Eq. 2 and Eq. 3. At table Tab. 3 is shown how the UEGk bins are generated by these equations. Note that compared with Tab. 2, it's really the same bins.

$$prefix(SE) = \begin{cases} U(SE), & SE < uCoff \\ U(uCoff + size(suffix) - k - 1), & SE \geq uCoff \end{cases} \quad (1)$$

$$suffix(SE) = FL(2^k + SE - uCoff) \quad (2)$$

$$cMax_{suffix} = size(suffix) \quad (3)$$

The architecture developed to implement all binarization methods is presented at figure Fig. 3. The module called *Dec* calculates the bin size generated by UEGk method. The registers *prefix* e *suffix* are used to calculate some bin values. The register *size* stores the bin size. The five memories ROM are used to define bins without behavior logic generated by MB&SUB method.

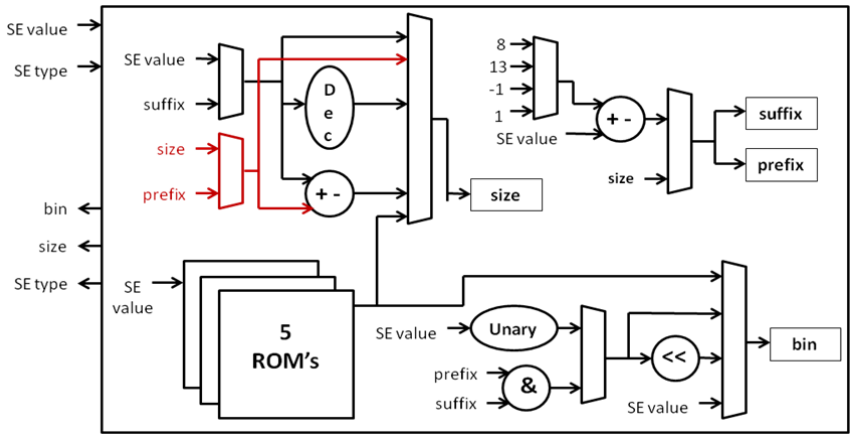


Fig. 2 – The developed binarizer block diagram

The SE can admit many types and each type of SE has its specific binarization process. Apart from SE_type, the control should schedule the SE's to their correct binarization method and determine their parameters *cMax*, *uCoff* e *k,* when present. The diagram of figure Fig.4 shows that the binarizer proposed finishes the processing of SE in maximum three cycles. In one cycle, our architecture executes the FL binarization, in two cycles it executes the other methods and the QPd, UEGk and MB&SUB can demand three cycles to generate a bin.

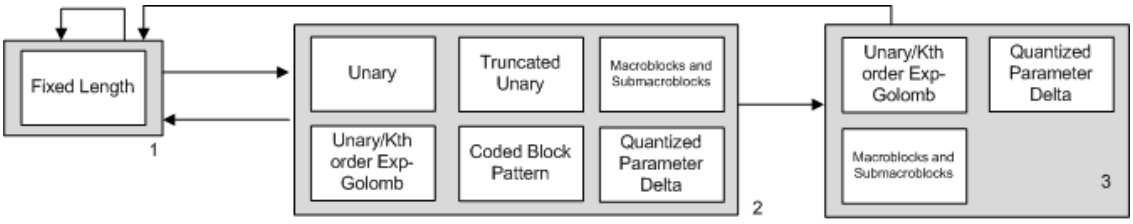


Fig. 3 – The binarizer functional diagram

4. Results

The architecture developed was described in VHDL and synthesized to FPGA and ASIC. The table Tab. 4 shows the synthesis results in terms of area and frequency.

Tab.4 – Synthesis results

Leonardo – TSMC 0.18um		Leonardo - TSMC 0.35um		ISE – FPGA Virtex II VP30			
Area	**Frequency**	**Area**	**Frequency**	**Slices**	**FFs**	**LUTs**	**Frequency**
2013 gates	348.3 MHz	2258 gates	205.7MHz	382	238	651	241.1 MHz

The table Tab. 5 shows the comparison of the results with other's related works. There are some other works which developed the binarizer or part of the binarizer and they don't have on the table because a comparison wouldn't be fair or accurate [10], [11]. It's important highlight that [4], [5] didn't develop all binarization methods although the their good results. The binarizer presented in [3] demands a lot of area compared with our architecture. In the other hand, to get the low-cost of area, we needed to decrease the throughput, but our design is able to process video HDTV in real time too.

Tab.5 – Comparison with related works

	Binarizer?	**Area**	**Throughput** (bins/cycle)	**Frequency** (MHz)
[5] (0.35um)	UEGk e U	Gates: 6500	2	343
[5] (FPGA)	UEGk e U	Slices: 403	2	185
[4] (FPGA)	UEGk e U	Slices:367	1	100
[3] (FPGA)	Yes	Slices: 3424	1	243
Ours (FPGA)	Yes	Slices: 382	0.66-0.8	241.1
Ours (0.35um)	Yes	Gates: 2258	0.66-0.8	205.7

5. Conclusions

In this paper, a new hardware architecture design for binarizer of the H.264/AVC CABAC has been proposed. The related works founded in literature do not focus in architectures for binarizer and we showed the implementation of all the methods of binarization defined by H.264/AVC standard is essential for application of the CABAC. Our design is able to process videos FullHD 1080p in real time to main profile and gets the better results in terms of area compared with related works and supports all binarization processes.

6. Bibliography

[1] RICHARDSON, I. H.264/AVC and MPEG-4 Video Compression - Video Coding for Next-Generation Multimedia. Chichester: *John Wiley and Sons*, 2003.

[2] MARPE, D.; SCHWARZ, H.; WIEGAND, T. "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard". *In: IEEE TCSVT*, Vol. 13, Nº 7, July 2003.

[3] DEPRA, D. A. ; ROSA, V. S. ; BAMPI, S. . "Design and Implementation of a High-Performance Architecture for Binarization Methods of Defined by H.264/AVC Standard". *In: XIV Workshop Iberchip, 2008*.

[4] OSORIO, R. Roberto.; Bruguera, J. D. "Arithmetic coding architecture for H.264/AVC CABAC compression system." *DSD, 2004. Proceedings. Euromicro Symposium* In. pp 62-69. Sept. 2004.

[5] R. R. Osorio and J. D. Bruguera, "High-Throughput architecture for H.264/AVC CABAC compression system," *IEEE TCSVT.*, vol. 16, no. 11, pp. 1376–1384, Nov. 2006.

[6] PASTUSZAK, G. "A High-Performance Architecture of the Double-Mode Binary Coder for H.264.AVC." *IEEE TCSVT,* v. 18, n. 7, p. 949-960, July 2008.

[7] L. Li, Y. Song, T. Ikenaga, and S. Goto, "A CABAC Encoding Core with Dynamic Pipeline for H.264/AVC Main Profile," in *Proc. IEEE APCCAS*, Dec. 2006, pp. 761 – 764

[8] H. Shojaina and S. Sudharsanan, "A high performance CABAC encoder," in *Proc. IEEE Int. NEWCAS Conf.*, Quebec City, Canada, Jun. 2005, pp. 104–107.

[9] INTERNATIONAL TELECOMMUNICATION UNION. *ITU-T Recommendation H.264 (03/05):* advanced video coding for generic audiovisual services. [S.l.]. 2005.

[10] W. Zheng et al.: "Efficient Pipelined CABAC Encoding Architecture" in *IEEE Transactions on Consumer Electronics*, Vol. 54, No 2, May 2008

[11] Chien-Chung Kuoel al.: "Design of a Low Power Architecture for CABAC Encoder in H.264" in *Proc. IEEE APCCAS*, Dec. 2006, pp. 243-246

Adaptive Distortion Metric Architecture for H.264/AVC Video Coding

[1]Guilherme Corrêa, [1]Cláudio Diniz, [2]Luciano Agostini, [1]Sergio Bampi
{grcorrea, cmdiniz, bampi}@inf.ufrgs.br, agostini@ufpel.edu.br

[1] Microelectronics Group (GME) – UFRGS – Porto Alegre, Brazil
[2] Group of Architectures and Integrated Circuits (GACI) – UFPel – Pelotas, Brazil

Abstract

Video coding in portable devices is a challenging research subject nowadays. Just recently, the research for dynamically adaptive video coding systems has been addressed. This work proposes a power-aware architecture with adaptability to choose a different distortion metric accordingly to the power/energy constraints, the coding quality and the time required to perform the video coding. With our architecture, it is possible to trade-off image quality and computational complexity. Three distortion metrics were implemented and its efficiency was assured through the reuse and sharing of logic resources, such as operators and registers. Clock gating was also applied to further reduce the power dissipation. Results showed a power decrease of 65% to 80%, depending on which distortion metric is used. Also, the used area was reduced around 14% through resource sharing. The developed architecture achieves by far the minimum requirements of the design in which it is used, allowing the encoding of HD 1080p digital videos in real time.

1. Introduction

The last technological advances have brought us a wide range of portable multimedia-capable devices, although the power supply keeps being one of the main limitations in such designs. As portable multimedia devices generally do not require the best image quality and present a stronger power constraint than other non-portable devices, the development of algorithmic and architectural approaches for trading off image quality and power consumption in such applications can be explored.

The Intra-frame prediction of H.264/AVC [1] is responsible for decreasing the spatial redundancy present in each video frame. On the other hand, the Inter-frame prediction, composed by the motion estimation and motion compensation modules, is responsible for the reduction of temporal redundancy of a video. These two types of prediction are performed in several different modes of operation, which are then compared in order to select the best one in terms of bit-rate and distortion. H.264/AVC reference software uses a Rate-Distortion method to select the optimal coding mode, which is extremely expensive due to the large number of candidate modes provided in H.264/AVC. Several distortion metrics, which are different in terms of quality and complexity, can be used in this method. As the complexity influences directly in the power consumption, the choice of which distortion metric is used in the encoding process is related to the power and the energy requirements of the whole video coding system.

In this paper, we propose a power-aware architecture in which it is possible to choose a distortion metric used accordingly to the power and energy constraints, the coding quality and the time required to perform the video coding. This approach can be specially used in devices where these requirements and constraints change frequently, such as mobiles, palmtops, handhelds and portable televisions. The architecture is, thus, based on an efficient trade-off between image quality and computational complexity. Three distortion metrics were implemented in the architecture and its efficiency was assured through the reuse and sharing of logic resources, such as operators and registers.

The rest of this paper is organized as follows. Section 2 explains the Rate-Distortion method for mode decision and the mathematical basis for the three metrics used in this work. Section 3 presents the implemented architectures. Section 4 shows the obtained results in terms of area, performance, power and energy and draws a comparison between them. Section 5 concludes this work and indicates some future work.

2. Mode Decision and Distortion Metrics

A large number of coding modes is provided by the H.264/AVC standard. The reference software uses a Rate-Distortion method to select the optimal coding mode. This can be done through the use of Lagrangian minimization, as proposed by Wiegand [2], which is the most effective method to improve the compression ratio (a ratio between raw and coded video) with a controlled loss in the output video quality, as shown in (1).

$$J = D + \lambda.R \quad (1)$$

In (1), **J**, **D** and **R** are the cost, the distortion and the rate of a mode, respectively, and λ is a Lagrangian multiplier dependent on the quantization parameter (QP) used in the coding process [1]. Rate is the number of bits per second of coded video and distortion is a measurement of information loss in the coding process. When considering H.264/AVC, near-optimal rate-distortion optimization is very complex than the video coding

process itself, because of the abundance of coding modes. However, as this stage is a non-normative feature, mode decision can be designed in a low complexity manner, targeting both real-time and low power requirements in mobile devices. For high resolution video coding applications, low complexity distortion metrics are also useful to achieve real-time requirements.

Several distortion metrics can be used to estimate the **D** value in (1), such as the Sum of Absolute Differences (SAD), the Sum of Absolute Transformed Differences (SATD), the Sum of Squared Differences (SSD) [3]. Among the cited metrics, SAD is the less complex and less accurate one, while SSD is the most complex, since it uses a multiplication unit to calculate the squared differences. In this work, three different metrics were specifically chosen in order to provide three different levels of quality accuracy and power-consumption. The chosen distortion metrics are explained in the following paragraphs.

SAD is the most intuitive distortion metric, since it calculates the difference between two blocks based on simple subtractions between their pixels. The absolute value of the difference between each pixel in the original block and its corresponding pixel in the predicted block is calculated. All the absolute differences are then summed in order to provide a single number which represents the block distortion, as shown in (2), where **P** is one candidate predicted MB (or block), **O** is the original MB (or block), **m** and **n** are the dimensions of the MB (or block) in samples.

Although still similar to SAD, the SATD metric includes a frequency transform in its calculation, increasing significantly its complexity. On the other hand, SATD is more accurate than SAD, since it evaluates both objective and subjective quality of the video through comparisons performed in the frequency domain. The most used transform in SATD is the Hadamard Transform [1], introduced in the video coding process of H.264/AVC through the direct and inverse transform modules (T and T^{-1}). After the subtraction between the pixels of the original and predicted blocks, the differences are transformed from the spatial to the frequency domain. The absolute values are taken from the transformed samples and then summed as in SAD calculation. The SATD function is shown in (3), where **S** represents the sample in the (**i**, **j)** position of the transformed block and **N** is the dimension of the MB (or block) in samples.

In SSD, the differences between the pixels of the predicted and the original blocks are squared and then summed. As in SAD and SATD, the minimum SSD value defines the smallest difference between two blocks. As it involves a multiplication, this distortion metric is more complex and time-consuming than the two others presented. The SSD function is shown in (4), where **P**, **O**, **m** and **n** present the same meaning as in (2).

$$SAD = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left|P_{i,j} - O_{i,j}\right| \quad (2) \qquad SATD = \sum_{i}^{N}\sum_{j}^{N}|\, S_{ij}\,| \quad (3) \qquad SSD = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(P_{i,j} - O_{i,j}\right)^2 \quad (4)$$

3. Implemented Architectures

As the distortion calculation is the most used function during the Inter-frame and Intra-frame predictions, the efforts of this work are all concerned on the architecture of this module.

3.1. Hierarchical SAD+SATD+SSD Architecture

An architectural version containing the three distortions metrics without any optimization was implemented to be used in comparisons with the proposed solution. The three distortion metrics were implemented separately and instantiated in a higher-level architecture. As the metrics were designed in a pipelined approach with one arithmetic operator per stage, a different number of clock cycles is used to calculate the distortion value in each one of them.

The SAD, the SATD and the SSD architectures receive eight samples per clock cycle (four samples from the predicted block and four samples from the original block). Then, four subtractions are performed in parallel and the results are stored in the first set of temporal registers.

In SAD, the absolute value calculators are followed by the adders' tree. Five groups of temporal registers are used in the five-stage pipeline. Once filled the pipeline (four clock cycles), the SAD value of a 4x4 block is generated after four clock cycles.

In SATD, the differences between the input samples are stored in a set of 16 registers during four clock cycles, since they are all required for the Hadamard transform execution. The Hadamard transform was divided in four clock cycles, since its 64 arithmetic operations can occur in four steps. The architecture was organized in 10 pipeline stages.

In the SSD architecture, the multiplication unit was implemented in a shift-and-add approach where the multiplication terms are generated through AND operations between the bits of the multiplication factors. The partial products are then added using two by two additions. The SSD architecture was implemented in seven pipeline stages.

3.2. Optimized SAD+SATD+SSD Architecture

As the three distortion metrics are not used at the same time, an architecture that shares a set of adders, subtractors and registers was designed in order to reduce the circuit area required and the overall power consumption. Thus, we have investigated which of the registers and operators can be used in the calculation of more than one distortion metric.

A group of multiplexers was added before the input of the shared resources in order to allow the selection of the input values based on the control signal that identifies the metric in use. Figure 1 shows the developed architecture. Each stage is separated from the previous one by a register barrier, represented by the name "REG". As the 4x4 Hadamard transform is composed by a series of arithmetic operations, its four stages are also separated by registers. For example, the adders' tree present in the SAD and SATD architectures shown in the previous subsection are implemented in this optimized architecture with the same registers and adders, since they are never used at the same time. Five blocks which correspond to adders' tree are shown in Figure 1.

When the distortion metric used is SATD, the module "4x4 Hadamard transform" is used and the modules "ANDs", "Reg Terms", "Sum Terms Level 1" and "Reg Level 1", which are used just in SSD, are skipped during the execution. In the same way, the module "4x4 Hadamard transform" is ignored when the distortion metric used is SSD and the other blocks are used for the distortion calculation.

In order to decrease still more the power consumption of the circuit, the final version of the architecture also includes the use of a clock gating technique, since not all the registers are used during the computation of the three metrics. Thus, the clock activity in such registers can be disabled to reduce the dynamic power consumption. Three clock signals were created, besides the original clock signal: clk_SAD, clk_SATD and clk_SSD. The registers used in just one of the metrics were described in processes which are just sensible to their correspondent clock signals. Registers used in the calculation of two metrics were created in processes sensible to the two correspondent metrics and registers used in all the three metrics are sensible to the original clock. The new clock signals are generated by simple logical operations between the original clock and the control signal that indicates which of the three metrics is being used at the moment.

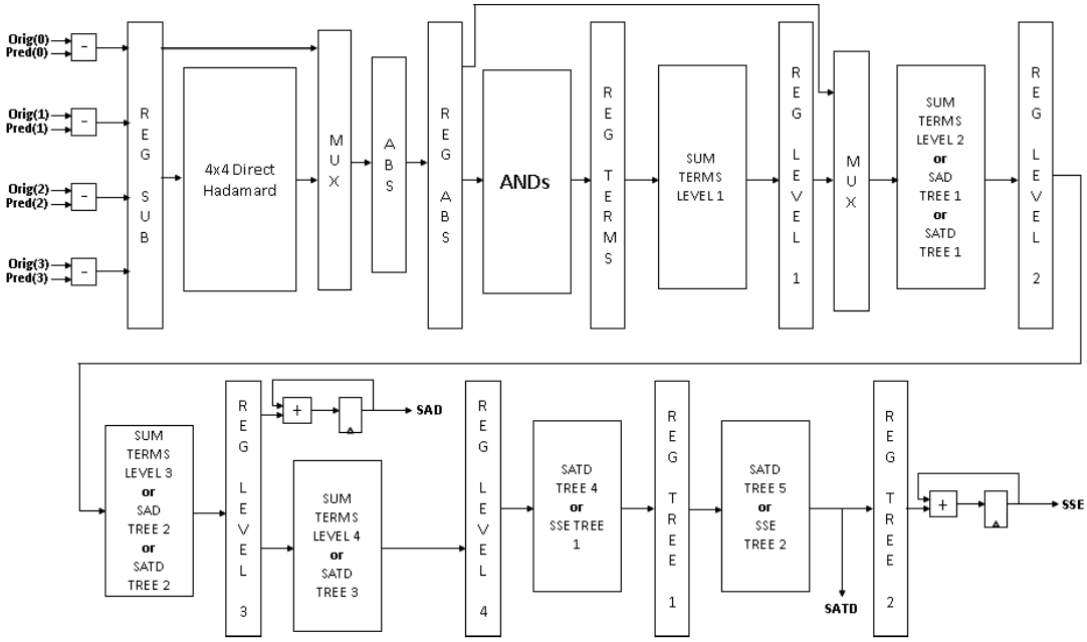


Fig. 1 – SAD+SATD+SSD architecture with resource sharing

4. Results

All the three solutions (SAD+SATD+SSD without resource sharing, with resource sharing and with resource sharing and clock gating) were described in VHDL and synthesized to TSMC 0.18 µm standard-cells technology through Synopsys tools. Table 1 shows the obtained results in terms of power dissipation, area and maximum frequency achieved for the three solutions.

In Table 1, we can observe a reduction of 13.9% in the used area of the second solution, which explores the resource sharing, when compared to the first solution that only instantiates a separated VHDL description for SAD, SATD and SSD at the same module. This slight reduction occurred because a large set of multiplexers was added to the architecture of the two other solutions before the shared registers and the arithmetic operators, increasing the area of the optimized circuit.

Table 1 - Power, area and frequency results

Version	Area (um²)	Fmax (MHz)	Power (mW) @ 100 MHz
SAD+SATD+SSD	470,789	281.4	65
SAD+SATD+SSD w/ resource sharing	405,328	265.1	27.82 @ SAD 28.12 @ SATD 28.73 @ SSD
SAD+SATD+SSD w/ resource sharing and clock gating	407,057	265.1	12.9 @ SAD 22.9 @ SATD 19.0 @ SSD

Very expressive reduction in power dissipation was achieved by the last two solutions when compared to the first one, as shown in Table 1. As the first solution does not present resources sharing between the metrics nor clock gating in some registers, it calculates and provides the three results in its outputs at the same time, thus resulting in high power consumption. While the area reduction was around 13.5% in the final architecture, the power consumption decreased about 80% in SAD, 65% in SATD and 71% in SSD operation. This happened because most of the shared resources are registers, which are elements that consume much more power than the arithmetic operators. The power reduction was achieved through the sharing of logical resources such as arithmetic operators and registers. The last solution achieves less power dissipation than the others because of the application of clock gating technique to the pipeline register barriers. These results were obtained through the Power Compiler tool, from Synopsys [4], considering the 0.18 µm technology from TSMC. The power analysis was performed by the application of average sample values as input of the developed architecture.

The addition of multiplexers has also enlarged the critical path of the circuit, thus increasing the critical delay. Table 1 shows the reduction of 6% in the maximum frequency of the optimized architecture, which has dropped from 281.4 MHz to 265.1 MHz.

The developed architecture can be used in both Inter-frame and Intra-frame prediction. In order to encode a HD 1080p video (1920 x 1088 pixels) in real time (30 frames per second), 244,800 macroblocks must be encoded per second. This way, considering that the proposed module is part of a complete H.264/AVC encoder design that operates at 130 MHz, a macroblock must be encoded in at least 531 clock cycles and the proposed architecture must satisfy this requirement. In the developed architecture, the SAD value of a 4x4 block is calculated in seven cycles and the complete flow of the Intra-frame coding, composed by the Intra-frame prediction, the mode decision, the residual calculation and the direct and inverse transform and quantization is performed in 22 cycles for each block. As a macroblock is composed by 16 4x4 blocks, 352 clock cycles are used to encode it, satisfying the requirement. An amount of 400 and 432 clock cycles are necessary to encode a complete macroblock using SATD and SSD metrics, respectively, which also satisfies the requirements.

5. Conclusions and Future Work

This work proposes a power-aware adaptive architecture which allows the use of three distortion metrics (SAD, SATD and SSD). It was designed using a resource sharing approach which decreases the total area used and the overall power consumption of the circuit. Clock gating was also applied and resulted in an overall decrease varying from 65% to 80%, depending on the distortion metric being used.

The used area of the original architecture was reduced around 14% in the final version. On the other hand, the critical delay has increased 6% due to the insertion of several groups of multiplexers that allow the sharing of logical resources. The developed architecture achieves by far the minimum requirements of the design in which it is used, allowing the encoding of HD 1080p digital videos in real time.

As future works, it is intended to expand the idea of the multiple distortion metric architecture through the development of an algorithm which dynamically chooses the used metric Also, it is intended to use optimized adders in the trees of the metrics, such as adder compressors.

6. References

[1] RICHARDSON, I. E. G. “H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia”, John Wiley & Sons Publishers, USA, 2002.

[2] WIEGAND, T. et al., "Rate-Constrained Coder Control and Comparison of Video Coding Standards", IEEE TCSVT, Jul. 2003, p. 688-703.

[3] PURI, A., CHEN, X., LUTHRA, A. “Video coding using the H.264/MPEG-4 AVC compression standard”, Signal Processing: Image Communication 19 (2004) 793–849

[4] Synopsis PowerCompiler, datasheet available at homepage: http://www.synopsys.com/products/power/power_ds.pdf

A New Parallel Motion Estimation Algorithm

[1]Diego Noble, [1,2]Marcelo Porto, [1]Gabriel Siedler, [1]Luciano Agostini
{dnoble. ifm, gsiedler. ifm, agostini}@ufpel.edu.br, {msporto}@inf.ufrgs.br

[1]GACI, Federal University of Pelotas – UFPel
[2]Institute of Informatics, Federal University of Rio Grande do Sul - UFRGS

Abstract

In this paper, we present a new algorithm focusing in high quality fast motion estimation process for high definition video coding. This algorithm provides more efficiency to avoid local minima falls in fast motion estimation. This algorithm is based on Diamond Search, and it was called Multi-Point Diamond Search algorithm (MPDS). The multi-point search could be done in a serial or parallel approach. In the parallel approach, there are no penalties in the performance. This solution could be a good approach for hardware implementation, since there is no data dependency and the multi-points can be calculated in parallel. The MPDS algorithm was implemented and evaluated in ten HD video sequences. The results show an average quality gain of 3.72dB in comparison with original Diamond Search.

1. Introduction

Motion Estimation (ME) presents the highest computational complexity among all steps of the current standards for video coding. In fact, ME represents 80% of the total computational complexity of current video coders [1]. The search for best vectors is known to be very expensive in terms of calculations and, consequentially, in terms of processing time. The Full Search (FS) algorithm must explore all possibilities in a given search window, which implies in a very high computational cost, especially for high resolution videos. Based on this fact, it is important to explore new solutions which bring a good tradeoff between objective quality (PSNR) and complexity.

There are many fast algorithms and techniques in scientific literature which handle with this complexity at different levels of impact in objective quality (PSNR). Generally, these algorithms exploit the characteristic of locality among temporal correlated blocks and good results in terms of numbers of calculations could be accomplished. However, these algorithms have a weak point which is the increase in local minima falls with the increase of the video resolutions, especially for high activity video sequences. This undesired feature results in worse motion vectors than the ones which FS algorithm would generate and, consequentially, perceivable losses in the visual quality. In the H.264/AVC standard [2], the current most efficient video coding standard, there is no restriction about how the block matching is done in the motion estimation process, so there is a lot of space to explore new ideas.

In this paper, we propose a new algorithm for fast motion estimation targeting high quality when processing high definition videos. This algorithm provides an efficient way to avoid local minima falls in the fast ME algorithms applications and therefore an increase in terms of final quality. This algorithm is not focused on a specific standard and it can be used with all current standards. The proposed algorithm uses the Diamond Search (DS) algorithm [3] as basis and it is called Multi-Point Diamond Search (MPDS) algorithm after it. This algorithm can be a good option for hardware implementation since the multi-points can be calculated in parallel, without data dependency. This paper presents the new algorithm, its results and comparisons with other solutions. The MPDS was applied to ten HD 1080p test video sequences and the results shows that a gain of 7.42dB could be achieved in some sequences in comparison with the original DS. The computational cost is also increased in comparison with the original DS; however it is more than 485 times lower when compared with FS algorithm for the same search area.

There are several fast search algorithms in the literature, as [1], [4] and [5], for example. All these algorithms use some techniques to speed up the search and also to achieve good motion vector, not necessarily the best ones. The algorithm presented in [4] combines two algorithms and also uses threshold to speed up the process, as [5] does. None of these algorithms are good solutions to avoid local minima in the search process, and this is the purpose of this our presented algorithm.

The rest of this paper is organized a follows: Section 2 describes the main concepts related to the ME process and also some concepts for common fast ME algorithms. The proposed algorithm is showed in Section 3 and the results at Section 4. The comparative results are presented in the Section 5. Conclusions are given at the section 6.

2. Motion Estimation

Full Search (FS) algorithm generates the optimum motion vector in a given search area, however, it demands a very high computational cost. There are many fast algorithms for ME in the literature, as Diamond Search (DS), Hexagon Search (HS), Dual Cross Search (DCS) [6] and others. These algorithms have in

common the use of a search pattern (diamond, hexagon and cross) that is repeated while a stop condition was not satisfied. When the search ends, a small pattern is applied to refine the final result. In this paper we use the search engine of the DS algorithm applied in our algorithm.

Diamond Search (DS) algorithm defines two diamond patterns, the Large Diamond Search Pattern (LDSP) and the Small Diamond Search Pattern (SDSP) [3]. The LDSP consists of nine comparisons around the center of the search area. The SDSP is used in the end of the search, when four candidate blocks are evaluated around the center of the LDSP, to refine the final result. The search ends when the best match is found at the center of the LDSP. So the SDSP is applied and the search is finished. When the best matching is found in a vertex, more five blocks are evaluated to form a new LDSP. When the best matching is found in an edge, more three blocks are evaluated, forming a new LDSP.

3. The MPDS Algorithm

The MPDS algorithm starts the search for the best matching in five different positions of the search area. Each position, except the central one, is inside a sector (A, B, C and D), as presented in Fig. 1. The MPDS algorithm uses the same search engine of the DS algorithm, when searching for the best block matching. It starts doing the LDSP until the criterion of termination is found and then it applies the SDSP to obtain a final refinement. However, the MPDS is not restricted to only one start point, exactly to avoid the same local minimum which the DS would reach. Moreover, the MPDS defines five different start points and five independent DS algorithms are triggered in the same search area.

Fig. 1 describes the MPDS algorithm. Each initial search point is defined by its coordinates inside the sector. The point (0,0) is the central position and it will obtain the same vector which the DS algorithm would due to the deterministic nature of the algorithm. The search in the sectors A, B, C and D starts according the distance parameter ***d***. The ***d*** parameter is the distance (number of pixels in X and Y axis) from the central point (0,0). Sectors A, B, C and D start the search respectively at positions (*d*,*d*), (-*d*,*d*), (-*d*,-*d*) and (*d*,-*d*). When the search ends for all evaluated regions, the MPDS algorithm selects the best result from the five applied diamonds. The variation of ***d*** parameter can substantially change the achieved quality result for a given sequence.

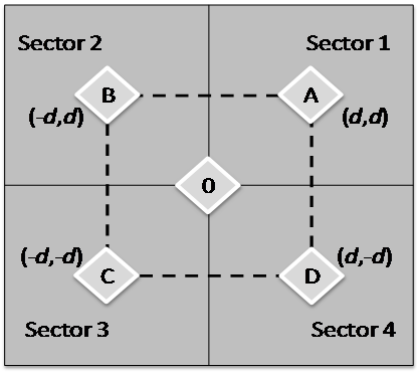


Fig. 1. MPDS algorithm starting search in five points

It is important to notice that the MPDS algorithm could be executed in a parallel or in a sequential manner. This algorithm was developed focusing on the quality of the results generated by the ME process, so both implementations, parallel or sequential ones, could be used since the quality results will be the same. Besides, the MPDS algorithm is an efficient solution to execute in processors with support to multi-thread or multi-core, exploring parallel programming features. The parallel or sequential implementations could be done also in hardware, when high throughputs are required. Time related results were omitted since we focused the design specifically at a high quality motion estimation process.

4. Simulation Results

The proposed algorithm was described in C language and it was evaluated in ten test video sequences [7]. The used search window was defined as 256x256 pixels, the block size was defined as 4x4 pixels and the similarity criterion is defined as the SAD [3]. The experiments consider the first two hundreds frames of all the ten HD 1080p sequences. The time spent for all simulations was 432 hours in the sequential way in a Quad Core Q8200 with 2 GB of RAM.

Fig. 2 show the curves of PSNR for the MPDS algorithm in the ten HD 1080p test video sequences [7], considering the variation of the ***d*** parameter (1 to 20 pixels). This variation was evaluated to find the best distance for the average case. The effects of this variation are very different to each sequence, depending of the video characteristics. In Fig. 2, the curve “traffic” is related to the sequence “traffic_dissonving_to_trees”; the curve “tomatoes” is related to “rolling_tomatoes” sequence and the curve “pedestrian” is related to the

"pedestrian_area" sequence. The names were abbreviated to allow a better representation in the graphic. PSNR is related to the Y axis and is measured in *dbs* and *d* is related to the X axys.

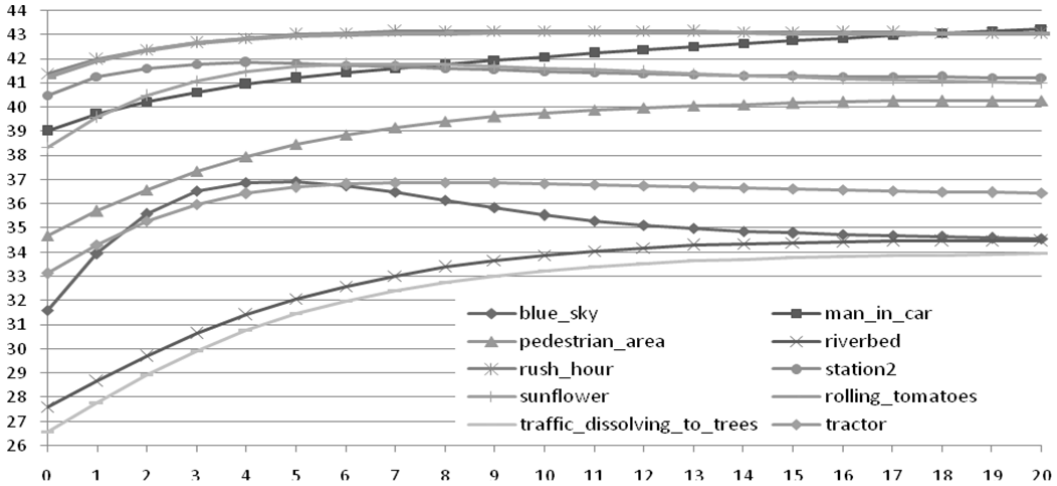


Fig. 2. PSNR curves for the MPDS algorithm with variation of *d* parameter, *d*=0 means original DS

The value 0 for the ***d*** parameter was introduced to show the results of the DS algorithm seeing that the results are the same for both algorithm when ***d=0***. The curves in Fig. 2 show an important quality gain, in comparison with original DS. The block size was defined as 4x4 pixels because a higher number of vectors would be generated and consequently the effect of the multi-point search would be more precisely visible.

As Fig. 2 suggests, the optimal distance from the point (0,0) depends on the characteristic of the considered sequence. In sequences with high movement as **riverbed** or **traffic**, the MPDS outperforms DS with a gain in PSNR around 7dB. This is a considerable value for objective quality. In this case the optimum values for ***d*** were 19 and 27, respectively. The best results for MPDS algorithm was found for the **traffic** sequence, with a gain of 7.4dB

Sequences with low movement, such as **rush_hour** and **station2**, where DS achieved a good result, the quality gain of MPDS was not so expressive. The optimal distances in these cases were 9 and 4, respectively, with a PSNR gain of 1.7dB for **rush_hour** and 1.3dB for **station2**. However, high detail sequences with a small global movement, as the **sun_flower**, **tractor** and **blue_sky** showed a fast increase in the PSNR for lower ***d*** values, followed by a significant reduction in the gain with the increase in ***d*** value. The optimal ***d*** values were 7, 8 and 5, respectively.

The analysis of the variation of ***d*** parameter and its consequences, presented in Fig. 2, revealed the possibility of definition of an optimal distance in the average case. Considering the results for the ten video sequences, the best value for ***d*** parameter is 17. It means that the MPDS presents an average gain of 3.72dB in comparison with the original DS, using ***d*=17** for this set of video samples. This gain over the original DS is obtained due to the search in the sectors A, B, C and D.

The center sector (original DS) is the best option for 52.01% of the motion vectors. The other 47.99% of the motion vectors were found at one of the four additional sectors. This evaluation demonstrates the potential of reduction in the local minima falls when the MPDS algorithm is used, since 47.99% of the vectors were found in other regions than that defined by DS algorithm. The distribution over the sectors is very similar, varying from 11.07% to 12.90% (sectors D and A, respectively).

5. Comparative results

The average result of the MPDS algorithm (***d*** = 17) is compared to original DS and FS algorithms in the Tab. 1 which also presents the average results for quality and computational cost in terms of number of SAD operations. The quality results were measured in the average PSNR and percentage of residual reduction (PRR) in comparison with the simple differential coding. The FS algorithm was evaluated in four different search windows, 16x16, 32x32 and 64x64 pixels.

The MPDS algorithm surpasses the PSNR of original DS in about 3.72dB in the average case. However, there is an increase in the computational cost. The MPDS computes about six times more SAD operations than original DS. This increase in the computational complexity may not represent any significant cost in terms of performance because the parallel nature of the MPDS algorithm makes possible to execute the five diamonds (the central one and sectors A, B, C and D) at the same time. The only additional cost, in comparison with the original DS, is the decision step between the five results that comes after the search step.

Comparing to FS algorithm, the MPDS algorithm, in the average, can achieve better PSNR results than the FS16 (search window of 16x16 pixels) and the FS32 (search window of 32x32 pixels), with a gain of 5.52db and 1.20db, respectively. Moreover, considering the computational cost, the FS16 uses 1.3 times more SAD

operations than the MPDS, and FS32 uses 6.4 times more SAD operations than MPDS. The FS algorithm can only outperform the MPDS for the search windows of 64x64 pixels. In this case, the FS64 can achieve a PSNR 1dB higher than MPDS with a computational cost 29 times higher than MPDS. The MPDS algorithm presents a better tradeoff between computational cost and achieved PSNR in this case.

Tab. 1 - Evaluation results for MPDS, DS and FS

Algorithm	PSNR (dB)	PRR (%)	# SADs x 10^6
DS	35.41	73.77	9.28
MPDS	**39.13**	**82.64**	**54.67**
FS16	33.62	32.48	70.09
FS32	37.94	79.77	348.78
FS64	40.78	85.02	1,543.17

Comparisons with published works are very difficult because there is no published work with the same set of video samples as the used in this work. The most part of the published works uses low resolution sequences to evaluate its algorithms. In [1], three HD sequences are used for test (station2, pedestrian_area and rush_hour), however, the algorithm was evaluated inside the JM12.4, the reference code for H.264/AVC. Its results were presented for the whole coding process, and not just for the ME.

6. Conclusions

In this paper, we presented a new algorithm for high quality fast motion estimation called Multi-Point Diamond Search (MPDS). The MPDS was implemented and ten HD video sequences were used to evaluate it. Its results were compared to original DS and the FS algorithm. The results show that the MPDS can achieve an average gain of 3.72dB, in comparison with original DS. The MPDS can also outperform the FS algorithm for a search window of 32x32, with a reduction in the computational cost of 6.4 times. Consequently, it has a better tradeoff between computational cost and quality.

The MPDS presents a computational cost increase in comparison with original DS. However it is possible to use parallelization to speed up the whole process. Thus the MPDS can be faster than FS and as fast as the original DS. As future work, we intend to adapt the MPDS algorithm for a two-step implementation. Based on preliminary results, a reduction in 20% of total computations is likely to be achieved. We also intent to develop a hardware architecture for this algorithm, based on [8] that present a hardware architecture for DS algorithm. Using four instances of the original architecture we can implement the MPDS algorithm in a high performance and high quality motion estimation architecture. An implementation of the MPDS inside the most recent version JM is also planned thus allowing our algorithm to be tested inside whole process of video coding. Furthermore, a more precise comparison with others algorithms would be possible.

7. References

[1] Cheng, Y., Chen, Z. and Chang, P., “An H.264 Spatio- Temporal Hierarchical Fast Motion Estimation Algorithm for High-Definition Video”, IEEE International Symposium on Circuits and Systems, ISCAS, pp. 880-883, 2009.

[2] 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), JVT-G050r1, Geneva, May 2003.

[3] Kuhn, P., Algorithms, Complexity, Analysis and VLSI Architectures for MPEG-4 Motion Estimation, Springer, June 1999.

[4] Po, L., et al., “Novel Directional Gradient Descent Searches for Fast Block Motion Estimation”, IEEE Transaction on Circuits and Systems for Video Technology, Volume 19, pp. 1189-1195, 2009.

[5] Tasdizen, O., et al., “Dynamically Variable Step Search Motion Estimation Algorithm and a Dynamically Reconfigurable Hardware for Its Implementation”, IEEE Transactions on Consumer Electronics, Volume 55, pp. 1645-1653, 2009.

[6] Porto, M., et al., “Investigation of motion estimation algorithms targeting high resolution digital video compression,” ACM Brazilian Symposium on Multimedia and Web, ACM, New York, pp. 111-118, 2007.

[7] Xiph.org: Test media, available at <http://media.xiph.org/video/derf/>, December, 2009.

[8] M. Porto, L. Agostini, S. Bampi and A. Susin, “A high throughput and low cost diamond search architecture for HDTV motion estimation,” In: IEEE International Conference on Multimedia & Expo, 2008, Hannover. IEEE International Conference on Multimedia & Expo, 2008.

A Dedicated Hardware Solution for the H.264/AVC Half-Pixel Interpolation Unit

[1]Marcel Moscarelli Corrêa, [1]Mateus Thurow Schoenknecht,
[1]Luciano Volcan Agostini
{mcorrea.ifm,mateust.ifm,agostini}@ufpel.edu.br

[1]UFPel – Federal University of Pelotas

Abstract

This work presents a hardware solution for the H.264/AVC half-pixel interpolation unit. This solution is able to process very high definition videos as QHDTV (3840x2048 pixels) at 33 frames per second. This solution can also be integrated to a complete Motion Estimation architecture without limiting the other modules performance. This architecture was described in VHDL and synthesized to two different Xilinx FPGA devices and achieved the best results when compared to related works.

1. Introduction

Video coding is an important research area due to the increasing demand for high definition digital video for applications like the Internet, digital television broadcasting, storage and many others.

A video coding standard primarily defines two things: (1) a coded representation or syntax which describes the visual data in a compressed form, and (2) a method to decode the syntax to reconstruct the visual information [1].

The H.264/AVC (Advanced Video Coding) is the most recent and most efficient video coding standard. It is designed to achieve much higher compression rates when compared to older standards [2]. However, the H.264/AVC has a very high computational complexity, which makes difficult for software solutions to encode high definition videos in real time, i.e. 1920x1080 pixels at 30fps. For this reason, dedicated hardware architectures are being designed.

In a raw digital video there is a lot of redundant information that can be explored in order to compress it. There are three kinds of redundancy. The spatial redundancy is the similarity in homogeneous areas within a frame, the temporal redundancy is the similarity between sequential frames, and finally, the entropic redundancy is the redundancy in the bit stream representation.

The Motion Estimation (ME) is the module that explores and reduces the temporal redundancy of a video. It works by splitting the current frame in several macroblocks (16x16 pixels) and searching in the previous coded frames (reference frames) for the macroblock that is most similar to the current one. When the most similar macroblock is found, a Motion Vector (MV) is generated indicating the motion. In the H.264/AVC Motion Estimation the macroblock can be partitioned into smaller blocks to find better matches [2].

However, the most similar block can be found in a fractional position, indicating a movement smaller than one pixel [1]. Our work focuses on a part of the Fractional Motion Estimation.

This paper is structured as follows: Section 2 presents the sub-pixel accurate Motion Estimation; Section 3 presents more details about the half-pixel interpolation process; Section 4 presents a software valuation; Section 5 presents the processing unit design; Section 6 presents the data flow and operation mode of our design; Section 7 presents some synthesis results and a comparison with a related work, and finally, Section 8 concludes this work.

2. Fractional Motion Estimation

A characteristic that contributes to the high compression rates achieved by the Motion Estimation of the H.264/AVC standard is the possibility to generate fractional motion vectors [1]. In other words, a movement that happens from a frame to another is not restricted to integer pixel positions only.

The Fig. 1a shows an integer motion vector pointing to a 4x4 block that is directly present in the reference frame and the Fig. 1b shows a fractional motion vector pointing to a 4x4 block that is not in the reference frames. The samples that are used by the Fractional Motion Estimation must be obtained through interpolation of integer position samples.

The H.264/AVC includes both half-pixel and quarter-pixel accuracy. This process increases significantly the computational complexity of the ME.

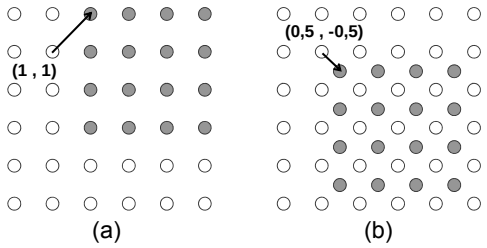


Fig. 1 – Two Motion Vectors pointing to different blocks.

3. Half-Pixel Interpolation Process

A single half-pixel *y* that has adjacent integer positions is derived by first calculating an intermediate value called y_1 by applying the 6-tap FIR filter presented in equation (1). In equation (1), A to F represent the nearest six integer position luminance samples (0-255) in the horizontal or vertical direction. Then, the equation (2) is applied to y_1.

$$y_1 = A - 5B + 20C + 20D - 5E + F \tag{1}$$

$$y = Clip_{0-255}\left[\left(y_1 + 16\right) >> 5\right] \tag{2}$$

A single half-pixel *y* that has adjacent half-pixel positions instead of integer positions (because its diagonally aligned between integer positions) is derived by first calculating an intermediate value called y_1 by applying the 6-tap FIR filter (3) using as *C* and *D* the y_1 values of the two adjacent half-pixels and using as *A*, *B*, *E* and *F* the *y* values of the other nearest half-pixels, and finally, applying (4) to y_1.

$$y_1 = A - 5B + 20C_1 + 20D_1 - 5E + F \tag{3}$$

$$y = Clip_{0-255}\left[\left(y_1 + 512\right) >> 10\right] \tag{4}$$

It is important to notice that to calculate a half-pixel that is diagonally aligned between integer position samples, either horizontal or vertical nearest half-pixels can be used because these wield an equal result [2]. In this work, our design uses the horizontal closest half-pixels.

This way, there are three half-pixels types: H Type, calculated using the closest horizontal integer position samples; V Type, calculated using the closest vertical integer position samples; and D Type, calculated using the closest horizontal half-pixels (which are V Type half-pixels).

The Half-Pixel Interpolation Unit gets the best match block (composed by integer position samples) from the ME and interpolates a new search area around these samples. Using this search area the Half-Pixel Refinement will test all the eight possible matches inside this search area to check if there is a block composed by half-pixels more similar to the current block than the one that the ME found.

4. Software Evaluation

Several video sequences with different resolutions were coded using the default configuration of the H.264/AVC reference software [3]. The resolutions are QCIF (176x144), CIF (352x288), 4CIF (704x480) and 1080p (1920x1080). The software evaluation was realized in order to check the utilization rate of each macroblock partition size in the Motion Estimation. The partition sizes in H.264 are 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4 pixels.

It was observed that 94.75% of the chosen blocks had a size greater or equal to 8x8 pixels. This way, this design works only with 8x8 blocks in order to reduce drastically the circuit complexity and cost.

Giving support to this block size is interesting because it is the same block size used by older video coding standards, like MPEG2. Some minor changes can make our design compatible with those.

Valuations with different features of the standard were also done. Five QCIF video sequences were coded and their resultant quality was measured by its bit rate/dB. One at a time, several features of the H.264/AVC Motion Estimation were activated like different block-matching algorithms, different number of reference frames and, finally, the sub-pixel refinement (half-pixel and quarter-pixel accurate motion vectors).

The best results were achieved when the sub-pixel refinement feature was activated. It is important to notice that fast search algorithms achieved the best performance without expressive losses of quality, especially when the sub-pixel refinement was used.

This way, our design focuses in an Interpolation Unit fast enough to not degrade the performance of the complete ME architecture.

5. Proposed Processing Unit Architecture

For a better hardware implementation, it was necessary to optimize the 6-tap FIR filter by grouping similar operations and decomposing multiplications into shift-adds in order to get a lower delay path and lower hardware cost. The Fig. 2 shows the proposed Processing Unit (PU).

The equation (5) is the first step of the processing unit, it calculates the y_1 value of a half-pixel. For V Type half-pixels, the y_1 value must be stored for later use in D Type interpolation.

$$y_1 = (A+F)+4\left[4(C+D)-(B+E)\right]+4(C+D)-(B+E) \quad (5)$$

The second step of the processing unit applies (2) if the control unit indicates the interpolation of V or H Type or (4) if it indicates the interpolation of D Type.

From (5), a combinational circuit was designed. However it achieved a low frequency and we had to use a three-stage pipeline to make our circuit suitable for high definition video coding. Deeper pipeline configurations are also possible because a high throughput is more important than a low latency for the interpolation process.

Nine PUs were used in a module called Filter Line. It is able to interpolate an entire line of H Type half-pixels, a column of V Type and a line of D Type in a single step.

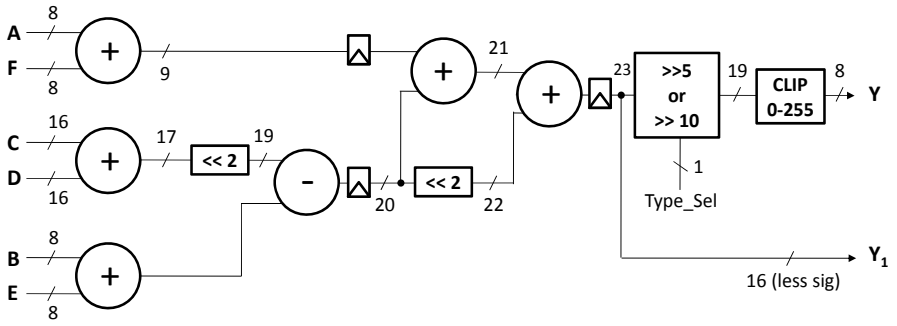


Fig. 2 – Processing Unit Architecture.

6. Data Flow and Organization

Five buffers were used to store and shift the integer position samples, V Type samples, V_1 Type values, H Type samples and D Type samples. All buffers are connected to the Filter Line.

The buffer for integer position samples stores a 14x14 block (an 8x8 block plus the three-pixel border). This buffer has two outputs: an entire line used for H Type interpolation and an entire column used for V Type interpolation. It is able to shift its lines and columns in order to change its outputs.

The buffer for V Type half-pixels stores a 14x9 area (two blocks in an 8x9 area plus a vertical three-pixel border). The buffer for V_1 Type values stores a 12x9 area, the y_1 values for each V Type half-pixel used in D Type interpolation. Both buffers have an entire line as output and are able to shift it for D Type interpolation.

The buffer for H Type half-pixels stores a 9x8 area (two 8x8 blocks). This buffer for D Type half-pixels stores a 9x9 area, four 8x8 blocks. Both buffers are used for storage purposes only.

The control unit selects which buffer will store the Filter Line output. It takes 34 clock cycles to interpolate a half-pixel search area using an 8x8 block.

7. Synthesis Results and Related Works

Our design was described in VHDL and synthesized to the Xilinx Virtex4 XC4VLX15 and Xilinx Virtex2 XC2V80 devices using the Xilinx ISE 10.1 synthesis tool [4].

The tab.1 shows the Processing Unit performance for each tested pipeline configuration and tab.2 shows the throughput of our complete design. Our interpolation unit can be used to process very high resolutions like QHDTV in real time (30 frames per second or faster).

Since the memory hierarchy of the project is not defined yet, our buffers are mapped as register banks. The tab.3 shows the cost of these buffers. Our interpolation process uses less than 1 kilobyte.

The tab.4 shows the use of FPGA resources. It is important to note that the cost of our Control Unit and Filter Line is very low when compared to a complex Motion Estimation module.

Tab. 1 – Processing Unit Results

Solution	Frequency	Latency	Critical Path
No Pipeline	69 MHz	0 cycles	14.4 ns
3-Stage Pipeline	153 MHz	2 cycles	6.5 ns

Device: Xilinx Virtex4 XC4VLX15

Tab. 2 – Interpolation Unit Throughput

Frequency (MHz)	Throughput (Mblocks/s)	HD 1280x720 (frames/s)	Full HD 1920x1080 (frames/s)	QHDTV 3840x2048 (frames/s)
141	4.156	288	128	33

Device: Xilinx Virtex4 XC4VLX15

Tab. 3 – Buffers Results (use of hardware elements)

Buffer	8-bit Registers	16-bit Registers	8-bit 4:1 MUX	8-bit 2:1 MUX	16-bit 2:1 MUX
Integer Positions	196	-	196	-	-
V Type	126	-	126	-	-
H Type	72	-	-	72	-
D Type	81	-	-	81	-
V_1 Type	-	108	-	-	108

Tab. 4 – Interpolation Unit Results (use of the FPGA resources)

Module	Slice	Slice Flip-Flops	4-input LUTs
Control Unit	16	6	27
Filter Line (9 PUs)	1386	486	2610

Device: Xilinx Virtex4 XC4VLX15

Works focusing a high performance Interpolation Unit were not found in the literature. The Interpolation Unit used in [5] gives support to variable block sizes by splitting those blocks into smaller 4x4, 4x8 or 8x4 blocks and then interpolating these blocks. It needs 25 clock cycles to interpolate an 8x4 block. Then, it needs 50 clock cycles to interpolate an 8x8 block. The tab.5 shows that our design needs 32% less clock cycles to interpolate an 8x8 block. Also, when synthesized to the same FPGA device, our design achieves a 57% higher processing rate.

Tab. 5 – Comparison with Related Work

	Yalcin 2006 [5]	This Work
Cycles to process a 8x8 block	50	34
Frequency (MHz)	85	91
Throughput (Mblocks/s)	1700	2670

Device: Xilinx Virtex2 XC2V80

8. Conclusions and Future Works

This work presented a Half-Pixel Interpolation Unit for the H.264/AVC video coding standard. This architecture was designed to be part of a Sub-Pixel Accurate Motion and achieved a maximum frequency of 141 MHz when synthesized to a Xilinx Virtex4 FPGA device and it is able to process over 4 million 8x8 blocks per second. This high throughput makes this architecture able to process very high definition videos as QHDTV (3840x2048) in real time.

As future works we plan to design and add the Half-Pixel Refinement Search to our design and then integrate it to a Motion Estimation module that uses the Diamond Search algorithm.

9. References

[1] I. G. Richardson, H.264 and MPEG-4 Video Compression, Wiley, 2003.

[2] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003.

[3] JM17. “H.264/AVC JM Reference Software.” Mar, 2010; http://iphome.hhi.de/suehring/tml

[4] Xilinx. “FPGA and CPLD Solutions from Xilinx, Inc.” Mar, 2010; www.xilinx.com

[5] S. Yalcin, I. Hamzaoglu, “A High Performance Hardware Architecture for Half-Pixel Accurate H.264 Motion Estimation”, *14th Int. Conf. on VLSI-SoC*, October, 2006.

A Low Cost Real Time Motion Estimation/Compensation Architecture for the H.264/AVC Video Coding Standard

Robson Dornelles, Luciano Agostini
rdornelles.ifm@ufpel.edu.br, agostini@ufpel.edu.br

Federal University of Pelotas

Abstract

This work presents a Motion Estimation/Compensation architecture for the H.264/AVC standard, able to process high definition videos in real time (30 frames per second) with reduced hardware resources usage. Also, when the Motion Compensation is performed together with the Motion Estimation, there is a reduction in the number of access to the coder main memory. This reduction speeds up the entire coding process, and also simplifies the coder project, since the Motion Estimation/Compensation architecture can be more easily integrated than two separated architectures. The designed solution was described in VHDL and synthesized to the TSMC 0.18 Standard Cell library, using a total of 51K gates, with a operation frequency of 280.2 MHz, which is enough to process HDTV videos (1920x1080 pixels) in real time (30 frames per second).

1. Introduction

In the video coding process, one of the most important techniques is the Inter Frame Prediction. Since digital videos have a high correlation between its frames, it is possible to explore this temporal redundancy by coding only the differences between the frames. In the H.264/AVC [1] standard, this task is done by the Inter Frame Prediction.

In the Inter Frame Prediction, a frame is coded using the already coded frames as references. The process starts by dividing the current frame in macroblocks (16x16 pixels), and then in smaller blocks. For each block, an equivalent block is found (predicted), using a search algorithm on the previous frames. The search algorithm will raise candidate blocks, and the best candidate block must be chosen, using a distortion metric. The most common distortion metric is the SAD (Sum of Absolute Differences), presented in (1).

$$SAD = \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} \left| CurrentBlock(i,j) - CandidateBlock(i,j) \right| \qquad (1)$$

In Eq. (1), *w* is the width and *h* is the height of both the candidate and the current block. The candidate block that presents the lowest SAD value is the best block to represent the current block. The position (x,y) of the best candidate block and the frame where it was found is the *motion vector*. The Motion Estimation is the module that searches for candidate blocks and generates the motion vectors.

The Motion Compensation module takes as input the motions vectors produced by the Motion Estimation, and retrieves the candidate blocks pointed by those vectors in the reference frame. The residual information that results from the subtraction of the current block and its best candidate block must be transmitted in the bitstream along with the motion vector that points to that candidate block. In the decoder, another Motion Compensation recovers the candidate block using the motion vector, and the current block can be reconstructed by a sum between the candidate block and the residual information.

This work presents a hardware design that integrates the Motion Estimation and the Motion Compensation, storing the residual values during the Motion Estimation process, in opposition to the traditional Motion Compensation, which is performed after the Motion Estimation.

This paper is organized as follow: the Section 2 presents the designed architecture; Section 3 shows the synthesis results and the comparison among related works; Section 4 concludes this work.

2. Designed Architecture

The designed architecture performs the Motion Estimation using the FullSearch algorithm for 4x4 blocks [2] in a search area of 19x19 samples, which means that each block in the current frame will be tested against 256 candidate blocks. The main modules of the architecture are the *Cyclic Register Bank* (**CRB**), which holds the reference samples and is connected to four *Processing Units* (**PU**) that are responsible to calculate the SAD of the candidate blocks. Connected to the PUs are the *Integrated Motion Compensation* (**IMC**) and the *Comparator*, and they are also connected to each other. The Comparator chooses the motion vector that points to the best candidate block (the one with the smallest SAD) and tells to the MC which residue is equivalent to that motion vector. Fig. 1 shows the block diagram of the designed architecture

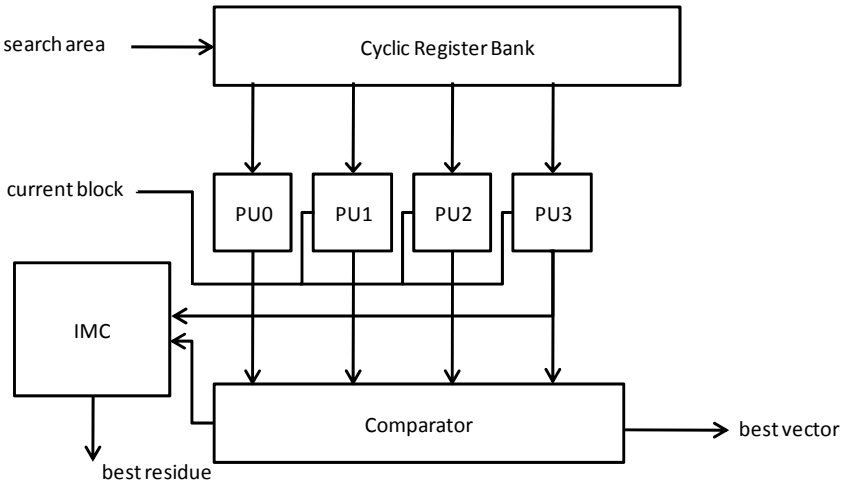


Fig. 1 - Block diagram of the design architecture.

2.1. Processing Unit (PU)

The PU takes 16 samples from the current block plus 16 samples from the candidate blocks as input per cycle. The SAD value between those blocks is evaluated in 5 clock cycles, in a pipeline approach. The first pipeline stage performs the subtraction and absolute operations. The other four stages are a pipelined tree-based sum between all values that were processed in the first stage. The output is the SAD value between two blocks, which will be the input of the Comparator, where the smallest SAD must be chosen.

The results of the subtractions performed in the first pipeline stage are the residual information that must be recovered by the traditional Motion Compensation if the input candidate block is chosen at the end of the Motion Estimation process. Thus, this residual information must also be an output of the PU, because the Integrated Motion Compensation must take this information as input.

2.2. Cyclic Register Bank (CRB)

This module is composed by four lines, each line composed by 19 8-bit registers, to be able to store a 19x4 area of the search area. That area has 16 candidate blocks. Connected in the CRB are four PUs, as presented in fig. 2.

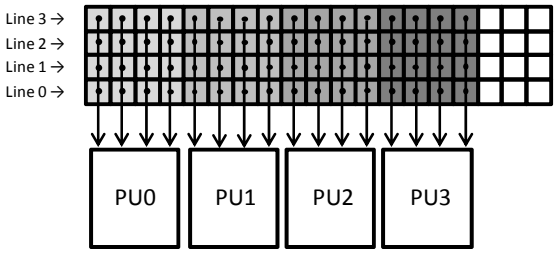


Fig. 2 - Cyclic Register Bank and the Processing Units

This CRB is controlled by a *finite state machine* (FSM). There is a *initial charge* state, that lasts 4 cycles and, in each cycle, line i receives the samples of the line$i-1$, $i \in \{3,2,1\}$, and the line 0 receives the new input. When the CRB is entirely filled, each PU has an entire 4x4 candidate block in its input, and the *rolling* state begins. It lasts 3 cycles, and in each cycle there is a cyclic-left-shift at register level, that changes the 4x4 candidate block in the PUs input. Then, the *new charge* states begin, when another reference line must enter in the CRB. This is done as the line i receives the samples of the line $i-1$, $i \in \{3,2,1\}$, and the line 0 receives the new input. A *restart_crb* signal must be send to the fsm to when the Motion Estimation for one block is done, so the CRB can be filled with the new search area for the next block.

2.3. Movement Coordinate Generator

Before a SAD value enters the Comparator, the *(x,y)* coordinate of the candidate block that resulted in that SAD value must be concatenated in the SAD value itself. A simple arithmetic circuit can be design to perform this task. The *x* coordinate starts in *-8* and is incremented at each *new charge* state, and its last value must be *7*. The *y* coordinate starts as *-8* to the PU0, *-4* to the PU1, *0* to the PU2 and *4* to the PU3. Each cycle that the CRB FSM stays in the *rolling* state must increment the *y* coordinate, as presented in tab. 1.

Tab. 1 - Values for the *y* coordinate.

CRB FSM State	PU0	PU1	PU2	PU3
Initial or New Charge	1000	1100	0000	0100
Rolling 0	1001	1101	0001	0101
Rolling 1	1010	1110	0010	0110
Rolling 2	1011	1111	0101	0111

2.4. Comparator

The Comparator module takes four inputs per cycle: each input is a motion vector concatenated with the value of the SAD generated by the block pointed by that vector. The Comparator will decide what input has the smallest SAD value, and will store this value as a partial result, to compare with the smallest SAD of the next four inputs, and so on. This is done in a 3-stages pipeline. In the end of the Motion Estimation of a block, the Comparator output is the vector that points to the block that presents the smallest SAD. Fig. 3 shows the RTL scheme of the Comparator.

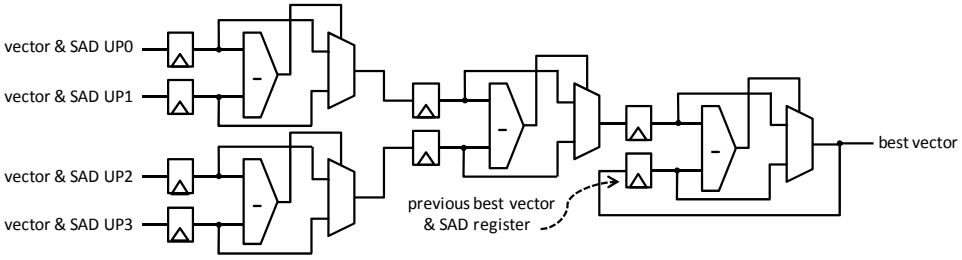


Fig. 3 - RTL scheme of the Comparator.

2.5. Integrated Motion Compensation (IMC)

The IMC receives the residue that comes from the PUs and stores those residues in four buffers (each PU has its own buffer). Those buffers are "shift-registers", where each "register" is composed by sixteen 8-bit registers. When the Comparator decides which one is the first best vector, the IMC must store the residue equivalent to that vector. This is done by connecting the buffers last stage to a multiplexer. The logic for the multiplexer selector is based in the two most significant bits of the best vector's *y* coordinate. Tab. 1 shows that those two bits never change for each PU. Thus, when the first choice is made, the best residue must be stored in a "best residue register", because the next best vector can still be the previous best vector. So, the mux selector is a concatenation between a signal that tells if the best vector changed ('1' if has not changed, '0' in the otherwise), with the PU number, that can be decoded using the two most significant bits of best vector's *y* coordinate. Fig. 4 presents the IMC architecture.

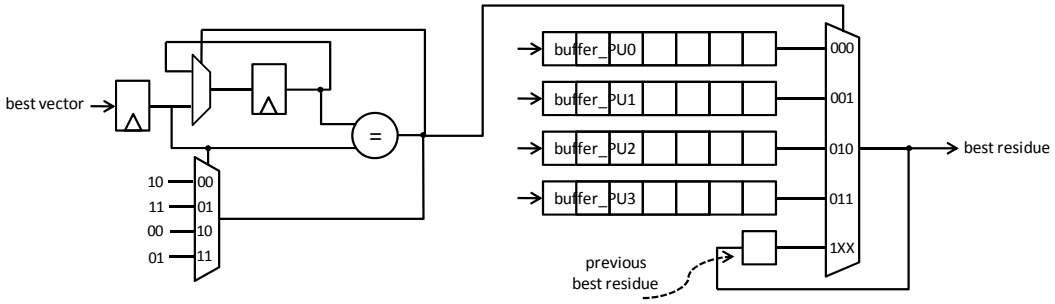


Fig. 4 - IMC architecture.

3. Synthesis Results and Comparison with Related Works

The designed architecture was described in VHDL and synthesized to the TSMC 0.18 µm Standard Cell Library, using the Leonardo Spectrum synthesis tool [3], resulting in a hardware resources usage of 51K gates and achieving an frequency operation of 280.2 MHz Since the architecture takes 4 cycles to process each 19x4 range of the search area, and there are 16 of those ranges, the total number of cycles needed to process one block is of 16*4=64, plus 4 cycles in the *initial charge* state of the CRB, the total number of cycles needed to process a 4x4 block is of 68. In a 1080HD video, there are 129600 4x4 blocks, times 68 cycles per block times 30 frames per second equals 264.28 MHz needed to process 1080HD videos at 30 frames per second. Since the architecture achieves a frequency of 280 MHz, it can process high definition videos in real time.

We have not found works in the literature that implement a Motion Estimation architecture with integrated Motion Compensation. This way, the comparison was made with architectures that implement FullSearch Motion Estimation with variable block size for the H.264/AVC standard. Among all compared works, ours present the best relation between the throughput and the cost. This relation, frames per number of gates, means how much HDTV frames are coded with one gate, so a higher value represents a best usage of the hardware resources. Also, our work is the second in search area size and low-cost. Tab. 2 shows the synthesis results of our work and the comparison with related works.

Tab. 2 - Synthesis results and comparison with related works.

Solution	# of Gates	Search Area	Technology	Frequency (MHz)	1080HD Frames/s	1080HD Frames/s / # of Gates
This Work	51K	19x19	TSMC 0.18 µm	280	31	0,00060
Kim [4]	39K	16x16	DonbuAnam 0.18 µm	416	12	0,00030
Porto [5]	113K	16x16	TSMC 0.18 µm	296	34	0,00030
Yap [6]	61K	16x16	TSMC 0.13 µm	294	8	0,00013
Ou [7]	597K	16x16	UMC 0.18 µm	123	60	0,00010
Liu [8]	486K	192x128	TSMC 0.18 µm	200	30	0,00006

4. Conclusion

This work presented the design of a hardware architecture that performs the Motion Estimation and Motion Compensation for 4x4 blocks using the FullSearch algorithm. The presented architecture efficiently uses a cyclic register bank to reduce the number of memory requisitions needed to raise all candidate blocks using the FullSearch algorithm in a 19x19 search area. Also, the architecture presents an efficient way to manage the residual information produced in the SAD calculation, in such a way that the Motion Compensation can be performed in parallel with the Motion Estimation.

The architecture was synthesized to the TSMC 0.18 µm standard cell technology, and resulted in a hardware resources usage of about 51K gates, a small number when compared to related works in the literature. In a relation between the throughput (in 1080HD frames per second) and the architecture cost (In hundred of standard cell gates), the designed architecture presented the best results, which means that the hardware resources are efficiently used to achieve high throughput.

An ongoing work is about the addition of bottom-up decision modes for the VBSME of the H.264/AVC standard to the architecture. Also, a solution for the motion vector predictor architecture is being investigated.

5. References

[1] International Telecommunication Union, ITU-T Recommendation H.264 (05/03): Advanced Video Coding for Generic Audiovisual Services, 2003.

[2] P. Kuhn, Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Kluwer Academic Publishers, Boston. 1999.

[3] Mentor Graphics, "LeonardoSpectrum", Feb. 2009; http://www.mentor.com/products/fpga_pld/synthesis/leonardo_spectrum.

[4] 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.

[5] R. Porto, L.Agostini, and S.Bampi. "H.264/AVC Variable Block Size Motion Estimation for Real-Time 1080HD Video Encoding". *Proc. 24th South Symposium on Microelectronics* (SIM'09). 2009.

[6] 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.

[7] 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.

[8] 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.

Design Automation Tools 2

Improvements in the Detection of False Path by using Unateness and Satisfiability

Felipe Marques, Osvaldo Martinello Jr, Renato Ribas, André Reis
{felipem, omjunior, rpribas, andreis}@inf.ufrgs.br

Instituto de Informática - Universidade Federal do Rio Grande do Sul - Brazil

Abstract

This paper presents improvements on a previous methodology for false path detection using satisfiability. As the previous methodology, which is referred as node-sat, it is still based on circuit node properties that are related to non-testable stuck-at faults as well as to false path detection. When compared to traditional satisfiability methods that generate sets of clauses associated to paths, the node-sat can be more efficient. This efficiency derives from the fact that most digital circuits have a number of nodes that is smaller than the number of paths, and therefore, a smaller number of satisfiability instances needs to be solved. This number can be reduced even more by considering the circuit topology. Experiments comparing the previous node-sat methodology and the new techniques presented on this paper reveal that the time consumption can be reduced by 60% on average.

1. Introduction

Timing analysis is an essential procedure in digital circuit design flow. It is heavily invoked in the inner loop of a performance driven optimization methodology. Each timing analysis can generate thousands of critical paths with thousands of critical paths being false paths. However, when false paths are dealt in timing analysis, a large number of constraints need to be created and propagated, leading to loss of efficiency. Therefore, it is very convenient to detect false paths in order to prevent project time waste.

Satisfiability (SAT) [1] is being very efficient on solving many combinatorial algorithmic problems. It has been used for test vector generation [2] as well as for false path detection [3]. Usually, false path detection using satisfiability is performed path by path. This means that in order to determine if a path is functionally sensitizable, it is necessary to solve a satisfiability instance. As circuits have many paths, due to path reconvergence, the cost of using satisfiability to detect false paths is increased.

Alternatively, the method introduced in [4], which is referred as node-sat, detects false paths solving satisfiability instances that are associated to unateness properties of circuit nodes, instead of being associated to paths. Since most circuits have a number of nodes that is smaller than the number of paths, this method can be more efficient. A circuit node may be positive or negative unate with respect to each of the variables on which it depends (primary inputs). When the node is both positive and negative unate in a given variable, it is said that the node is binate (or mixed) on that variable. As demonstrated in [4], it may be very easy and fast to prove that a node is binate through logic simulation. However, in order to prove that a node is not binate, it is necessary to solve a satisfiability instance. In the worst-case scenario, the number of satisfiability instances to be solved is equivalent to the number of paths of a circuit. It can be very time consuming. This paper presents some pruning techniques that can significantly reduce the number of needed satisfiability instances, resulting in major time savings.

This paper is organized as follows. Section 2 discusses false paths and related concepts. The previous work on node-sat is briefly reviewed in section 3. The pruning techniques to reduce the number of satisfiability instances are presented in section 4. Results will be shown in section 5, while conclusions are outlined in section 6.

2. Preliminaries

False paths can be introduced in a digital circuit design through different processes. Some logic synthesis process, in-place optimizations or unreachable states of finite state machines can establish false paths in a circuit. Consider a signal *s*, which is connected to a gate *G*. It is considered to dominate *G* if the stable value and the stable time at *G* are determined by those at *s*. This way, a path is considered to be sensitized, under a certain delay configuration, by a vector pair if each on-input of the path dominates its connected gate. Given a delay configuration, a path is a true (sensitizable) path if there exists at least one vector pair which sensitizes the path. Otherwise, it is a false path.

Under a delay configuration, a vector pair sensitizes a path iff each on-input of the path is either the earliest controlling value or the latest non-controlling input with all its side-inputs being non-controlling inputs. This criterion is called the exact sensitization criterion [5]. To efficiently check the sensitizability of target paths, a delay-independent method, using the functional sensitization criterion, is proposed in [6]. If there exists an input vector *v* such that all the side-inputs of s_i along *P* settle to non-controlling values on *v* when the on-input s_i has a non-controlling value, then *P* is functionally sensitizable. Otherwise, *P* is functionally unsensitizable.

Because functional sensitization, a delay independent criterion, is only a necessary condition of exact sensitization, an identified functional sensitizable path might not be sensitizable under certain delay configurations. On the other hand, the identified functional unsensitizable paths must be false under any arbitrary delay configuration.

Figure 1 illustrates a combinational circuit with the mutual exclusion problem. The path going from input *a* to output *o* is functional unsensitizable. It is not possible to set all side-inputs to non-controlling values. The input *c* needs to assume both Boolean values *0* and *1* at the same time. The false path can be removed applying logic duplication all over the path. It results in the circuit of Figure 2. Area optimization algorithms could find the circuit of Figure 1 as the best solution for area saving. However, it introduces a false path in the circuit.

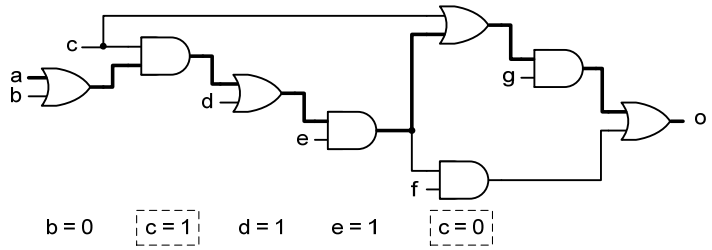


Fig. 1 - Mutual exclusion problem.

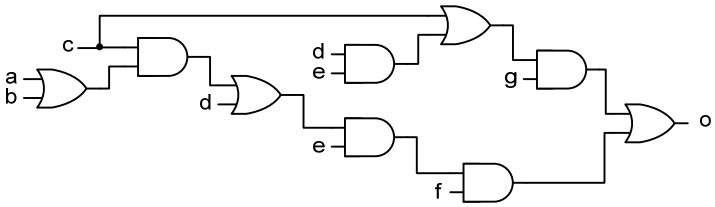


Fig. 2 - False path vs. logic duplication.

A typical design flow of digital circuits optimizes a design through an iterative process. Usually, it involves a large number of static timing analysis (STA). The STA process verifies if there exist paths in a circuit with delays that exceed a given timing constraint. Such paths are known as critical paths. When such a critical path is identified, all elements associated with the critical path may need to be modified to achieve the required timing. Therefore, each time the design is optimized via synthesis, place and route, in-place optimization and so one, static timing analysis can be performed.

Each timing analysis can generate thousands of critical paths with thousands of critical paths being false paths. This way, false paths may be analyzed and constraints can be created for false paths, and fed back it to the design tools. Generation of false path constraints for a typical design leads to project time waste and is very error prone.

There are different sensitization criteria and methodologies for false path detection. The sensitization criteria can be divided into three categories: static sensitization, dynamic sensitization and floating mode/viability sensitization. Most of all methodologies work at the gate level. More recent approaches are bringing back methods working at the register transfer level (RTL). The methodology presented in this paper works at the gate level, using a static sensitization model. Functionally unsensitizable paths are identified checking the unateness properties of each gate of a circuit.

3. The Detection of False Paths by using Unateness and Satisfiability

In a previous work [4] we proposed a method that is able to detect false paths in combinational circuits. It computes the unateness properties of each node of a circuit in two ways: *topological unateness* and *functional unateness*. Comparisons among topological and functional unateness can be used to detect false paths.

Consider the combinational circuit of Figure 3. Tables 1 and 2 show the topological and the functional unateness properties for all nodes of the circuit, respectively. It uses a binary code of 2 bits per variable representing positive and negative unateness. The topological unateness can be computed in linear time using dynamic programming. The binary code of a node *n* is the result of a bitwise OR operation between the binary codes of the gate inputs. Accordingly to the definition 6 (from [4]), if *n* is the output of a negative gate, the generated binary code has to be complemented. The functional unateness is individually calculated for all nodes. It has been demonstrated that simulation can accelerate this process. However, in the worst-case scenario, several satisfiability instances will need to be solved. There is only one difference among the tables: the node *f* is binate in all variables of the topological unateness while it is positive unate in all variables of the functional unateness. This means that any negative path from the inputs *a* or *b* to the output *f* is a false path.

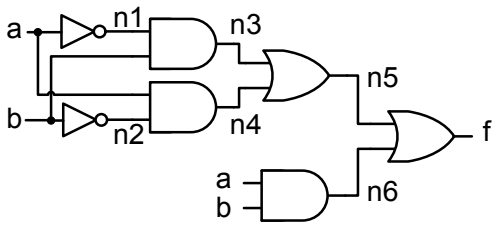


Fig. 3 – Redundant Combinational Circuit.

Tab.1 – Topological Unateness

Node	a+	a-	b+	b-
n1	0	1	0	0
n2	0	0	0	1
n3	0	1	1	0
n4	1	0	0	1
n5	1	1	1	1
n6	1	0	1	0
f	1	1	1	1

Tab. 2 – Functional Unateness

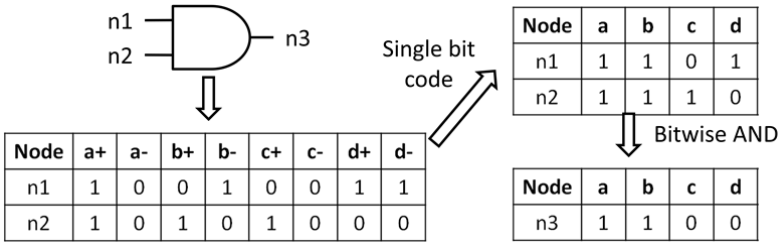
Node	a+	a-	b+	b-
n1	0	1	0	0
n2	0	0	0	1
n3	0	1	1	0
n4	1	0	0	1
n5	1	1	1	1
n6	1	0	1	0
f	1	0	1	0

4. Pruning Techniques for Node-Sat

The node-sat approach identifies false paths using the unateness properties of each circuit node, rather than paths. Although the number of nodes is far smaller than the number of paths, in some cases, it is not possible to analyze a circuit in a short period of time. Simulation can be used to accelerate the analysis. However, it depends on the choice of input vectors, which is a random procedure. When it does not succeed, satisfiability instances have to be solved to prove the unateness properties of a given node. In this case, SAT clauses are generated to represent a sub-circuit going from the primary inputs to the node that should be tested.

There are other techniques that can be used to accelerate the analysis process. Some sub-circuits have special properties and the method does not need to prove the functional unateness of a node neither by simulations nor solving SAT instances. In this case, the topological unateness is enough to ensure the unateness condition of each primary input related to the node. When the topological unateness determines that a node *n* is not positive (negative) unate with respect of a given variable *v*, it means that there are no positive paths going from the *v* to *n*. Therefore, there is no need to compute the functional unateness for the input. It can be directly coded as zero. This strategy has been used in the previous approach [4]. Nevertheless, there is still a considerable amount of satisfiability instances to be solved.

This paper presents another possible pruning technique. It considers the path reconvergence to identify nodes that do not need functional unateness computation through simulation or satisfiability. When there is no path reconvergence from a given input to a given node, it is impossible to have differences among the functional and topological unateness. This way, the functional unateness assumes the same values of the topological unateness. The main point of this pruning technique is to identify the path reconvergence. Searches over the circuit graph on each iteration of the functional unateness computation would not be practical. A simple mechanism can extract this information from the topological unateness table. This process is depicted in Figure 4. The first step converts the 2-bit binary code to a single bit binary code. It is done through an OR operation between the positive and negative bits of each variable. The second step performs a bitwise AND between the single bit code of the gate inputs. The resultant binary code indicates when there is path reconvergence from the primary inputs to a given node. The binary code of the node *n3* has at least one input set as '1' that indicate reconvergence, and the functional unateness has to be calculated for all inputs. When the code is fulfilled with zeros, it indicates that the node does not generate any reconvergence.



Node	a+	a-	b+	b-	c+	c-	d+	d-
n1	1	0	0	1	0	0	1	1
n2	1	0	1	0	1	0	0	0

Node	a	b	c	d
n1	1	1	0	1
n2	1	1	1	0

Node	a	b	c	d
n3	1	1	0	0

Fig. 4 – Identifying Path Reconvergence.

5. Results

Table 3 shows a comparison among the previous and the new node-sat methodologies. No simulation was used to accelerate the functional unateness computation. It is clear that the proposed pruning techniques are very effective. Since the number of satisfiability instances was reduced, the time consumption was significantly reduced.

Tab.3 – Comparisons among the previous and the new node-sat methodologies

	Node-sat [4]		Node-sat using pruning techniques		
Circuit	**# SATs**	**Time (s)**	**# SATs**	**Time (s)**	**Improvement (%)**
c1355	15196	1027,464	13756	907,24	11,70
c1908	12846	208,727	11424	193,414	7,34
c2670	12508	283,212	8292	191,563	32,36
c3540	24850	1437,163	21056	1374,293	4,37
c432	7010	78,550	6130	70,895	9,75
c499	14986	1023,630	13546	1005,787	1,74
c880	8006	91,464	4846	57,094	37,58
i10	71862	3121,834	49300	2244,290	28,11
i4	2752	34,467	1096	14,115	59,05
i5	2636	27,489	160	1,692	93,84
i6	3360	43,586	1258	15,937	63,44
i7	3958	62,103	1798	26,540	57,26
i8	18998	355,798	14282	270,297	24,03
i9	11554	157,171	8460	117,686	25,12
s208	572	5,030	206	1,630	67,59
s27	54	0,458	12	0,109	76,20
s298	638	5,436	276	2,228	59,01
s349	1000	8,433	570	5,154	38,88
s382	1200	9,964	488	4,043	59,42
s386	1192	9,549	298	2,419	74,67
s400	1292	10,664	530	4,697	55,95
s420	1906	15,769	768	7,228	54,16
s444	1296	10,640	534	5,142	51,67
s510	2060	16,826	940	8,555	49,16
s526	1340	10,944	436	3,725	65,96

6. Conclusions

This paper presented pruning techniques to improve the detection of false path by using unateness and satisfiability. The pruning techniques are very effective on reducing the number of satisfiability instances that need to be solved to compute the functional unateness. This leads to major time savings. The information of the differences among topological and functional unateness may be used on a variety of algorithms and methods that need information about signals observability such as static timing analysis.

7. Acknowledgements

Research partially funded by Nangate Inc. under a Nangate/UFRGS research agreement, by CNPq and CAPES Brazilian funding agencies, and by the European Community's Seventh Framework Programme under grant 248538-Synaptic.

8. References

[1] Marques-Silva, J.P., and Sakallah, K.A., "Boolean Satisfiability in Electronic Design Automation", *Design Automation Conference*, 2000, pp. 675-680.

[2] Larrabee, T., Test pattern generation using Boolean satisfiability, *IEEE Transactions on CAD*, Vol. 11, Issue: 1, January 1992, pp. 4 – 15.

[3] Ringe, M., Lindenkreuz, T., and Barke, E., "Path verification using Boolean satisfiability", *Design, Automation and Test in Europe*, Feb. 1998, pp. 965 – 966.

[4] Marques, F.; Ribas, R.; Sachin, S. and Reis, A. "A new approach to the use of satisfiability in false path detection". Proceedings of the 15th ACM Great Lakes symposium on VLSI, GLSVLSI'05, pp. 308 – 311, 2005.

[5] Chen, H.-C; Du, D. H.-C; and Liu, L.-R. "Critical path selection for performance optimization", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 185–195, Feb. 1993.

[6] Cheng K.-T. and Chen, H.-C. "Classification and identification of nonrobust untestable path delay faults", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 8, pp.845–853, Aug. 1996.

A Case Study about Variability Impact in a Set of Basic Blocks Designed to Regular Layouts

Jerson Guex, Cristina Meinhardt, Ricardo Reis
{jpguex, cmeinhardt, reis}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul (UFRGS)
Instituto de Informática – PPGC/PGMicro
Av. Bento Gonçalves 9500 Porto Alegre, RS - Brazil

Abstract

The new submicron technologies are more susceptible to the effects of process variability, making these effects very important to the designer in the previous steps of physical synthesis. To deal with this problem, many authors suggest the adoption of regular methodologies for layout. In this work we investigate the effects of process variability in a set of basic cells designed to regular layouts. Our methodology adopted a reduced set of basic blocks, implement two options of layout for each cell: one with only basic cells NAND, NOR and Inverters and other with complex cells layout. These two descriptions were simulated with variability in the main parameters and without variability. The results for timing and power are compared with the goal find witch implementation is less susceptible to process variability effects. The conclusions indicate that this set of basic cells have suffered 13% less impact in timing parameters observed.

1. Introduction

The circuits' fabrications in sub-micron technologies are changing the physical design of circuits requiring complex layout rules and requiring the adoption of a wide range of safe design of the blocks to handle the increased process variability. The large reduction in scale of transistors size introduced new variation sources and makes the control of variability more and more complex. The variability already affects the generation of circuits in the technologies of 180 nm, according to [1], process variability will increase and will decrease the predictability of performance of nano circuits in future technologies, which directly affects the operation of the circuit. For example, the leakage current has exponential dependence on the size of gate, so the variance in L has exponential impact on leakage current.

Above all, the demands for performance, power consumption, cost, manufacturability and time-to-market, make the designers use some methodologies like as design for performance - DFP or design for manufacturability – DFM to guarantee the project success. Some researchers say the need for a bidirectional path between production designers, CAD tools and manufacturing, where the cost and value are the main guides, especially when built in sub-micron technologies [1]. In DFMs, the projects use techniques to increase the yield of the project, such as analysis of DFM rules, physical tests, Litho-friendly design and Nanometer Silicon Modeling [3].

The adoption of regular structures is an alternative to dealing with problems of variability in process steps, as well as problems related to the sub wavelength lithography and interconnections [2]. The work done is to find solutions for integrated circuits physical synthesis less susceptible to the effects of variability arising from the use of nano technologies manufacturing. The aim of this paper is to analyze the effects of variability on a range of cells taken to compose the basic blocks within a physical synthesis where the generation of regular layouts is adopted [4]. This tool [4], generate layouts where the geometric regularity is exploited by the repetition of basic patterns of layout along an array. Regularity is identified as one of the best alternatives for dealing with current problems of manufacturing in sub-micron technologies. Regular projects are less susceptible to problems of lithography, increase yield and reduce the time spent on re-design. In addition, regular layouts have greater predictability of results of power, delay and yield, mainly because the cells are pre-characterized.

This tool [4] aims working with two types of physical synthesis for regular layouts, producing integrated customizable circuits by all layers and customizable circuits for few layers. The main objective of this tool is the ease of conversion and adaptation depending on the matrix approach chosen. This will facilitate comparison between different alternatives matrix, the adoption of logical blocks and various new technologies [4]. The following sections present concepts adopted for the generation of regular layouts and the criteria adopted in choosing the basic blocks studied.

2. Regular Layout Generator

The basic architecture of Structured ASICs is composed of two parts: the basic block, also called the basic

unit or element, and the array of basic blocks [5]. This matrix is prefabricated (also known as array of elements or sea of components). The basic block contains a small amount of logic, implementing elements such as basic gates, multiplexers and LUTs. In a single chip, there may be one or more types of elements forming the Structured ASIC. Each element is designed for different purposes, for example, can be adopted one type of element for combinational logic and other logic for sequential.

As Gu [6] the granularity of a project is defined by the smallest logic function developed by smaller logical component. Depending on the granularity of the architecture, these logic blocks may contain registers and RAMs [7]. The granularity of a component can be thick, large, medium, small, or thin, depending on the size of the complex logic contained in the component.

The choice of component to be used in the project directly affects issues of routing and area. Projects with low granularity require more components to perform the same function that projects with high granularity. The low granularity also implies a greater number of pins to connect to the router. However, projects with great granularity can result in wasted area inside the component, i.e., greater logical effort is necessary to use all the logic functions provided by components in architectures with great granularity [8]. Structured ASIC designs adopt different degrees of granularity, depending on the purpose and flow of synthesis prioritized by the manufacturer.

The physical synthesis adopts a flow toward a generator of regular arrays composed of basic cells read in cell libraries. The tool developed aims to work with two types of physical synthesis for regular layouts, producing integrated circuits customizable by all masks or circuits customizable by few layers. This will facilitate comparison between different alternatives matrix, the adoption of logical blocks and various new technologies. The regular matrix generator explores the geometric regularity.

The geometric regularity is achieved by repetition of patterns in the layout. Fig. 1 shows the general architecture of these arrays. The matrix model used is the uniform. Each cell occupies a position in the array. All cells are aligned vertically and horizontally. The cells used adopt simple layouts, with short connections and prioritizing the use of straight lines of polysilicon.

In the synthesis of circuits customizable by a few layers, all positions of the array are determined in advance. The array can work with a single basic block, a set of building blocks or basic blocks configurable. The type of basic block used will depend on the logical design of the circuit. The basic blocks used are stored in libraries of cells. The adoption of cell library enables rapid change of the layout of blocks and technology migration. The restrictions of the matrix and the cells used are:

1. All cells must have the same height.
2. All cells must have the input pins and output aligned with the grid routing.
3. Width adopted for each position in the array will be the largest width among the cells used.

The design of customizable arrays by all layers generates regular circuit, where each basic cell has a position lined up horizontally and vertically with neighboring cells, forming an array of cells, as shown in Fig 1. The position of each cell is determined in placement step. In addition, spaces are inserted to make easy the step of routing. These spaces can be of two types: dummies cells or extra tracks. Dummies cells are empty cells in the matrix, with size equal to that occupied by the cells. Extra tracks are inserted into vertical tracks in regions where the routing is congested. These tracks have a width corresponding to a step in the routing grid, in order to maintain the alignment of cells and pins with the routing grid. In Fig. 1 are shown three basic cell types: INVERTERS, NANDS and NORs. However, other cells can be inserted in the matrix.

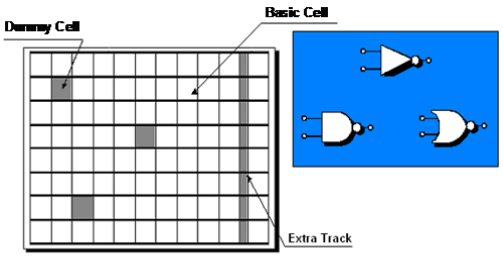


Fig. 1 - Example of Regular Matrix Architecture

3. Experiment

To evaluate the elements front the process variability effects, firstly we choose a subset of basic cells implemented with complex cells and compare this set with a subset of basic cells implemented only with basic cells NAND, NOR and INVERTERS. Both subsets represent the same logic functions and the cells are dimensioned to offer the same delay times of the signal in the out pin cell.

Each cell of the subsets were descript in SPICE, in the PTM 65nm technology. The experiment obeys the flow showed in Fig. 2. Firstly, according to [1] the random process variability impacts mainly the parameters Vdd, Vth, L, Leff and W of the transistors in the nano technologies. With this information, the 65nm PTM models were modified to receive random values according to an statistical normal distribution around the original values from the 65nm PTM model, with 3 Sigma of Standard Deviations. As the values are modified in the model, the correlation among the parameters is preserved. After, the circuits were simulated without the process variability for extract the normal values, i.e., values obtained with the original 65nm PTM model. The same circuits were simulated with Monte Carlo adopting the 65nm model modified to receive random parameters. The parameters observed for each circuit after the simulations were total power dissipated in the same interval t of time in the simulation and low-to-high propagation delay and high-to-low propagation delay.

Fig. 3 shows the complex cells and basic cells used in simulations and their logic functions. To each experiment, the logic function defined and implemented to obey the logical equivalence between the basic and complex versions.

The results from the experiments are exposed in Table 1 and 2. The first table, Table 1 descripts the results from the complex cells. The column Normal show the results from each circuit simulated with the original 65nm PTM model, without variability in the parameters. The column Max, Min, Mean, Variance and Standard Deviation (SD) present the data provided by the SPICE output as result after 10000 Monte Carlo simulations, where the parameters VDD, Vth, L, Leff and W of 65nm PTM model were varied according a statistical normal distribution with 3 Sigma. The same results are showed in Table 2 to the second set of cells: the basic cells.

The results found in Table 1, shows that in average the behavior of complex cells related to power and Thl (high-to-low propagation delay) and Tlh (low-to-high propagation delay) suffer small variation. However, it is important to highlight that the most relevant values in this experiments are the maximum and minimum values because these values indicates possible critical conditions of operation for this set of cells. Observing the maximum and minimum values for complex gates, note that there are cases where the process variability affects over 30% the values of Tlh and more than 15% the power results. It is important to highlight that the cct3 have its logical behavior affected with the process variability and by this reason the results for timing are omitted.

The simulations results of the implemented basic cells circuits were showed in the Table 2. Observe that the process variability cause maximum alterations around 20% for Tlh. In this case, the results de Tlh and Thl for Cct4 (correspondent to cct3) have not been take into account because cct3 fail when exposed to variability. However, the cct4 maintain the logical behavior under effects of process variability.

When we compare the results for the timing parameters observed in the experiments with the two set of cells, we observe that the timing parameters varied 13% less in the basic cells circuit than the results from the circuits with complex cells, for this contrast only cct1, cct2, cct5 and cct6 were considered because the timing results for cct3 present fail in the logical behavior when exposed to variability.

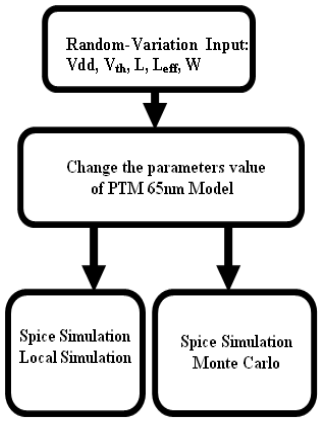


Fig. 2 - Experiment Flow

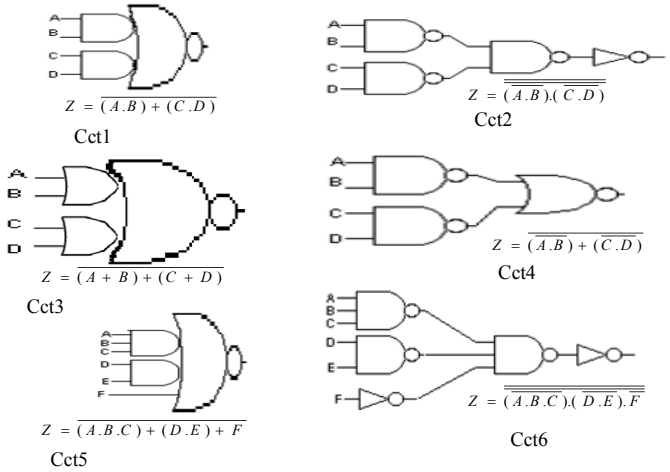


Fig. 3 - Subcircuits

4. Conclusions

This article presents a study about basic blocks to be adopted in a regular layout methodology. This research is engaged to find solutions to physical synthesis of integrated circuits less susceptible to process variability effects in the nano technologies. The main goal of this work is analyze the effects of process variability over a set of basic cells adopted in the regular layout approach. The procedure used to determine the best cells to be adopted in this approach, compare logic functions implemented with basic cells with complex cells circuits. The results demonstrate that circuits composed by basic cells suffer less impact of process variability in our simulations. This robustness could compensate the area penalties of these circuits.

Tab.1 - Complex Cells Simulation Results

Circuit	Parameter	Normal	Max		Min		Average		Variance	SD
Cct1	power	3,24E-05	3,34E-05	3,23%	3,17E-05	-2,23%	3,24E-05	0,13%	4,37E-14	1,67E-07
	Tlh	9,43E-11	1,25E-10	32,30%	7,59E-11	-19,50%	9,42E-11	-0,11%	3,01E-23	4,34E-12
	Thl	9,75E-11	1,07E-10	9,52%	8,82E-11	-9,51%	9,77E-11	0,25%	5,99E-24	1,95E-12
Cct3	Power	8,22E-05	9,47E-05	15,13%	7,04E-05	-14,37%	8,23E-05	0,08%	9,23E-12	2,45E-06
	Tlh	1,04E-10	--------		--------		--------		--------	--------
	Thl	1,27E-11	--------		--------		--------		--------	--------
Cct5	Power	1,20E-05	1,22E-05	1,64%	1,18E-05	-1,29%	1,20E-05	0,02%	3,56E-15	4,76E-08
	Tlh	9,43E-11	1,25E-10	32,30%	7,59E-11	-19,50%	9,42E-11	-0,11%	3,01E-23	4,34E-12
	Thl	9,75E-11	1,07E-10	9,52%	8,82E-11	-9,51%	9,77E-11	0,25%	5,99E-24	1,95E-12

Tab.2 - Basic Cells Simulation Results

Circuit	Parameter	Normal	Max		Min		Average		Variance	SD
Cct2	Power	3,58E-05	3,66E-05	2,41%	3,51E-05	-1,93%	3,58E-05	0,15%	3,43E-14	1,48E-07
	Tlh	1,77E-11	2,17E-11	22,99%	1,50E-11	-14,90%	1,79E-11	1,38%	7,46E-25	6,85E-13
	Thl	1,50E-11	1,83E-11	21,43%	1,31E-11	-12,81%	1,48E-11	-1,44%	3,09E-25	4,45E-13
Cct4	Power	9,67E-05	1,10E-04	14,16%	8,32E-05	-13,88%	9,67E-05	0,02%	1,23E-11	2,80E-06
	Tlh	3,50E-11	5,84E-11	66,67%	2,33E-11	-33,59%	3,55E-11	1,40%	1,28E-23	2,79E-12
	Thl	8,40E-12	1,12E-11	33,40%	7,69E-12	-8,47%	9,15E-12	8,95%	2,68E-25	4,34E-13
Cct6	Power	2,88E-05	2,95E-05	2,41%	2,85E-05	-1,26%	2,89E-05	0,36%	2,82E-14	1,34E-07
	Tlh	4,14E-11	4,92E-11	18,96%	3,64E-11	-11,91%	4,17E-11	0,88%	3,92E-24	1,59E-12
	Thl	7,09E-11	7,58E-11	6,90%	6,61E-11	-6,79%	7,13E-11	0,46%	2,66E-24	1,29E-12

5. Acknowledgment

This work is partially supported by Brazilian National Council for Scientific and Technological Development - CNPq – Brazil and Improving Coordination of Senior Staff – CAPES- Brazil.

6. References

[1] P. Gupta, A. B. Kahng, “Manufacturing-Aware Physical Design” in Proc. IEEE/ ACM Int. Conf. on Computer Aided Design, ICCAD-2003. San Jose, USA, 2003. pp. 681-687.

[2] L. Pileggi, et al., “Exploring Regular Fabrics to Optimize the Performance-Cost Trade-Off” in Proc. 40th Design Automation Conference, Anaheim, USA, 2003. pp.782 – 787.

[3] J. Sawicki, Achieving Better DFM: EDA Tools Pave the Way to Improved Yield. EDA Tech Fórum, v. 2, n. 2, p.28-32, jun. 2005.

[4] C. Meinhardt, R. Tavares, R. Reis, “Logic and Physical Synthesis of Cell Arrays” in Proc. 14th IEEE Int. Conf. on Electronics, Circuits and Systems, Marrakesh, Marocco, 2007.

[5] K. Wu, Y. Tai, “Structured ASIC, Evolution or Revolution?” in Proc. Int. Symp. on Physical Design 2004, Phoenix, USA, 2004. pp. 103-106

[6] J. Gu, K. F. Smith, “A Structured Approach for VLSI Circuit Design.” Computer, New York, v. 22, n.11, pp. 9 – 22, nov. 1989.

[7] B. Zahiri, “ Structured ASICs: Opportunities and Challenges” in Proc. 21st Int. Conf. on Computer Design, San Jose, USA, 2003, pp. 404- 409

[8] A. Weinberger, “Large Scale Integration of MOS Complex Logic”, IEEE Solid State Circuits, SC-2:182-190. Dez. 1967.

A Graph-based Approach to Generate Optimized Transistor Networks

Vinícius Possani, Éric Timm, Luciano Agostini, Leomar da Rosa Jr.

{vpossani.ifm, erict.ifm, agostini, leomarjr}@ufpel.edu.br

Group of Architectures and Integrated Circuits – GACI
Federal University of Pelotas – UFPel

Abstract

The number of transistors required for implementing a logic function is an essential consideration in digital VLSI design. While the generation of a series-parallel network can be straightforward once a minimized Boolean expression is available, this may not be an optimum solution. This paper proposes a graph-based solution for minimizing the number of transistors that compose a network. The algorithm starts from a sum-of-products expression and can achieve non-series-parallel arrangements. Experimental results demonstrates the efficiency of the approach when compared to the quick-factor algorithm implemented in the SIS software.

1. Introduction

Nowadays, VLSI design has definitely established a dominant role in the electronics industry. Automated tools have held designers to manipulate more transistors on a design project and shorten the design cycle. In particular, logic synthesis tools have contributed considerably to reduce the cycle time. In full-custom designs, manual generation of transistor netlists for each functional block is performed, but this is an extremely time-consuming task. In this sense, it becomes comfortable to have efficient algorithms to derive transistor networks automatically.

The common technique to optimize a transistor network is based on factorization [1-2]. In this procedure an input Boolean expression is manipulated in order to reduce the number of literals that compose the expression. Subsequently, this optimized expression is translated in a transistor network composed by a reduced number of switches. Alternative methods are also available in the literature. They are based on graph optimizations, were each edge in the graph keeps an association with a transistor in the network. The main idea is try to minimize the edges in a existent graph [3] or to compose a new graph with a reduced number of edges [4].

In this sense, this paper proposes a graph-based method to generate transistor networks. In our approach, the input Boolean expression is translated into a graph that is later optimized through edges sharing.

The remainder of this paper is organized as follows. Section 2 presents the proposed method to optimize transistor networks. Section 3 describes the implemented tool and presents the experimental results. The conclusions are presented in Section 4.

2. Graph-based Approach

The proposed graph-based approach accepts a sum-of-products (SOP) expression as input. In order to translate the expression to a graph, a parser is needed. The basic idea of this parser consists, initially, in separating all products that compose the SOP. In the sequence, each literal present in the product is extracted and stored in a vector. For each product a vector is created. At the end, the parser will delivery '*n*' vectors for the '*n*' products that compose the input Boolean expression. Figure 1 illustrates all vectors obtained from the Expression 1 that represents an input SOP.

$$S = \overline{A}CEF\overline{G}H + \overline{A}BF\overline{H} + A\overline{B}C\overline{G}H \quad (1)$$

!A	C	E	F	G	H
!A	B	F	!H		
A	!B	C	!G	H	

Fig. 1 – Vectors representing the products from Expression 1.

Once the list of vectors is obtained, they are organized according to the number of literals that compose them. Figure 2 exemplifies this situation.

Afterward, it is started the assembly of the graph by removing a vector at a time from this list and creating a edge in the graph for each literal found in the vector. Notice that this operation will create a set of edges

connected in series. This is demonstrated in Figure 3, where it is also possible to see the vertices that make connections between the pairs of edges.

!A	C	E	F	G	H
A	!B	C	!G	H	
!A	B	F	!H		

Fig. 2 – Ordered and organized vectors representing the products from Expression 1.

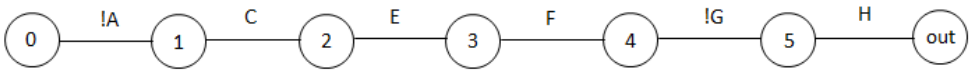


Fig. 3 – First product as a graph.

In the sequence, another vector is loaded from the list of vectors and placed in the graph. Figure 4 illustrates that for the second vector in the list. All paths in the graph are traversed in order to recognize identical vertices (vertices that represent same literals). If this condition is verified in the graph, then the identical edges are shared. This operation leads to a decrease of edges count. This is exemplified in Figure 5, where the edges 'C', '!G' and 'H' are merged.

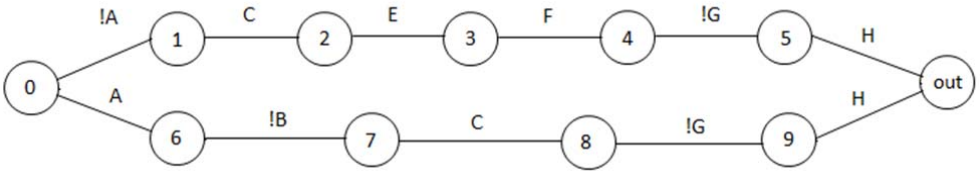


Fig. 4 – Graph composed by the two first products.

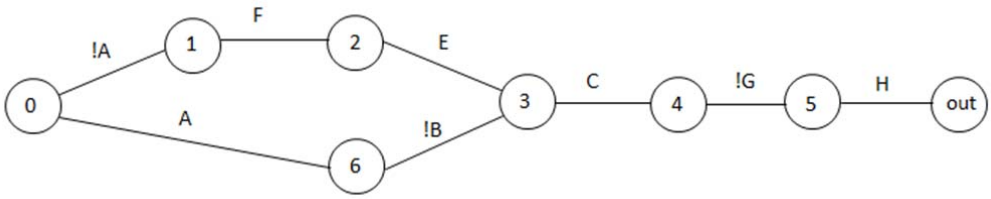


Fig. 5 – Optimized graph for the two first products.

This procedure is performed until the list of vectors is completely empty. Figure 6 shows the last vector added to the graph, and Figure 7 presents the optimized graph obtained after sharing the edges.

The basic idea behind organizing the vectors consists in permitting the edge sharing for larger products first. It has been empirically observed that worst solutions are obtained if the sharing process starts using smaller length vectors (products with less literals).

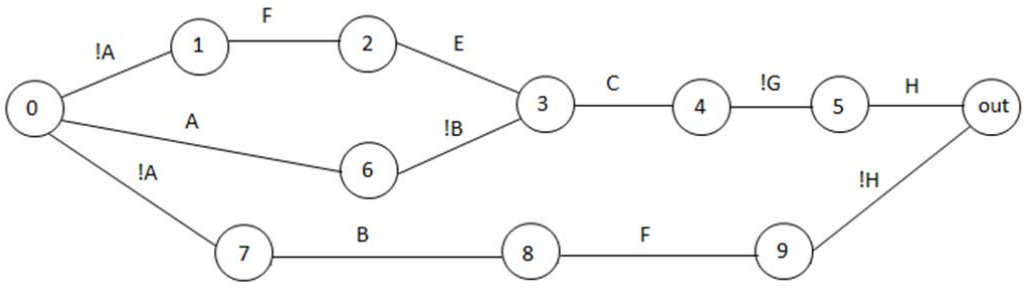


Fig. 6 – Graph with the last product added.

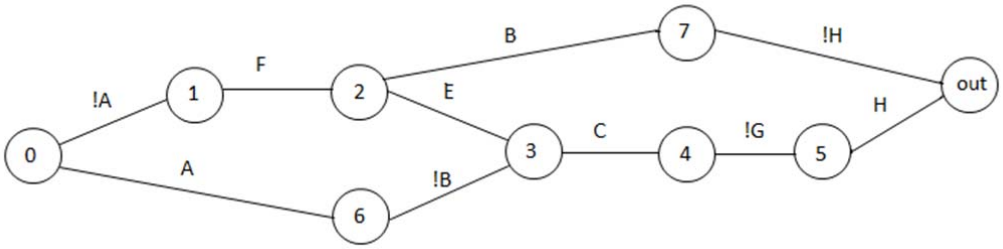


Fig. 7 – Obtained graph after the final optimization.

To guarantee that sneak-paths (forbidden paths) are not introduced in the graph, a routine that traverses the graph and compares it to the original products of the expression is regularly invoked. This is necessary because if a sneak-path is introduced, the graph will not be a true representation of the input Boolean expression.

Notice that all original products of the SOP are present in the graph through the paths '!A F B !H', '!A F E C !G H' and 'A !B C !G H'. However, by sharing edges a new path 'A !B E B !H' was also introduced. This path is allowed since it not modifies the logical behavior. When thinking in a transistor network, this new path cannot be sensitized because it contains both literals 'B' and '!B'. In order words, this is not a valid path.

Another interesting fact is that the proposed approach may derive bridge networks like methods proposed by [3] and [4]. The example illustrated in Figure 7 presents a bridge configuration (through edge 'E'). It is a benefit over optimization approaches based on factorization that can only derive series-parallel networks. For several Boolean functions the series-parallel arrangement is not the most optimized solution [5].

3. Experimental Results

The proposed method has been implemented in Java language using the NetBeans IDE 6.5.1. A tool containing the core algorithms and the graphics interface was developed. The graphics interface uses the Prefuse library [6]. Figure 8 illustrates it for the example from Expression 1. Also, the tool presents a Spice netlist output module that is capable to print Spice files to be used in electrical simulators.

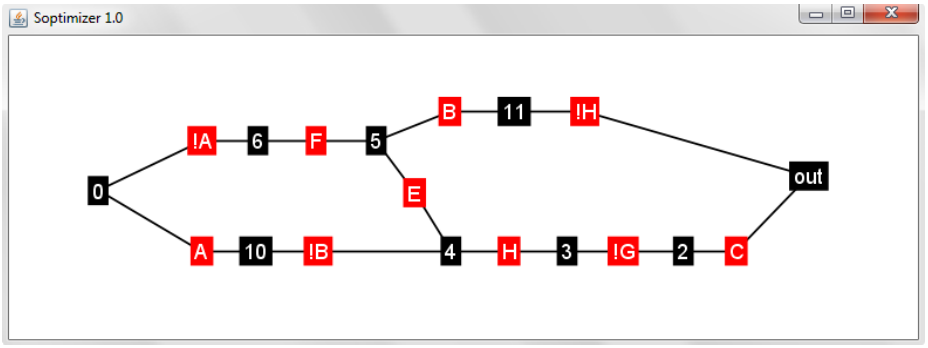


Fig. 8 – Graphical interface of the developed tool.

In order to evaluate the proposed approach, 10 Boolean functions with 7 input variables were randomly choose. They were introduced in SIS software [7] and extracted in their SOP form. Table 1 presents the obtained results. The total number of literals for the SOP form are described in column "# literals SOP form". The expressions were factorized using the quick-factor algorithm from SIS. The results are shown in column "# literals SIS". Results obtained using the proposed approach are shown in column "# edges Soptimizer". The obtained gain over the SIS software are shown in column "% of gain".

In most cases the proposed approach achieved smaller results. Analyzing the obtained networks we identified several bridge configurations in the arrangements delivered by the tool. On the other hand, SIS software delivers optimized expressions composed by 'AND' and 'OR' operators. In this case only series-parallel networks can be implemented, representing an overhead in terms of area (transistor count).

For function F7, the proposed algorithm was not able to deliver a smaller solution. The point in that case is the ability of the proposed algorithm to achieve bridge configuration. For this input expression the algorithm could not find bridges. We believe that it is related to the SOP ordering that is used to compose and to optimize the graph. As mentioned before, we starts using products with a larger number of literals. However, when the products that compose the SOP presents almost same number of literals, we choose them randomly. This situation could be leading for these kind of result.

Tab. 1 – Results for 10 randomly choose Boolean functions with 7 input variables.

Function	# literals SOP form	# literals SIS	# edges Soptimizer	% of gain
F1	133	80	59	26,25
F2	92	70	44	37.44
F3	78	62	39	37.10
F4	150	78	70	10.26
F5	119	82	55	32.93
F6	71	44	38	13,64
F7	170	76	82	-7,89
F8	135	78	62	20.51
F9	111	74	59	18.75
F10	97	64	52	18,75

4. Conclusion and Future Works

This paper presented a graph-based approach to generate optimized transistor networks. The proposed solution is able to deliver bridge networks. The algorithm was implemented in Java language and a graph interface using Prefuse library is available. 10 Boolean expressions with 7 input variables were randomly choose to be used as benchmark. The results demonstrated that the proposed approach can delivery networks with a transistor reduction up to 37.44% if compared to the quick-factor algorithm from the SIS software.

As future works we intend to investigate the impact of selecting different products ordering to start the optimization process. Also, we intend to compare the proposed solution with the method described in [4].

5. References

[1] Brayton, R. K. Factoring logic functions. IBM J. Res. Dev. 31, 2 (1987), 187-198.

[2] Mintz, A. and Golumbic, M. C. Factoring boolean functions using graph partitioning. Discrete Appl. Math. 149, 1-3 (2005), 131-153.

[3] J. Zhu et al. On the Optimization of MOS Circuits. IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications. (1993), 412-422.

[4] D. Kagaris et al. A Methodology for Transistor-Efficient Supergate Design. IEEE Transactions On Very Large Scale Integration (VLSI) Systems. (2007), 488-492.

[5] Da Rosa Jr, L. S. Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles. PhD Thesis PGMicro/UFRGS, Porto Alegre, Brazil. (2008), 147 p.

[6] Prefuse.org. The Prefuse Visualization Toolkit. [Online] Avaliable: http://prefuse.org/ [Acessed: Mar. 25, 2010].

[7] Sentovich, E.; Singh, K., Lavagno; L., Moon; C., Murgai, R.; Saldanha, A., Savoj; H., Stephan, P.; Brayton, R.; and Sangiovanni-Vincentelli, A. SIS: A system for sequential circuit synthesis. Tech. Rep. UCB/ERL M92/41. UC Berkeley, Berkeley. (1992).

GDLR: a Detailed Routing Tool

Charles Leonhardt, Adriel Ziesemer, Ricardo Reis
{ccleonhardt, amziesemer, reis}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul

Abstract

Routing is the step of the EDA where the interconnections between different elements of the circuit are performed. This stage has major consideration because the influence of the interconnection in total delay of the circuit and the increase of area that a difficult routing can bring. Moreover, the execution time is a limiting factor that prevents algorithms with greater complexity of being used to solve this problem. So this work aims to develop a tool for detailed routing that is capable of routing circuits with a significant number of elements with a good compromise between routing quality and execution time. This is achieved using A, Pathfinder and other optimizing techniques that are presented in this paper. The results obtained are 50% faster in execution time and have similar results for interconnection cost when compared to another tool for detailed routing called RotDL.*

1. Introduction

Despite being one of the first areas of EDA to be automated, VLSI routing remains an area of active research and development. Furthermore, if compared to positioning problem, there is significantly less algorithms proposing solutions to this problem.

Routing is a crucial step in the project of VLSI integrated circuits. It is the following step to positioning that determines the location of each element in the integrated circuit. Summarizing it is the step that connects all the positioned components obeying to design rules.

It is typically an extremely complex combinatorial problem. It is usually solved by an approach of two steps: global routing followed by detailed routing. The global router first divides the circuit in regions and decides the paths between the regions for all nets. After, according to the paths obtained in global routing, the detailed router assigns channels and vias for all nets.

The GDLR is based on the Pathfinder algorithm [1] to solve conflicts between nets and relies on A* [2] to perform the routing of each net (signal routing).

For using A* algorithm was used a grid for the data structure of the nodes instead of a simple graph. This is done because a simple graph would require the use of a lookup table, and the use of a lookup table would add extra processing and usage of memory.

The article is organized in 5 sections. Section 2 explains the A* algorithm. In section 3 the algorithm Pathfinder is explained. A technique to improve the results is present in section 4. The results are presented in section 5. Finally, in section 6 we have the conclusions and future works.

2. A* algorithm

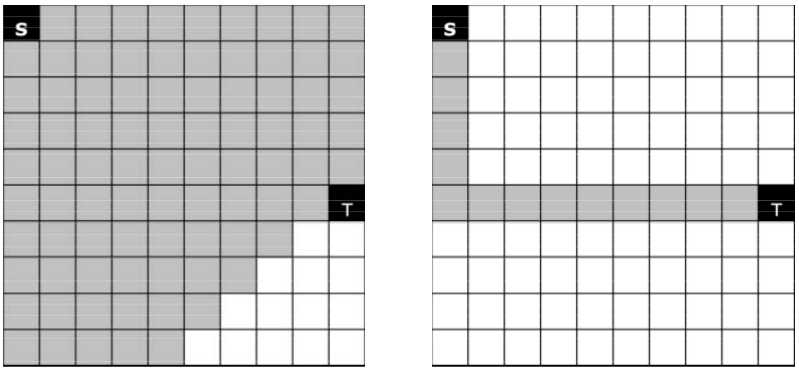
The algorithm is used to find the least-cost path from an initial node to a goal. The key of the algorithm is the cost function F(x) that take into account the distance from the source G(x) and the estimated cost to the goal H(x) as shown below:

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

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 [3].

H(x) must be an admissible heuristic, for this reason it cannot 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 Figure 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 14 nodes were visited until the target node be reached.



a) b)

Fig. 1 – Search using: a) Lee; b) A*.

3. Pathfinder

Pathfinder is a routing algorithm used in the congestion negotiation of nets (NCR). Primarily develop to be used in FPGA's, have been used in several academic routing tools as [4], [5] and [6]. It is commonly used for global routing, where the edges have capacities greater than 1, but its use for detailed routing has not been much explored yet.

In this algorithm all the nets are routed using the Pathfinder algorithm 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 Pathfinder router calls the signal router many times to connect all nodes from every net of the circuit. If a conflict happens, the router adjusts the cost of the use of the congested node accordingly to one of the equations below to make its use more expensive by the signal router.

$$C_n = B_n + (H_n * P_n) \quad (1)$$

$$C_n = (B_n + H_n) * P_n \quad (2)$$

These equations takes in account the base cost B_n, 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 router finds no more conflicts between the nets. The nets that are already solved are not routed again while it is free of conflicts.

4. Source selection

Reviewing the results from the comparison with RotDL [7] (standard of comparison used in this work) it was possible to see that was necessary to improve the execution time of the algorithm. Within this goal several techniques have been studied.

One that retrieves good results was setting as initial node (source) to the signal router the one closer to the geometric center of the net. The reason for this improvement is that starting from the center of the net less nodes in average are expanded to complete the routing of the net. But for more certainty is necessary a better study over the topologies generated.

This technique has also been applied to fractions of the total nets. The results we obtained were not as good as applying it for all nets.

5. Results

In order to illustrate the results, a set of tests has been performed. Tab. 1 compares the results between source selection technique and the original approach. The Tables 2 and 3 compare the results from the approach using the source selection technique and RotDL for the same test cases. RotDL is a detailed router from the Grupo de Microeletrônica(GME) from UFRGS.

For comparison two measures were used, total interconnection cost and execution time. The total interconnection cost is proportional to the wirelength but penalties are set if the preferred layer direction is not followed. The column #Nets represents the number of nets of each circuit. These nets are composed by an amount of nodes varying from 2 to 5. The comparison between different approaches is shown by fractions from the original approach. Between parentheses in each column identifier is the equation used (E1 or E2) and if the

source technique is used (+S). Three executions of each test case were performed and the results are the averages of these executions. For generating the results was used the tool ASTRAN (described in [8] and obtained in [9]) in a PowerPC G5 2GHz with 4GB DDR SDRAM.

According to the Tab. 1, the best results are obtained when we select the node closer to the geometric center of the net as the source node for all nets. In this configuration the interconnection cost is approximately the same and the execution time is 4.4% faster than the original approach.

Tab.1 – Comparison between source selection technique and the original approach considering total interconnection cost and execution time.

		Interconnection cost		Time(s)		Comparison	
Grid	# Nets	GDLR(E2)	GDLR(E2+S)	GDLR(E2)	GDLR(E2+S)	Cost	Time
400x400	120	252034	252352	0,642	0,693	1,001	1,081
400x400	200	415454	413676	3,736	3,356	0,996	0,898
400x400	280	590423	591074	12,965	11,396	1,001	0,879
600x600	180	531329	529873	1,435	1,442	0,997	1,005
600x600	300	917201	916476	9,268	8,876	0,999	0,958
600x600	420	1335346	1336206	53,562	44,615	1,001	0,833
800x800	240	997996	995030	3,929	3,776	0,997	0,961
800x800	400	1648364	1648971	23,230	24,608	1,000	1,059
800x800	560	2324052	2327460	98,877	100,066	1,001	1,012
1000x1000	300	1520575	1517268	5,232	5,701	0,998	1,090
1000x1000	500	2584728	2577709	46,963	43,629	0,997	0,929
1000x1000	700	3692922	3696869	208,526	185,193	1,001	0,888
Average						**0,999**	**0,966**

After incorporating the source selection technique to GDLR, the results obtained with the two equations of Pathfinder are compared with the tool RotDL. The results of this comparison are shown in the Tables 2 and 3. The results using equation (1) are in Tab. 2 and the ones using equation (2) are in Tab. 3.

Equations obtain similar results in terms of interconnection cost while the second one obtains better results for execution time. The speed-up of the equation (2) is approximate 50% when compared to RotDL.

Tab.2 – Comparison of the equation (1) for Pathfinder with RotDL.

		Interconnection cost		Time(s)		Comparison	
Grid	# Nets	RotDL	GDLR(E1)	RotDL	GDLR(E1)	Cost	Time
400x400	120	246731	248387	5	0,778	1,007	0,156
400x400	200	404844	422531	8	15,387	1,044	1,923
400x400	280	576316	592037	11	59,657	1,027	5,423
600x600	180	521273	548652	16	5,954	1,053	0,365
600x600	300	898972	904999	25	18,319	1,007	0,723
600x600	420	1304887	1328778	37	194,155	1,018	5,295
800x800	240	981389	954945	39	4,458	0,973	0,113
800x800	400	1618263	1638575	66	140,199	1,013	2,135
800x800	560	2276922	2332361	95	551,929	1,024	5,789
1000x1000	300	1492669	1566274	107	5,303	1,049	0,050
1000x1000	500	2539368	2679106	197	265,340	1,055	1,347
1000x1000	700	3631745	3516838	241	932,998	0,968	3,866
Average						**1,020**	**2,265**

Tab.3 – Comparison of the equation (2) for Pathfinder with RotDL.

		Interconnection cost		Time(s)		Comparison	
Grid	# Nets	RotDL	GDLR(E2)	RotDL	GDLR(E2)	Cost	Time
400x400	120	246731	252352	5	0,693	1,023	0,139
400x400	200	404844	413676	8	3,356	1,022	0,420
400x400	280	576316	591074	11	11,396	1,026	1,036
600x600	180	521273	529873	16	1,442	1,016	0,088
600x600	300	898972	916476	25	8,876	1,019	0,350
600x600	420	1304887	1336206	37	44,615	1,024	1,217
800x800	240	981389	995030	39	3,776	1,014	0,096
800x800	400	1618263	1648971	66	24,608	1,019	0,375
800x800	560	2276922	2327460	95	100,066	1,022	1,050
1000x1000	300	1492669	1517268	107	5,701	1,016	0,053
1000x1000	500	2539368	2577709	197	43,629	1,015	0,221
1000x1000	700	3631745	3696869	241	185,193	1,018	0,767
Average						**1,020**	**0,484**

6. Conclusions and future works

The GDLR obtain good results in the task of reducing execution time with speed-up close to 50% when compared to RotDL, with equivalent results for interconnection cost.

Another result is that the equation (2) for Pathfinder is more adequate than the other due better results for execution time and comparable results for interconnection cost.

One idea to improve this work is to direct the search to use Steiner nodes on finding the path from source node to target node. Another future work is to develop a global router that supplies smaller problems to the detailed router. The idea is to also use the algorithm Pathfinder in this solution. The objective of this development is to allow the tool to route even bigger circuits with reasonable quality in feasible time.

7. References

[1] L. McMurchie, C. Ebeling, “Pathfinder: a negotiation-based performance-driven router for FPGAs,” in International Symposium on Field Programmable Gate Arrays, Monterrey, Feb. 1995.

[2] 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, number 2, pp. 100-107, July 1968.

[3] C. Y. Lee, “An algorithm for path connections and its applications,” in IRE Transactions on Electronic Computer, vol. EC-10, number 2, pp. 364-365,1961.

[4] C. M. Pan, C. Chu, “FastRoute: A step to integrate global routing into placement”, in International Conference on Computer Aided Design, pp. 464-471, 2006.

[5] Y.J. Chang, Y.T. Lee, T.C Wang, “NTHU-Route 2.0: A Fast and Stable Global Router”, in Computer-Aided Design, ICCAD, pp. 338-343, 2008.

[6] J. A. Roy, I. L. Markov, “High-performance routing at the nanometer scale,” in Proceedings of IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, pp. 496–502, 2007.

[7] G. Flach, R. Hentschke, R. Reis. Algorithms for improvement of RotDL router. in South Symposium On Microeletronics, SIM, 2004, Ijuí, Brazil, 2004.

[8] A. Ziesemer Jr., C. Lazzari, R. Reis, “Transistor level automatic layout generator for non-complementary CMOS cells”, in Very Large Scale Integration – System on Chip(VLSI-SOC), pp. 116-121, 2007

[9] A. Ziesemer Jr., "ICPD" [Online]. Available: www.inf.ufrgs.br/~amziesemerj/icpd/, April 2010.

A Software Tool for the Analysis of Reliability in Combinational Logic Circuits

[1]Mateus Borges, [1]Rafael Oliveira, [1]Denis Franco
{mateibor, raf.floretino}@gmail.com, denisfranco@furg.br

[1]Universidade Federal do Rio Grande - FURG

Abstract

The manufacturing process of electronic integrated circuits approaches certain limits. With the implementation of nanometric dimensions technology, combinational logic circuits may present operating characteristics outside the predefined error margins, making system reliability an important requirement to be taken into account in the design process, alongside with performance, power consumption and costs of manufacturing. This paper proposes a system that will evaluate the reliability of combinational logic circuits generated by synthesis. The proposed tool should be able to support the automatic generation of circuits that are more resistant to faults. It will use the PBR (Binomial Probabilistic Model for Reliability analysis) circuit evaluation technique, which provides accurate results with reduced computation complexity compared to other evaluation approaches. The circuit analysis will be done in a multiple fault environment emulated on a FPGA device in order to achieve a higher gain in performance, without disregarding the possibility of multiple simultaneous faults occurrence.

1. Introduction

Integrated Circuits have evolved constantly in the last decades. The continuous reduction in the dimensions of integrated circuits results in an increase of density and speed obeying Moore's Law. Due to technological developments in the manufacturing process, the systems incorporate new features, improving performance, accompanied by the reduction of manufacturing cost and power consumption. However, the continuous reduction in the size of integrated structures is leading the process to reach technological, financial and physical limits, which will result in the break of Moore's Law.

All the above effects can be observed through the difficulty that currently exists in fabricating circuits with nanotechnology that present the characteristics of operation within the predefined margins of error, as shown by Franco et al. [1]. For this reason, the scientific community is studying the adoption of fault-tolerant circuits in the implementation of integrated systems in nanotechnology.

Fault tolerance is an area of study that deals with systems for critical applications where reliability is extremely vital, such as in military, aviation, space and medical systems. Its implementation in general purpose systems, however, is not a simple design decision, since the main method of implementing fault tolerance is the incorporation of redundancy (temporal or material) in the concerned circuits. This causes loss of performance, increased cost and power consumption, design requirements that are often more privileged in consumer electronics.

Thus, designers of integrated circuits in nanotechnology are faced with this paradoxical problem. On one hand smaller dimension circuits must be protected against this loss of reliability. On the other hand the application of fault tolerance means loss of performance, increased energy consumption and manufacturing cost. The solution may be a partial implementation of redundancy in the circuits, as shown by Zhou and Mohanram [2] in order to increase the reliability only on the critical elements reducing the redundancy needed for reasonable protection of the system.

To implement the partial fault tolerance, the critical elements in question must therefore be determined. Then, the final system reliability considering the additional protection circuits must be evaluated. Several techniques have been proposed to evaluate the reliability, but computational complexity and the various metrics associated with the problem show that exact solutions are not possible yet, which leads to specific solutions for each system. Furthermore, in most of the proposed techniques only single faults have been considered and no accurate analysis were made for circuits under multiple faults.

One technique that shows interesting results is called PBR as demonstrated by Vasconcelos et al. [3], which is based on simulation and / or emulation of the target circuit in the presence of faults. Despite the high complexity of calculating the exhaustive simulation, exact results may be obtained through the simulation of specific cases for each type of analysis in order to reduce the complexity involved in the circuit evaluation. Through the use of reconfigurable substrates like FPGAs, further reduction of computation time is possible, in order to achieve gains in performance due to the evaluation of the circuit in its actual frequency of operation and the possibility of replication of the test circuits in order to simultaneously evaluate multiple copies of the target circuit.

This project aims to develop a tool to evaluate the reliability of logic circuits using the PBR analysis method and circuit emulation based on FPGA. Thus, it is a proposal to develop a software environment that can

be integrated with other tools used in the design of integrated circuits as well as the hardware platform for the emulation of the target circuits in the presence of faults.

2. Probabilistic Binomial Model for Reliability Analysis (PBR)

The reliability of a system can be defined as the probability that it will perform the functions that were proposed under certain conditions and for a certain time, being referred to as R or R (t). Considering the logical gates as the basic components of circuits, the reliability of these can be expressed through its fault rate, defined as λ or λ(t), which is determined by simulations and data obtained from testing after manufacture or during operation.

The reliability of a logic circuit is a function of the reliability of its individual components, as well as the entire logic structure of the circuit. Nevertheless, the relationship between a fault in an individual component of the circuit and a fault in the outputs of the circuit does not occur directly. There may be fault masking of one of these components caused by the logical values present in the input of the circuit at the time of failure. This masking mechanism, called logical masking, together with the electrical and temporal masking, allows these circuits to function correctly many times in the presence of faults, as demonstrated in Miskov-Zivanov and Marculescu [4].

In the PBR model of reliability analysis, the objective is to determine the probability that a failure occurs in a certain circuit by evaluating its ability to logically mask certain faults. This way, the evaluation represents the probability of the output of the circuit being correct in the presence of faults. The PBR model is described by the following expression (Fig. 1) for a circuit with n inputs and G gates:

$$R_{cir} = \frac{1}{2^n}\sum_{k=0}^{G-1}(1-q)^k q^{G-k}\sum_{j\in\sigma}\sum_{i=0}^{2^n-1}\left(\vec{y}\left(\vec{x_i}, f_0\right)\oplus\vec{y}\left(\vec{x_i}, f_j\right)\right)$$

Fig 1 - The expression of the PBR model.

The model takes into account a binomial distribution of faults in the circuit, considering all the gates with the same reliability q and fault probability 1 - q, and the ability of masking (logically and electrically) this faults, expressed through the XNOR function presented in the expression. The ability of masking is calculated by simulating the circuit in the presence of faults (vector fj) and verification of the correctness of values in the output of the circuit. The circuit to be evaluated must be modified so that it is possible to inject controlled faults, according to Fig 2. The vector fj is composed of control signals that, when in level 1, trigger the injection of a fault in the circuit (inverts the value present at the output of its logical gate).

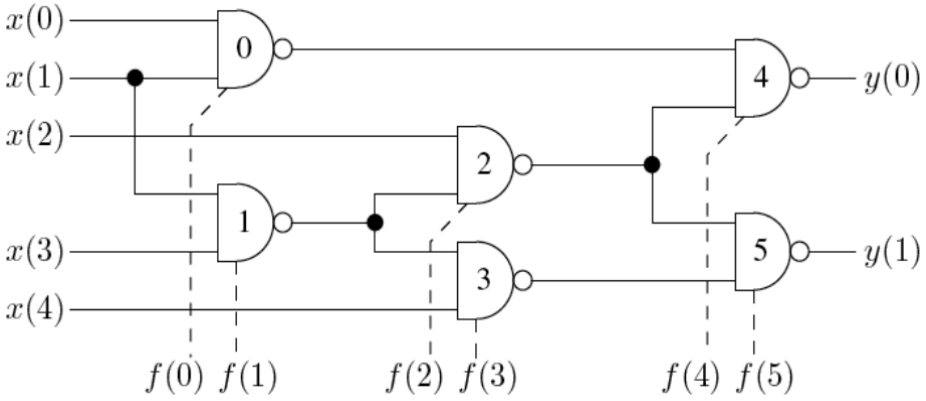


Fig 2 - Target circuit modified for the injection of faults.

3. A Tool for the Analysis of Reliability

To evaluate the reliability of combinational logic circuits, the current work proposes a tool that will work according to the scheme shown in Fig. 3. The system contains the circuit to be evaluated and its replica, modified for fault injection purposes. The verification of fault masking is done by comparing the output values of the circuit simulated in the presence of faults and the values of the original circuit simulation. The coefficients determined in the simulation reflect the capacity of logical masking occurred for a certain number of simultaneous faults.

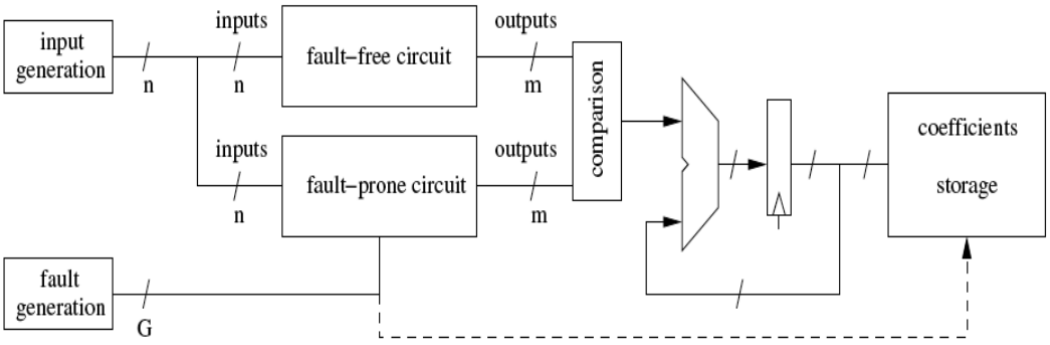


Fig 3 - Testbench structure for the PBR analysis

For circuits with dozens of logical gates, the simulation may have a very long time response or even prohibitive. The aim is then to simplify the method considering that for certain values of reliability of logical gates, the set of possible simultaneous faults is very small and can be determined by the expression of the binomial distribution of faults. Furthermore, applying random input patterns results in good accuracy with a reduced number of simulations.

This project aims to develop an integrated system to evaluate the reliability of combinational logic circuits generated from automatic synthesis tools.

The analysis will be done in a tool that aims to integrate several routines for automatic evaluation of logic circuits. These software routines will be used to interpret the files of the circuit description (in VHDL), generate the files needed for test bench simulation, and format the results in terms of tables and charts for reliability. Beyond environmental analysis software, the project foresees the creation of an emulation platform based on FPGAs, as shown in Fig. 4. The circuit emulation can result in important gains in terms of performance, if compared with the software simulation.

Beyond environmental analysis software, that has already been implemented in other studies, the application of PBR model in this tool will stress the emulation using the FPGA board Nios II Embedded Evaluation Kit, Cyclone III Edition, by Altera, as shown in Fig. 4. The circuit emulation can result in important gains in terms of performance, if compared with the software simulation.

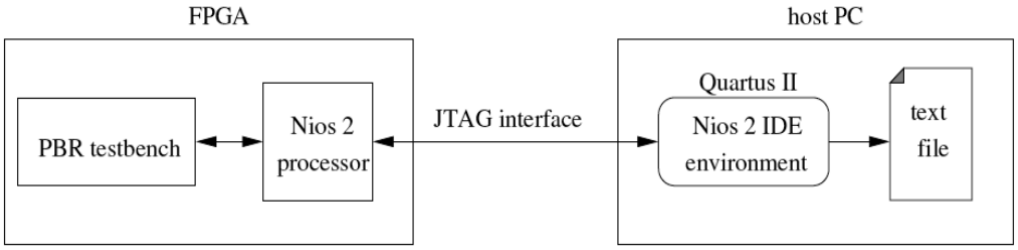


Fig. 4 – Emulation scheme for the PBR analysis

In addition to automatically evaluate fault-tolerant circuits, this tool will assess the effectiveness of techniques of fault tolerance in environments with multiple simultaneous faults. Most of these techniques only take into account that a single fault will occur in each moment, and the probability of multiple faults occurring in nanometer technologies cannot be disregarded, as demonstrated by Shivakumar et al. [5]. A fault-tolerant circuit usually has signs that indicate errors, so that the operation must be annulled and rerun. In such cases, the time penalty must be viewed with caution, so as not to compromise the expected performance of the circuit.

The time penalty is the fraction of useful results generated by the circuit compared to the total number of results generated. This reading may indicate the inadequacy of the technique of fault tolerance in use for the application in question indicating the need of other techniques for the solution of the system. The analysis tool proposed in this project also aims at assessing the time penalty of target circuits, providing an important and unprecedented measurement in this type of analysis.

The tool will be produced in a Windows environment and implemented in Java. As the simulation has already been implemented in other studies, the application of PBR model in this tool will stress the emulation using FPGA board Nios II Embedded Evaluation Kit, Cyclone III Edition, by Altera.

4. Conclusion

In this work is presented a proposal of a tool that will evaluate the reliability of a combinational logic circuit in a multiple simultaneous faults environment. To do that, it creates an emulation environment based on

FPGA implementation in order to inject controlled faults in the input of the circuit and, than, compare the output of this circuit and the output of a free-fault one.

The main contribution of this work is the use of the PBR approach, which presents reduced complexity to evaluate the reliability of a given circuit without losing accuracy. Using this method, it proposes the evaluation of existing fault-tolerant techniques by computing the time penalty behavior of circuits designed with these techniques.

5. References

[1] D.T. Franco, J.F. Naviner, and L. Naviner, "Yield and reliability issues in nanoelectronic technologies," *Annales des télécommunications*, 61(11-12):1422-1457, Nov.-Dec.2006.

[2] Q. Zhou, and K.Mohanram, "Transistor sizing for radiation handening," *Reliability Physics Symposium Proceedings, 2004. 42nd Annual. 2004 IEEE International*, April 2004, pp.310-315.

[3] M.C. Vasconcelos, D.T. Franco, L. Naviner, and J.F. Naviner, "Reliability analysis of combinational circuits based on a probabilistic binomial model," *IEEE Northeast Workshop on Circuits and Systems,* 2008. NEWCAS-TAISA 2008, June 2008, pp. 310-313.

[4] N. Miskov-Zivanov, and D. Marculescu, "Circuit reliability analysis using symbolic techniches," *Computer-Aided Design of Integrated Circuits and Systems*, IEEE Transactions on, 25(12):2638-2649, Dec. 2006.

[5] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi, "Modeling the eect of technology trends on the soft error rate of combinational logic, " *In DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, Washington, DC, USA, 2002. IEEE Computer Society, pp 389-398.

Evaluating Power Consumption on Buses under the Effect of Crosstalk

Carolina Metzler, Luigi Vaz Ferreira, Gustavo Wilke,
Ricardo Augusto da Luz Reis
{cmetzler, lvfernandes, wilke, reis}@inf.ufrgs.br

UFRGS - Universidade Federal do Rio Grande do Sul

Abstract

With the continuous scaling of technology, crosstalk increases proportionally due to reduction in metal line pitch and the increase in the interconnects aspect ratio. The crosstalk effect causes noise in adjacent wire when switching in opposite directions affecting the switching delay and power consumption of on-chip buses. In on-chip buses the crosstalk effects is more evident because on-chip bus is a set of long wires connecting two or more devices intra-chip, due to the long wire length there are more interconnects, thus, it is more susceptible for crosstalk effects.

In the present paper we study the crosstalk effects in power consumption of two coupled transmission lines of on-chip buses in accordance with their switching. The bus is modeled as distributed resistance-capacitance (RC) lines that utilizes 3-π wire model and the transmission lines are capacitively coupled to each other. The simulation results show that the power consumption in on-chip buses depends not only on the switching but how transmission lines are coupled as well. Results are collected from electric simulations for a 65nm ptm technology and indicate a significant difference of 40.95% in power consumption according the switching.

1. Introduction

Throughout the past years, VLSI technology has advanced at exponential rates in both productivity and performance. Thus, the intra-chip devices grow in complexity and results in more modules in the same chip. To connect these modules is used on-chip buses which is a set of long wires connecting two or more devices intra-chip. This implies that the number of intra-chip modules will increases and so will the number of on-chip buses connecting these modules where there is a need for significant data transfer and power dissipation [1].

A good evaluation on power consumption is crucial in current integrated circuit design, especially for low power constraints devices. With the continue scaling of the technology the power consumption of on-chip buses is becoming significant [2]. Due to the long wire length in on-chip buses there are more interconnects, thus, it is more susceptible for crosstalk effects.

While transistors switch much faster each new technology, meanwhile in wires is narrower because in many paths the RC delay is bigger than gate delay due to the growth of wire resistance. Furthermore, wires are packed very close each other and in consequence, part of their capacitance is to their adjacent wires. Thus, crosstalk is when one wire switches it tend to affect an adjacent wire due to the capacitive coupling between them, seem in Figure 1 and Figure 2. Therefore, when adjacent transmission lines of on-chip bus are switching in opposite directions the coupling capacitance between them lead to crosstalk, then, this effect causes noise in adjacent transmission line impacting in timing and power of a signal on a chip.

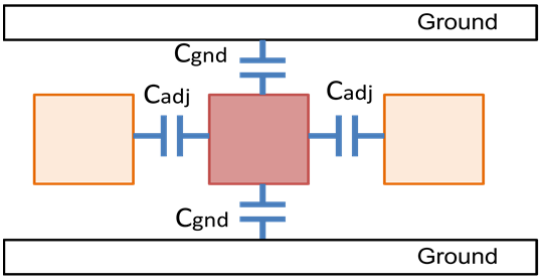


Fig. 1 – Coupling lines between two metal ground planes.

In this paper will be presented a evaluation of power consumption on two coupled on-chip bus lines, the main objective is show how significantly crosstalk affects power consumption in on-chip buses. Most of techniques that evaluates power consumption due to the number of transitions in on-chip buses [1,]. The aim of this work is show that power consumption depends not only on the switching but how transmission lines are coupled as well. The correlation between switching transition lines was evaluate in 50% statistically utilizing ISCAS 85 benchmarks, and indicates that is possible arrange transmission lines according their switching correlation.

The paper is organized as follows, in section1 an introduction about the theme, in section 2 are the related works and in section 3 the simulation results in two adjacent bus lines switching. Finally, section 4 presents the conclusion.

2. Related Works

Most of related works deals with the effects on delay due to switching adjacent bus lines as [3],[2], and papers concerned with the power dissipated there are [1],[2],[5]. In [1] Ghoneima et al proposes a variation-tolerant low power source-synchronous multicycle (SSMC) it replaces the intermediate flip-flops in the SSMC their circuits simulations saves up to 20% energy, there, is presented delay models and explains the Miller coupling coefficient is the value of the contribution of the coupling capacitances to the overall capacitance. According to [2] Ghoneima and Ismail, as the introduced delay increases the achieved power reduction increases while decreasing the bus throughput. As shown by Mahmoud et al [3] the improvement of the performance of an interconnect wire is physically limited when using a reverse-scalling approach, so they make an analysis of the absolute minimum damping factor and presents expressions that shows the maximum bit-rate on a wire can be quite limiting for longer wires and scalling trends.

According [5], Ghoneima and Ismail makes an analytical and qualitative analysis of relative delay on dissipated energy and the Miller coupling factor in coupled lines. In their work skewing the worst switching case is shown to provide up to 50% reduction in energy dissipation.

Different of related works, here is presented another perspective considering crosstalk in on-chip buses, in the other works, they aim to decrease crosstalk effects or do not consider it in total power consumption. In section 3 will be possible observe that crosstalk affects power consumption in on-chip buses and it is related with the switching in transmission lines.

3. Simulation Results

This section will show simulations results for two coupled adjacent bus lines according their switching. In this work, a transmission line is simulated with the 3-π model and an inversor as a driver for the switching process. To analyze how much crosstalk affects the power consumption different switching patterns was simulated.

With the purpose of quantifying the effects of crosstalk on a bus, was simulated the circuit on Figure 2 that shows two transmission lines coupled switching. All transmissions are simultaneous, assuming same clock on drivers. According the dynamic power equation (1) is possible calculate the values of power consumption for switching lines. All the capacitances, line capacitances, drain capacitances and inversors input capacitances are denoted by C. C is multiply by the switching frequency *f* on the load capacitances to obtain the current, and multiplying again by voltage V_{DD}, as shown below:

$$P_{dyn} = CV_{DD}{}^2 f \qquad (1)$$

The simulations results are taken in Synopsis HSPICE, where is described a wire model and an inversor to simulate a transmission line, all RC values are taken from [6] and the inversors have the following values L= 65nm, W_n=1um and W_p=2um for a wire length of 100um and the inversors input capacitance is 1.33fF. Calculating the dynamic power of the circuit simulated in this work, disregarding the drain capacitance, it is 1.1384E-04W for 1mm wire length and the simulation, table1, has 1.2810E-04W for lines switching in same directions, considering that the drain capacitance was disregard in the first value and not on the simulation value is possible conclude that data is according the theory.

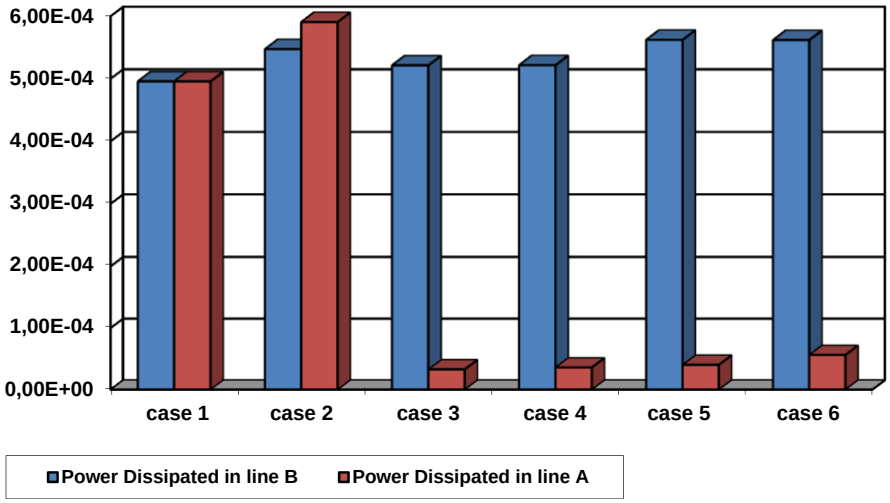
Regarding table 1 and graph 1, below, is possible understand which is the worst case in crosstalk, it is the case when switching in opposite directions. Therefore is possible conclude that direction of switching affects the amount of charge that must be delivered, this can be written as:

$$Q = C_{adj}\Delta V \qquad (2)$$

Equation (2) gives the charge delivered to the coupling capacitor, ΔV is the charge between line A and line B. If both lines switch in the same direction ΔV is zero, case 1 and the power dissipated is the same. In case 2, line A and line B switch in opposite direction ΔV is 2*VDD that explains why the power consumption is higher. When line A switches and line B does not, ΔV is equal to VDD the capacitance effectively seen by A is just the capacitance to ground and to B [7]. According graph 1, cases 4 and 6 have the proportional dissipated power value and both switch to a different value that line A is attached. Note that on contrary, in cases 3 and 5 line B switches to the same value that line A is constant. Evaluating graph 1 with equation (2) is possible see the results and confirm the theory.

The Power consumption in two wires adjacent are estimated in 23.64% bigger in the worst case for a 100um wire length and 40,95% for 1mm, simulations results are showed in table1. Hence, when the bus lines are switching in equal directions are adjacent, is possible save power.

Graph.1 – Power consumption when two coupled bus lines are switching in a 65 nm technology for a 100um wire length.



In second simulation was increasing the line size from 100um to 1mm and maintains the same technology and inversors size, as expected the RLC values increases according the wire length. Seem below the corresponding table 1 for a 1mm wire length.

Tab.1 – Power consumption when two coupled bus lines are switching in a 65 nm technology for a 1mm wire length.

Case	Line A	Line B	Power Dissipated in line A	Power Dissipated in line B
1	0→1	0→1	1.2802E-04W	1.2802E E-04W
2	0→1	1→0	2.1691E -04W	2.1664E -04W
3	0	1→0	4.3848E-05W	1.7241E-04W
4	1	1→0	4.4103E-05W	1.7247E-04W
5	1	0→1	4.4686E-05W	1.7228 E-04W
6	0	0→1	4.4918E-05W	1.7226E-04W

Evaluating the results are possible conclude that power values changes according transmission lines are coupled; for this reason is very important consider this when designing an on-chip bus.

In Figure 2 is the circuit simulated for 65 nm technology, all RC values are taken from PTM's web site [6]. PTM's model card provide accurate, customizable, and predictive model files and are compatible with standard circuit simulators, and scalable with a wide range of process variations.

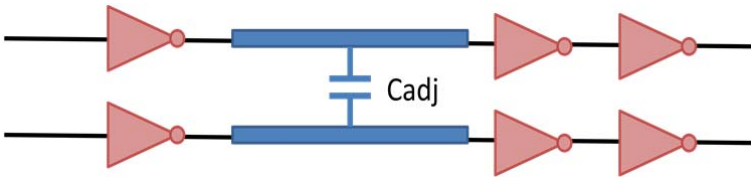


Fig. 2 – Coupled adjacent bus lines used in simulations.

4. Conclusions

This paper presented a study of power consumption on coupled on-chip bus lines, the main objective was show how significantly crosstalk affects power consumption in on-chip buses. These results are preliminary and the maximum difference of 23.64% for 100um wire length and 40.95% for 1mm wire length on power consumption according the switching direction. The simulation results show that the power consumption in on-chip buses depends not only on the switching but how transmission lines are coupled as well.

As a preliminary study, in this paper is presented simulation results for 65 nm technology utilizing the predictive technology model RC model. Different switching patterns are included in the model due to evaluate the changes in power consumption.

5. References

[1] M. Ghoneima, Y. Ismail, M. Khellah, V. De, "*Variation-Tolerant and Low-Power Source-Synchronous Multi-Cycle On-Chip Interconnection Scheme*", VLSI Design,vol.2007, article ID 95402, 2007.

[2] M. Ghoneima and Y. I. Ismail, "*Utilizing the effect of relative delay on energy dissipation in low-power on-chip buses,*" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 12, pp. 1348–1359, December 2004.

[3] N. Mahmoud, M. Ghoneima, and Y. Ismail, "Physical Limitations On The Bit-Rate Of On-Chip Interconnects," Proc. Great Lakes Symposium on VLSI (GLSVLSI), pp. 13-19, 2005.

[4] L. Macchiarulo, E. Macii and M. Poncino, "*Wire Placement for Crosstalk Energy Minimization in Address Buses*". Proc. of 2002 Design Automation and Test in Europe Conference and Exibition (DATE 2002) p.158, March 04-08, 2002

[5] Maged Ghoneima , Yehea Ismail, "*Delayed line bus scheme: a low-power bus scheme for coupled on-chip buses*", Proceedings of the 2004 international symposium on Low power electronics and design, August 09-11, 2004, Newport Beach, California, USA

[6] PTM- Predictive Technology Model http://ptm.asu.edu/

[7] Neil Weste, David Harris, "*CMOS VLSI Design: A Circuits and Systems Perspective*" 4th Edition, Hardcover, Mar. 11, 2010.

Video Coding 2

Low Latency and High Throughput Architecture for the H.264/AVC Transforms and Quantization Loop Targeting Intra Prediction

Daniel Palomino, Felipe Sampaio, Luciano Agostini
{danielp.ifm,fsampaio.ifm,agostini}@ufpel.edu.br

UFPel - Universidade Federal de Pelotas

Abstract

This work presents a low latency and high throughput architecture for the H.264/AVC transforms and quantization loop, focusing in the Intra Prediction constraints. The main goal is 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 FPGA Stratix III and Standard Cells. Considering the H.264/AVC main profile, the architecture is able to process 37 QFHD frames per second and 148 HD 1080p frames per second when mapped to FPGA, easily reaching real time for the two target resolutions. Besides, the architecture designed in this work reaches the best results considering the Intra Prediction restrictions, when compared with related works.

1. Introduction

The H.264/AVC standard is the state-of-art in video coding. It was developed by experts from ITU-T and ISO-IEC intending to double the compression rates, with no quality loss, when compared with previous standards, like MPEG-2 [1].

The video compression is extremely important considering the great amount of data in high resolution videos. The H.264/AVC has efficient tools to compress these videos. However, H.264/AVC encoders have high computational complexity as well. This way, hardware solutions are necessary to process high resolution videos in real time (24 to 30 frames per second).

The main modules of the H.264/AVC are: Inter-Frame Prediction (composed by Motion Estimation and Motion Compensation), Intra-Frame Prediction, Forward and Inverse Transforms, Forward and Inverse Quantization, Deblocking Filter and Entropy Coder.

The Intra-Frame Prediction module exploits the spatial redundancy in a frame, exploring similarities among neighbor blocks. The predicted block is generated coping neighbor pixels of blocks already coded. The blocks can be predicted by the Intra Prediction module of three different ways: (1) I4MB (prediction over 4x4 luminance blocks), (2) I16MB (prediction over 16x16 luminance blocks) and (3) chroma 8x8 (prediction over 8x8 chrominance blocks) [4]. In the coding process the H.264/AVC standard defines that the residual data between the original and the predicted block needs to be processed by the Forward Transforms (T), Forward Quantization (Q), Inverse Quantization (IQ) and Inverse Transforms (IT), to generate the reconstructed block that will be used as reference by the Intra Prediction module. While a block is processed by the T/Q/IQ/IT loop, the Intra Prediction module stays idle waiting for the reconstructed block. It means that there is a data dependency between neighbor blocks in the Intra Prediction process. Thus, the time that one block needs to be processed by the T/Q/IQ/IT loop affects directly the performance of the Intra Prediction module.

This work presents a low latency and high throughput architecture for the T/Q/IQ/IT loop to generate the reference blocks as fast as possible, decreasing the Intra Prediction idle time. Besides, the architectures for the T/Q/IQ/IT modules were designed intending to exploit the similarities between the transforms and quantization calculations, with the goal to decrease the hardware consumption. This work is part of the Brazilian research effort to develop the Brazilian System of Digital Television (SBTVD) [2], since H.264/AVC is the chosen video coding standard to be used in SBTVD.

This work focus in the main profile of the H.264/AVC standard [1], which works with YCbCr space color and 4:2:0 format. In this case, a Macro Block (MB) is composed by 16x16 luminance samples (Y), 8x8 blue chrominance samples (Cb) and 8x8 red chrominance samples (Cr) [1].

Fig. 1 shows the block diagram of an H.264/AVC encoder, where the target blocks are highlighted.

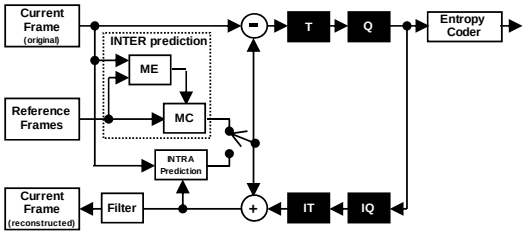


Fig. 1 – H.264/AVC encoder block diagram.

This work is organized as follows: Section 2 presents the Transforms and Quantization definitions. Section 3 shows de designed architecture. Section 4 shows the architecture pipeline schedule. Section 5 presents the synthesis results and some comparisons with related works. Finally Section 6 presents conclusions and some future works.

2. Transforms and Quantization

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].

$$Y = C_f W C_f^T \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} \quad \\ W \\ \quad \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 2 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (1)$$

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 [3]. 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 I16MB mode.

$$\begin{aligned} |Z_{(i,j)}| &= (|W_{(i,j)}| \cdot MF + f) >> qbits \\ sign(Z_{(i,j)}) &= sign(W_{(i,j)}) \end{aligned} \quad (2)$$

$$W'_{(i,j)} = Z_{(i,j)} \cdot V_{(i,j)} \cdot 2^{\lfloor QP/6 \rfloor} \quad (3)$$

All the other transforms and quantization definitions are very similar and they are presented in [4].

3. Designed Architecture

The T/Q/IQ/IT loop architecture was designed to generate the reference blocks as fast as possible to the Intra Prediction module. To achieve this goal, two decisions were taken: (1) to exploit the parallelism in the transforms and quantization operations as much as possible and (2) insert pipeline stages as less as possible to decrease the latency. Too many pipeline stages are not efficient, since there is a data dependency between the T/Q/IQ/IT loop and the Intra Prediction module.

The architectures for the T/Q/IQ/IT modules were designed intending to exploit the similarities between the transforms and quantization calculations, decreasing the hardware consumption.

Fig. 2 shows the block diagram of the designed architecture.

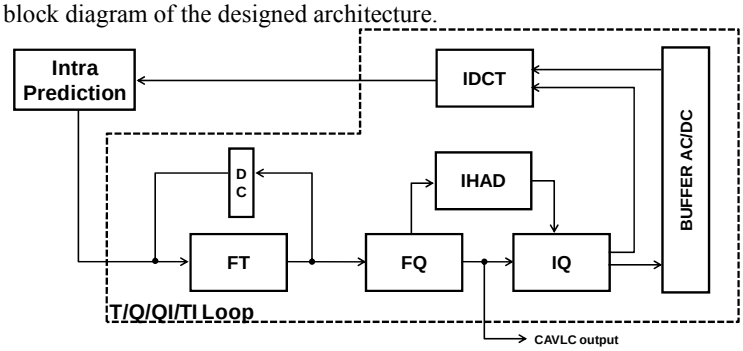


Fig. 2 – T/Q/IQ/IT loop block diagram.

In the T/Q/IQ/IT loop architecture each module represents one pipeline stage. The FQ was the only module designed with two pipeline stages, due to its long critical path.

3.1. Transforms and Quantization Architecture

All transforms defined in the H.264/AVC standard were grouped in three different architectures: FT, IHAD and IDCT modules. It was possible, since there are similarities among all transforms operations. Actually, all transforms operations could be grouped in only one module. It was not performed, in order to make the control unit simpler and the critical path smaller. All these groups are shown with more detail in [5].

All the forward quantization operations were grouped in one module (FQ) and the inverse quantization operations in another one (IQ). An architecture based in adders and shifts was designed to perform the multiplication operation in these modules. This way, all possible constants values were pre-calculated and

storage in dedicated memories. Thus, it was not necessary design a data path to perform the calculation of these constants, decreasing the hardware consumption and decreasing the critical path of these architectures.

Both buffer shown in the Fig. 2 (DC and Buffer AC/DC) are used to synchronize the modules of the T/Q/IQ/IT loop.

4. Pipeline Schedule

The T/Q/IQ/IT loop architecture works in three different ways, according to the operation modes: I4MB, I16MB to luminance samples and chroma 8x8 to chroma samples [4]. Fig 3 shows the architecture time diagram considering the three operation modes.

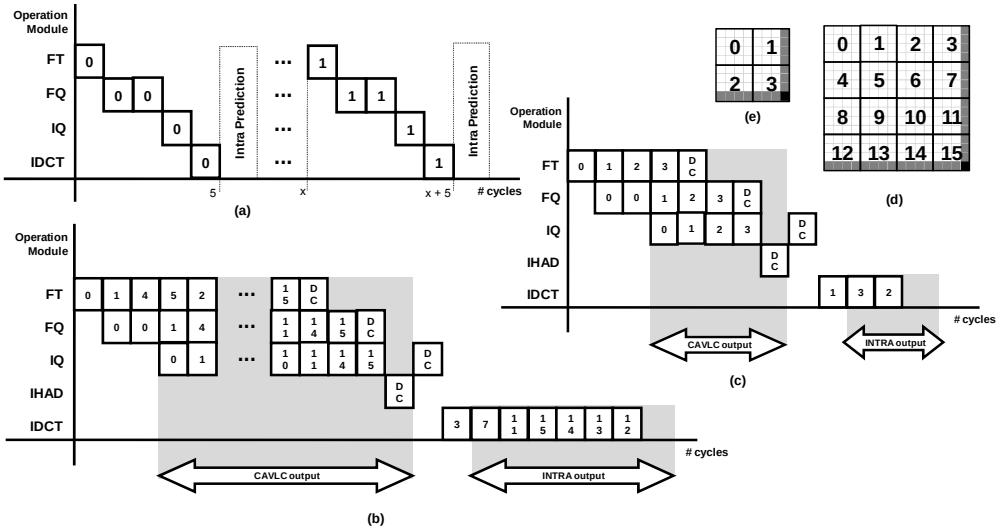


Fig. 3 – T/Q/IQ/IT loop time diagram for the three modes: (a) I4MB, (b) I16MB and(c) chroma 8x8. (d) luma 16x16 block and (e) chroma 8x8 block.

For the I4MB mode, the loop architecture delivers reconstructed blocks in five clock cycles to the Intra Prediction module. When the prediction mode was I16MB the loop architecture takes 28 clock cycles to deliver the reconstructed block to the Intra Prediction. Considering chroma blocks, 12 clock cycles are necessary to process an 8x8 block, 24 clock cycles for both chroma components (Cb and Cr).

5. Synthesis Results and Comparison

The architecture was described in VHDL and synthesized targeting two different technologies: FPGA Stratix III [6] and TSMC 0.18µm standard cells [7]. Tab. 1 shows the synthesis results considering operation frequency and hardware consumption (ALUTs and DLRs for FPGA and number of gates for standard cells).

Tab. 1- Synthesis Results.

Module	Stratix III*			TSMC 0.18µm	
	Freq. (MHz)	#ALUTs	#DLRs	Freq. (MHz)	#Gates
FT	161.1	1,135	528	185.3	9.6K
IT	182.7	1,131	544	189.4	8.2K
FQ	136.3	7,032	1,640	125.7	40.2K
IQ	131.5	3,833	880	136.9	21.9K
IDCT	180.3	1,445	688	161.5	10.3K
Control Unit	532.8	171	39	458.4	0.6K
T/Q/IQ/IT loop	125.0	14,111	6,846	112.0	155.0K

*Device: EP3SL50F780C3N

Two resolutions were considered, in order to evaluate the architecture performance: HD 1080p (1920x1080 pixels) and QFHD (3840x2160 pixels). Also, it was considered the Intra Prediction worst case, i.e., when the prediction mode is always I4MB for luminance blocks. This way, the designed loop will take 104 clock cycles to process one MB.

In the ideal case (when the Intra Prediction mode is able to deliver one 4x4 block per cycle) and considering the FPGA design, the architecture is able to process until 148 HD 1080p frames per second and 37 QFHD frames per second, reaching real time. In the real case, the Intra Prediction module delivers the predicted blocks in bursts. This way, to the Intra Prediction module and T/Q/IQ/IT loop reach real time with HD 1080p videos, the Intra Prediction module can takes until 410 clock cycles per MB.

It was not found works in the literature that implements only the T/Q/IQ/IT loop. However, there are works as [8], [9], [10] and [11] that implements Full Intra Coders, where the T/Q/IQ/IT loop is an internal module. Since these works do not show results only for the T/Q/IQ/IT loop, the comparison was made considering the latency for the T/Q/IQ/IT loop. Tab. 2 shows the number of clock cycles that each work takes to process the three different Intra Prediction modes: I4MB, I16MB and chroma 8x8.

Tab. 2- Comparison with related works.

Mode	This work	Kuo [8]	Chuang [9]	Suh[10]	Hamzaoglu [11]
I4MB	5	16	17	34	100
I16MB	28	159	-	441	1,718
Chroma 8x8	24	77	-	242	-

The architecture designed in this work presents the lowest latency considering all modes when compared with the related works. In the worst case the design architecture is 3.2 times lesser than [8], when chroma 8x8 blocks are processed. When compared with [11], the latency of the architecture designed in this work is 61.4 lesser, when 16x16 luminance blocks are processed.

6. Conclusions and Future Works

This work has presented an architecture for the T/Q/IQ/IT loop of the H.264/AVC standard with low latency and high throughput, in order to decrease the time that the Intra Prediction module stays idle waiting for the reconstructed blocks. The architecture was described in VHDL and synthesized targeting two different technologies: FPGA Stratix III and TSMC 0.18µm standard cells. The synthesis results shows that this architecture is able to process until 148 HD 1080p frames per second and 37 QFHD frames per second considering the FPGA design, reaching real time (24 to 30 frames per second) when high resolution videos are targeting. Besides, the architecture presented in this work shows the best results of latency than all related works.

The next step of this work is integrating the T/Q/IQ/IT loop architecture with the Intra Predictor module and the Entropy Coder in order to design a Full Intra Coder.

7. References

[1] ITU-T Recommendation H.264/AVC (05/03): advanced video coding for generic audiovisual services. 2003.

[2] 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] Sampaio, F. et al (2009) “A Multitransform Architecture for the H.264/AVC Standard and Its Design Space Exploration”. In ICECS 2009, pages 711-714.

[6] Altera Corporation. “Altera: The Programmable Solutions Company”. Available at: www.altera.com.

[7] Artisan Components. *TSMC 0.18 µm 1.8-Volt SAGE-X™ Standard Cell Library Databook*. 2001.

[8] Kuo, H and Lin, Y. (2008) “An H.264/AVC Full-Mode Intra-Frame Encoder for HD1080 Video”. In IEEE Int. Conf. on Multimedia & Expo, pages 1037-1040.

[9] Chuang, T-D., et al (2007) “Algorithm and Architecture Design for Intra Prediction in H.264/AVC High Profile”. In PCS 2007.

[10] Suh, K., et al (2005) “An Efficient Hardware Architecture of Intra Prediction and TQ/IQIT Module for H.264 Encoder”. In ETRI Journal, pages 511-524.

[11] Hamzaoglu, I., et al (2007) “An Efficient H.264/AVC Intra Frame Coder System Design”. In IFIP/IEEE Int. Conf. on Very Large Scale Integration, pages 1903-1911.

Efficiency Evaluation and Architecture Design of SSD Unities for the H.264/AVC Standard

Felipe Sampaio, Gustavo Sanchez, Robson Dornelles and Luciano Agostini
{fsampaio.ifm, gsanchez.ifm, rdornelles.ifm, agostini}@ufpel.edu.br

Universidade Federal de Pelotas - UFPel
Grupo de Arquiteturas e Circuitos Integrados - GACI

Abstract

This paper presents the design and evaluation of architectures that performs the SSD (Sum of Squared Differences) similarity criterion calculation. The comparison was made with other widely used criterion: the SAD (Sum of Absolute Differences). In order to compare the impact of both criteria in the coding process, a set of executions using the JM 16.0 reference software were performed. In these tests, SSD almost ever got a better video quality than SAD. Three architectures are proposed to perform SSD: (a) the first one uses a multiplexer, (b) the second uses a memory and (c) the last one uses a dedicated multiplier. One architecture to perform SAD is proposed to be compared with the architectures using SSD. Each solution was described in VHDL and synthesized to an Altera Stratix II FPGA. The video quality gain using SSD over the SAD encourages the use of SSD calculators even with a lower operation frequency when compared with an SAD implementation. In the best case and considering HDTV 1080p videos (1920x1080 pixels), it is possible to reach real time processing (30 frames per second) by putting 12 SSD calculators working in parallel.

1. Introduction

The H.264/AVC [1] is the newest and the most efficient video compressing standard. The main objective of H.264/AVC standard is to ally high compression rates with high quality of the compressed videos. In the Inter and Intra Frame Prediction [2], it is needed to define a similarity criterion which provides the information of how similar is the predicted block in relation to the current block that is being encoded. The similarity criterion chosen has a big and direct impact in the quality of the generated video and in the bitstream final size [3]. This paper considers two widely used metrics in video processing: the Sum of Absolute Differences (SAD) and the Sum of Squared Differences (SSD).

To obtain the similarity result between two blocks using SAD is necessary to perform the summation of the modules of the differences of each corresponding sample. The SSD value is calculated by the summation of the squared differences between each corresponding sample. The mathematical expressions that defines SAD and SSD are presented in Equation (1) and (2) , where the matrix O represents the original block and the matrix R represents the predicted block [3].

$$SAD(R,O) = \sum_{j=0}^{n-1}\sum_{i=0}^{n-1}|R(i,j) - O(i,j)| \quad (1)$$

$$SSD(R,O) = \sum_{j=0}^{n-1}\sum_{i=0}^{n-1}(R(i,j) - O(i,j))^2 \quad (2)$$

Both criteria direct the decision for the block that is most alike the original block. The difference is that using SSD, the chosen block is going to have more homogeneous values. That occurs just because the SSD assigns really big weight for large errors, whereas those differences are squared. Then, blocks with homogeneous values tend to have SSD values smaller than those blocks where the values are distant from the general average of the block values. The residual treatment path, which is formed by the operations of transforms and quantization, acts in a better way over the homogeneous error and it generates a video with more satisfactory objective quality [3].

This paper presents a comparison between these two similarity metrics widely used in video coding (SAD and SSD). The comparison was done considering the encoded video quality and the reached compressing rates. Three different implementations for SSD calculation were designed and compared to a basic SAD calculator implementation. The goal is to encourage the use of the SSD architectures for Inter or Intra Predictors as the similarity criterion in its decision modules. This way, the trade-off between the quality gain (using PSNR that is measured in dB units) and the hardware performance is analyzed.

This paper is organized as follows: Section 2 shows an evaluation of both criteria efficiency; Section 3 presents three different architectures designed to calculate the SSD; Section 4 discusses about the synthesis

results and the performance results reached and also compares them with a SAD calculator; Section 5 concludes this paper.

2. Software Evaluation

A total of 1440 executions were performed to evaluate the criteria. The H.264/AVC JM 16.0 reference software was used to allow this evaluation [4]. Several standard video sequences for the video compression community have been used as inputs for the JM executions. The target resolution was QCIF (352x288 pixels) with 30 frames per second as the frame rate.

The QP directly impacts in the video quality and in the bitstream size. This way, several typical QP values were used in the simulations in order to cover the most coding scenarios as possible.

The graphics and results presented in this paper consider the JM executions using the Full Search (FS) algorithm in the Inter Frame Prediction [5]. The FS executes the complete search in the delimited search area. This way, this algorithm always finds the optimal result considering the target search range.

The Fig. 1 present the graphic corresponding to the evaluation results with fifteen different QP values using search ranges of 16x16, 32x32 and 64x64 samples.

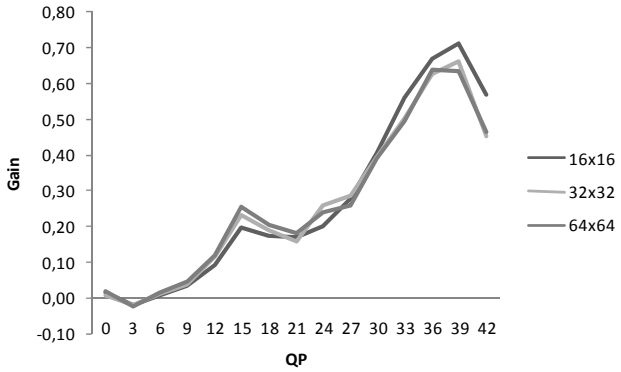


Figure 1 - SSD PSNR gain (in %) over the SAD for using 16x16, 32x32 and 64x6 search range

The graphic shows that the PSNR gains vary in small percentages. The best result, for example, is 0.71% in the scenario with search range of 16x16 samples. However, an unit variation in the PSNR value (express in dB) represents an expressive gain in the video quality. Then, it is possible to conclude that the use of the SSD criteria caused a real and important increase in the compressed videos quality.

The highest SSD gain was reached when high QP values are used. This is explained by the own characteristic of the SSD calculation, where homogeneous residual blocks generate smaller SSD results than heterogeneous residual information. The homogeneity is a property that increases the efficiency of the transforms and quantization process, allowing the generation of videos with better quality.

Typically, the QP range used is far from the minimum value, which is zero, assuming intermediated ranges. This fact justifies even more the employment of the SSD error metric in the decision modules in H.264/AVC encoders, since this criterion allows better results in higher QP ranges.

Considering the final bitstream size generated by the encoding process, the use of SSD impacts in little increases in the compressed video when compared with an SAD approach. The use of SSD metric directs the search algorithm to small block sizes, which will generate an increase in the final bitstream.

3. Designed Architectures

Three different architectures were designed in this paper to calculate SSD between two 4x4 blocks. The main objective of these designs was to find the best hardware solution to the SSD calculation. This is also to encourage the use of the designed architectures to calculate SSD in Intra Frame Prediction and Inter Frame Prediction instead of the SAD. It was also designed a SAD calculator with the goal of measure how efficient in terms of performance and hardware resources consumption is the SSD calculator compared to SAD calculator.

It is expected that the solutions that perform the calculation of SDD did not reach better processing rates than the SAD calculator. It is also expected that the SSD solutions uses more hardware than the SAD. However, this is a low price if considered the quality gain in the compressed videos. It is important to notice that we expected that the SSD designs will be able to reach a processing rate high enough to process high resolution videos in real time.

The adder tree (which is responsible for the summation of the differences) parallelizes all SAD/SSD operations for each 4x4 block position. Furthermore, the architecture remains the same regardless of using the sixteen SAD or SSD calculators. The only two differences would be in the SAD/SSD calculators and in the adders bit width. All the solutions assume input samples represented with 8 bits.

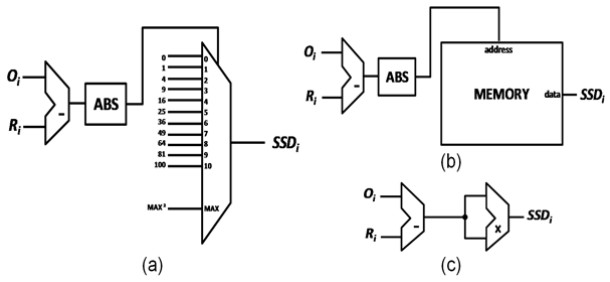


Figure 2 - Designed SSD Architectures: (a) SSD Multiplexer, (b) SSD Memory and (c) SSD Multiplier

3.1. Multiplexer SSD Architecture

This architecture realizes the operation of square through the implementation of a multiplexer. This multiplexer is selected by the module of the difference between the correspondent samples of two blocks. For each value inside the possible value interval the multiplexer selects the respective output for squaring the difference. Fig. 2(a) presents the designed architecture.

This architecture performance is limited by the critical patch formed by the subtract operation, the module operation (block ABS in Fig. 2) and the multiplexer combinational logic.

3.2. Memory SSD Architecture

This architecture consists in a read only memory which stores all the possible values that represents the square of the module values of the difference between the input samples. The error values are used to access the memory correct position. The output will receive the squared difference. Fig. 3 shows the designed architecture. In this case, the architecture performance depends of the time that is spent accessing the memory.

3.3. Multiplier SSD Architecture

This architecture uses one multiplier that is used to multiply the subtraction output by itself. This will produce the squared value. Fig. 4 presents the architecture diagram. The critical path of this architecture is determined by the architectural scheme used for the multiplier implementation. The advantage of this solution is that it is not needed an ABS operation in the subtract operation output, since two values with the same signal, when multiplied, always will generate positives values.

4. Results

The designed architectures were described in VHDL and synthesized targeting the Stratix II EP2S180F1508I4 FPGA [6]. Table 1 presents the synthesis results of the architecture for the FPGA device. The solution for the SSD that uses a memory to store all the square values achieves the best operation frequency among all designed SSD calculators, 275.71 MHz. This solution uses the on-chip memories available in the FPGA devices and this fact increases its performance, since this memory is located inside the FPGA. So, the communication between the data path and the memory is as fast as possible.

Table 1 - Synthesis Results

	ALUT	ALM	DLR	DSP Block	Mem. Bits	Freq. MHz
SSD Multiplexer	1173	900	496	-	-	197.71
SSD Memory	**640**	**390**	**257**	**-**	**65,536 (< 1%)**	**275.71**
SSD Multiplier	128	128	256	16 (2%)	-	190.29
SAD	257	193	384	-	-	391.85

All solutions used less than 1% of the device total capacity. In the best case, the architecture which uses multipliers consumes less logic elements than the SAD calculator. This is possible because the multipliers were mapped for multipliers structures available in the dedicated DSP blocks of the FPGA. The Memory SSD Calculator consumes memory bits, instead of Logic Elements (LEs) of the target FPGA device. On the other hand, the multiplexer solution for the SSD calculation is whole mapped into LEs.

The performance evaluation was performed considering the Inter Predictors' blocks that are needed to be processed by the SSD unities, since the Motion Estimation (one step into the Inter Prediction) represents the most complex module in the encoding path. This way, the worst case is considered. The FS algorithm using a search range of 32x32 samples was assumed. Considering this scenario and HDTV 1080p videos (1920x1080

pixels), there are 108,993,600 4x4 blocks that must be compared with original blocks in order to measure their similarity. This performance analysis is presented in Tab. 2.

Table 2 - Performance Results

Solution	Throughput (billions of samples)	HDTV frames/s	Num. of Calc. (30 fps HDTV)
SSD Memory	**4.41**	**2.529598**	**12**
SSD Multiplexer	3.26	1.813960	17
SSD Multiplier	3.10	1.745882	18

The best case of the designed SSD architectures is achieved by the memory approach, as it can be seen in Tab. 2. In this case, this architecture is able to process around 4.4 billion of samples per second. This throughput rate allows a processing of 2.53 HDTV frames per second, when using only one SSD unity to perform all calculations. Then, it is possible to reach real time processing (30 frames per second) using 12 SSD architectures working in a parallel way.

Considering a FPGA implementation of the SSD calculators designed in this work, all versions reached the performance that justify their employment in Intra or Inter Predictors decision modes with the goal to increase the PSNR results in the encoding process, when compared with an simple SAD calculator.

5. Conclusions

This paper presented the efficiency evaluation and the architectural design for the SSD (Sum of Squared Differences) similarity criteria on H.264/AVC video coding standard. A set of executions of the JM 16.0 Reference software proved the increase of compressed video quality using SSD as similarity criterion when compared to other criterion widely used, the SAD (Sum of Absolute Differences). This evaluation shown more specifically that for big QPs range, the SSD provides a more expressive gain than for small QPs. The quality criterion used was PSNR. Three architectures were designed for SSD calculators. They were described in VHDL and synthesized for a Stratix II Altera FPGA. As expected, the operation frequency reached by the designed SSDs calculators was worse than that reached when using SAD. Nevertheless, FPGA dedicated internal structures (multipliers, memories, etc) can be used to reach a high throughput solution for the SSD calculation. This way, considering the PSNR gain verified with the software evaluations and the hardware results, it is possible to conclude that the use of SSD in Inter or Intra Predictors is a good solution when the video quality is the main goal. Even with lower operation frequencies, the best solution for SSD unities designed in this work can deal with real time constraints by using several instances in order to parallelize the processing, which is typically done in related works in the literature.

6. References

[1] ITU-T Recommendation H.264/AVC (03/05): advanced video coding for generic audiovisual services, 2005.

[2] 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.

[3] I. Richardson, H.264 and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia. John Wiley&Sons, Chichester, 2003.

[4] Joint-Video Model (JM 16.0). Available at: http://iphome.hhi.de/suehring/tml/

[5] P. Kuhn. Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Boston: Kluwer Academic Publishers, 1999.

[6] Altera Corporation. “Altera: The Programmable Solutions Company”. Available at: www.altera.com.

Evaluating the Ginga Media Processing Component for Implementation of a Video Player

Marco Beckmann, Tiago H. Trojahn, Juliano L. Gonçalves, Lisane Brisolara, Luciano V. Agostini

{marcob.ifm, tiagot.ifm, juliano.lucas, lisane.brisolara, agostini}@ufpel.edu.br

Group of Architectures and Integrated Circuits – GACI
Federal University of Pelotas – UFPel
Pelotas – Brazil

Abstract

Nowadays the middleware of the Brazilian digital TV system (SBTVD) consists of two different paradigms; the procedural using Java language and named GingaJ, and the declarative one using the Nested Context Language and called GingaNCL. To provide the communication between these approaches the Ginga Commom Core (GingaCC) has created as part of the project Ginga Code Development Network. The GingaCC has a set of components, like components for video and audio reproduction. One of these components is the Media Processing, which provides functions directly involved in rendering video streams. This paper presents the development of a video player application using methods provided by the Media Processing component and discusses facilities to integrate modules using the FlexCM component model.

1. Introduction

To provide a transmission of the Brazilian digital TV system, a device called set-top-box is used. This equipment has a middleware implementation, called Ginga, whose function is providing an abstraction of the hardware work (transmissions protocols, for example) to developers of applications for digital TV [1].

The Ginga middleware can be subdivided into three main parts: Ginga Nested Context Language (GingaNCL) [2], GingaJ [3] and Ginga Commom Core (GingaCC). The GingaNCL subsytem is based on Nested Context Language (NCL) and uses a declarative approach. This subsystem has created by Pontifical Catholic Universty of Rio de Janeiro and has recommended by International Telecommunication Union (ITU) for use in Internet Protocol Television (IPTV) systems [4]. The GingaJ subsystem is based on the Sun Java language and uses a procedural approach. It has built by Lavid laboratory from the Federal University of Paraíba. GingaCC has as objective to make the bridge between the procedural and the declarative approaches provided by GingaJ and GingaNCL, respectively, as illustrated in Fig 1. To develop the GingaCC, a development effort has created, called Ginga Code Development Network (GingaCDN), which links 13 Brazilian universities and is headed by Federal University of Paraíba. Each university is responsible for building a number of components. In order to facilitate the integration of these components, they must be developed following a component model called FlexCM [5].

As part of this collaboration, our group has developed the Media Processing component. To validate this component, we have developed a video player built using its methods. This paper describes this development work and discusses the modules integration infra-structure. Beside the validation, this study has as objective to evaluate the use of FlexCM to turn easier the components integration and also create a more pleasant interface to test the functions offered by the Media Processing component.

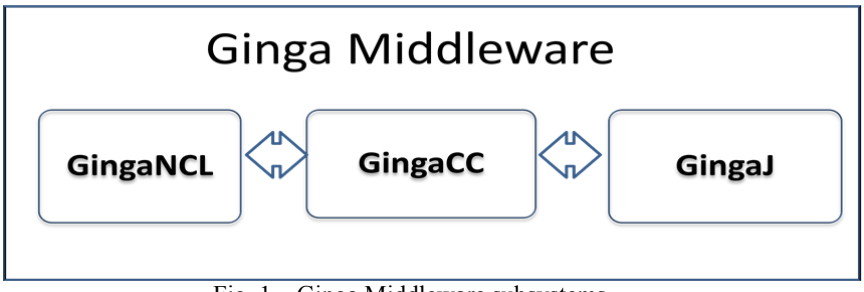


Fig. 1 – Ginga Middleware subsystems

The remaining of this paper is organized as follows. Section 2 presents a background for the discussion, presenting the Media Processing component and FlexCM component model. Section 3 describes the video player development used as case study. Section 4 concludes the paper and points out directions for future work.

2. Background

2.1. Media Processing Component implementation

The Media Processing Component [6] has developed using C++ language and the libVLC library [7] and can be responsible for handle video and subtitles streams. For that reason, our implementation also follows the Java Media Framework (JMF) version 1.0 [8], which is an API that specifies an architecture to synchronize and control audio, video and other time-based data structures like subtitles.

The current Media processing implementation provides a set of basic functions to other components, that are:

- Support resources allocation and media flow control;
- Media stream receiving and decoding;
- Load, select and show subtitles;
- Screenshots;
- Provide video stream information, like total duration, actual time position, resolution and frame rate per second;
- Support for streaming video using Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), User Datagram Protocol (UDP) and Real-Time Transfer Protocol (RTP).

2.2. FlexCM

The components provided by the GingaCDN, including the Media Processing, have built using the FlexCM component model. The use of this model garantes the integration among the modules and facilitates the distribued development of modules, what is essential for the GingaCDN project.

Following this model, each component must inform about interface(s) provided to other components as well as required interface(s) (endorsement by another module). Besides, every component must have two files each:

- Architecture file: Describes the data for execution, including the path for the dynamic library and the component unique identification code.
- Registry file: Specifies the conections made by the component, considering both provided as required interfaces and using the unique identification of components.

The use of this model obliges developers to adopt well-defined interfaces, guarantying the integration among modules. After these interfaces are defined, the integration is done by the FlexCM infrastructure, which reads the information from Registry file and connects the modules in run-time.

3. Case study: Video Player Application

The player application, written in C++, uses the methods provides by the Media Processing Component, detailed in [6]. This system was run on the operating system Ubuntu 9.10, using the FlexCM to make the connection between the video player application and the module Media Processing. When the application is started, a list of functions provided by the application is listed on the command prompt. In order to request the execution of an operation, the user should type the name of the desired function.

The application provides the following functions, divided in six groups in the illustration from Fig 2, which are: manipulate video, video information, video dimension, manipulate subtitles, screenshot, and exit.

- Manipulate Video:
  a) play: Tries to set the player status to the state “play”. As pre-condition for this operation, the player needs to be in the states “stop” or “pause”. This operation requires also a correct allocation of player resources provided by the method. “load”
  b) pause: Setting the player to the state “pause”. There is no effect for the activation of this operation if the player is not playing a video;
  c) stop: It stops the reproduction of an allocated player and also clear charged resources;
  d) load: charges the video from the folder selected by the user. If a video is playing, the method “load” uses the method “stop” and deallocates resources;
- Video Information:
  a) fps: Returns the frame rate of the current video. This return value can change sometimes if the video stream is encoded with variable frame rate (VRF);
  b) time: Gives the time of the current frame in microseconds;
  c) duration: Informs the total duration of the current allocated video stream when available. Some internet streaming data and TV broadcast videos do not provide this information;
  d) info: Provides miscellaneous information such as file name, artist, album, etc;
- Video Dimensions:

a) dimension: Returns the integer representations of height and width resolutions of the current allocated video;
b) getscale: informs the actual scale size of the video, showing a value between 0 and 1, equivalent to the window size in percentage;
c) setscale: modifies the scale size, with a value between 0 and 1, equivalent to the window size in percentage;

- Manipulate Subtitles:
  a) addsubtitle: Adds a new libvlc compatible subtitle to the current allocated video stream. It supports a wide range of subtitles formats, like the basic SubRip (SRT), the Advanced Substation Alpha (ASS), and the ISDB-TB standard subtitle format;
  b) numsubtitle: Returns the number of current subtitles available for selection or 0 if no one is present;
  c) getsubtitle: Returns the subtitle ID number. There is not effect if no subtitle is present in the stream or if the player is not allocated;
  d) setsubtitle: Sets the subtitle ID specified to be played with the video. The subtitle must be a valid subtitle ID number and the player should be allocated before the invocation of "setsubtitle" operation;
- Screenshot
  a) screenshot: Takes a screenshot of the current frame being displayed by the player. The file created uses lossless compression of the Joint Photographic Expert Group (JPEG) format. There is no effect if the player is not in the state "Play" or "Pause";
- Exit:
  a) exit: Calls functions used to deallocate resources;

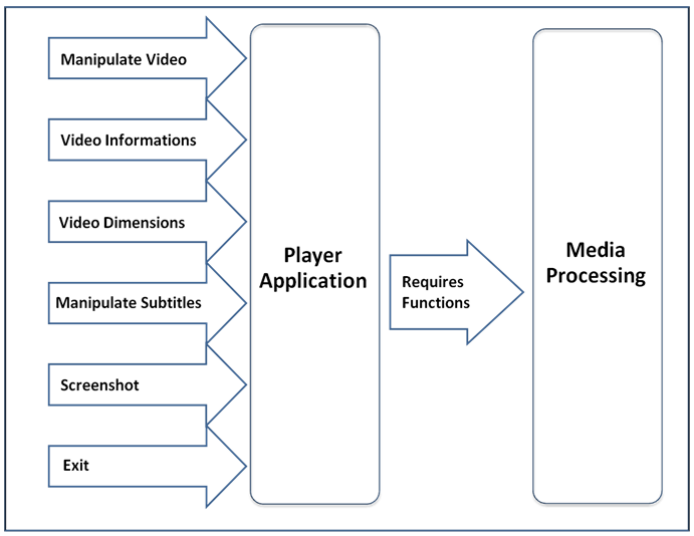


Fig. 2 – Application Functions and its interaction with Media Processing

Using the corresponding command, the Player Application calls functions from the module Media Processing, as illustrated in Fig. 2. The link between the two modules is provided by the FlexCM component that promotes the connection, as illustrated in Fig. 3.

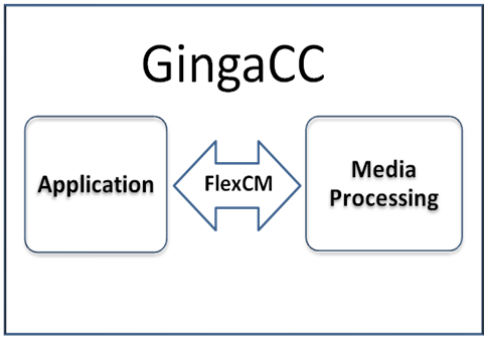


Fig. 3 – Communication between modules made by FlexCM

4. Conclusions and Future Work

This paper presents a case study of implementation of a video player using the Ginga Media Processing component, in which the FlexCM component model is adopted to facilitate the integration between both modules. In this way, this study allowed us to evaluate the integration facilities provided by the FlexCM model and also to validate the functions provided by the Media Processing component as well. This validation was facilitated, because using the application we can directly test the provided functions without any changes to the source code.

As future work, we plan to develop a new version of the player in which a graphical interface will be used to improve the application usability. The Player Application also could be used as the video player for TV showing video recorded on the set-top-box (if they offer this function), for example. Another possible use for the video player is the playing of videos from the internet, using an HTTP protocol, if it is offered by the set-top-box. While this hardware equipment is not available, we plan to test the player running in an embedded platform to check its portability and required resources.

5. References

[1] S.D.J. Barbosa, & L.F.G. Soares, “TV digital no Brasil se faz com Ginga”. Kowaltowski e K. Breitman, Atualizações em Informática 2008. Rio de Janeiro, RJ:Editora PUC-Rio. pp.105-174

[2] L.F.G. Soares, R.F. Rodrigues, M.F. Moreno, “Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System”. Journal of The Brazilian Computer Society, SBC – Agosto de 2007

[3] G.L.S. Filho, L.E.C. Leite, C.E.C.F. Batista, “Ginga-J: The Procedural Middleware for the Brazilian Digital TV System”. Journal of the Brazilian Computer Society. No. 4, Vol 13. P47-56. ISSN: 0104-65000, Porto Alegre, RS, 2007.

[4] ITU, “ITU-T Recommendation H.761: Nested Context Language (NCL) and Ginga-NCL for IPTV”, 2009

[5] .M. Filho, et al, "FLEXCM - A Component Model for Adaptive Embedded Systems", in Proc. COMPSAC (1), 2007, pp.119-126.

[6] T. H. Trojahn J. L. Gonçalves; J. C. B. Mattos; L. S. Da Rosa Junior; L. V. Agostini. “A Media processing Implementation using Libvlc for the Ginga Middleware”. The 5th International Conference on Future Information Technology (FutureTech), 2010, Busan, Korea.

[7] VLC Documention 1.0.0 [Online] Avaliable: http://www.videolan.org/developers/vlc/doc/doxygen/html/index.html [Acessed: Mar. 19, 2010].

[8] Sun Microsystems, Silicon Graphics, Intel Corporation. “1.0 Programmers guide” May. 11, 1998. [Online] Avaliable: http://java.sun.com/javase/technologies/desktop/media/jmf/1.0/guide/index.html [Acessed: Mar. 17, 2010].

An Implementation of Media Processing Component Using LibVLC Library for the Ginga Middleware

Tiago H. Trojahn, Juliano L. Gonçalves,
Leomar S. da Rosa Junior, Luciano V. Agostini
{tiagot.ifm, juliano.lucas, leomarjr, agostini}@ufpel.edu.br

Group of Architectures and Integrated Circuits – GACI
Federal University of Pelotas – UFPel
Pelotas – Brazil

Abstract

The Brazilian middleware for Digital TV, known as Ginga, is currently divided in two subsystems: the declarative, named Ginga Nested Context Language (Ginga-NCL), and the procedural, called Ginga-J. According to recent researches there is an increase need of a unique procedural and declarative middleware. To fulfill this need, a project to develop a unique, modularized and fully integrated middleware was created with the name of Ginga Code Development Network (GingaCDN), headed by Federal University of Paraiba. In this project a common core will be developed, named Ginga Common Core (GingaCC), to provide a set of methods, like video reproduction and channel tuning, to be used by both GingaNCL and GingaJ. The Media Processing is one of the components of the GingaCC and the main focus of this work is to show an implementation of that component using the LibVLC library and the FlexCM component model.

1. Introduction

The middleware for the Brazilian Digital Television System (SBTVD), named Ginga, is an effort to create a middleware using both a declarative, known as Ginga Nested Context Language (GingaNCL), and a procedural environment, known as Ginga-J. The GingaNCL is based in the Nested Context Language (NCL), a declarative language developed by PUC-Rio and was recommended by International Telecommunication Union (ITU) for use in Internet Protocol Television (IPTV) systems [1]. The Ginga-J is a procedural environment built to support the Sun Java language.

Actually, there is two available Ginga environments to a developer build his applications: the GingaNCL, an upgrade from a low-cost declarative middleware named Maestro built in 2001 by Moreno [2]; and the OpenGinga [3], an upgraded version of FlexTV [4], used nowadays as reference for the procedural middleware. Unfortunately, applications developed in GingaNCL cannot run in OpenGinga, and vice-versa.

A common core, named Ginga Common Core (GingaCC) is being developed to provide compatibility between GingaNCL and Ginga-J, forming a unique middleware for SBTVD. For this task, it was created the Ginga Code Development Network (GingaCDN), a network of 13 Brazilian universities, coordinated by Federal University of Paraiba (UFPB), where each university is responsible for the development of a pre-determined number of components. The Media Processing is one of these components and has a main role in GingaCC: the video decoding. The Ginga middleware being developed by GingaCDN project is presented in Fig. 1.

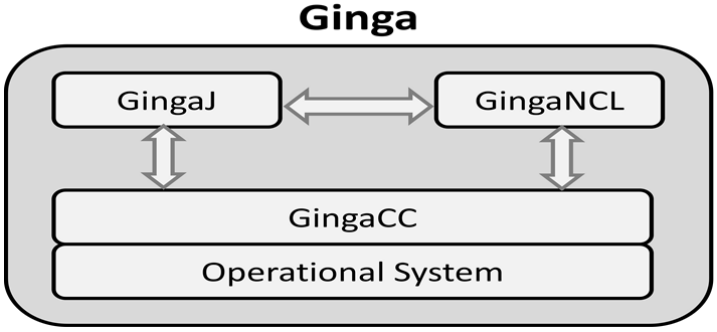


Fig. 1 – The Ginga middleware by GingaCDN project.

2. The Media Processing Component

The Media processing component is one of the main modules of GingaCC, and it is directly involved in rendering video streams. The sub-components involved in this task are described below:

- Tuner – Component responsible for the channel tuning and the capture of the Transport Stream that is transmitted in the channel. The tuner output is redirected to the Information Service;
- Information Service – Component responsible to analyze the Transport Stream, to obtain the stream information, and to add some relevant information to reproduction;
- Demux – Component responsible to demux the streams, which compose the Transport Stream, using the information retrieved from Information Service component. The Demux output (video, audio and subtitle streams) is sent to Media processing;
- Media processing – Component responsible to decode the stream received from Demux component. The output is sent to Graphics component;
- Graphics – The Graphics component is responsible to control and to show the decoded video in the display. This is the last component involved directly in video player.

The Media processing component was developed in C++ language using the libVLC library. To being integrated with other components, the FlexCM [5] model component was used. The Media processing follows the Java Media Framework (JMF) version 1.0 [6]. The JMF is an API that specifies an architecture to synchronize and to control audio, video and other time-based data structures like subtitles. The 1.0 version specifies the media reproduction known as "Java Media Player". The first Media processing implementation is responsible for handle video and subtitles streams. To make possible the distributed development, all GingaCDN components are being implemented using encapsulation. Only an interface is provided to other components. This approach makes easier the distributed development and enables a fast error correction. Fig. 2 illustrates the connections between the Media processing and the components directly connected to him, the Demux and the Graphics components.

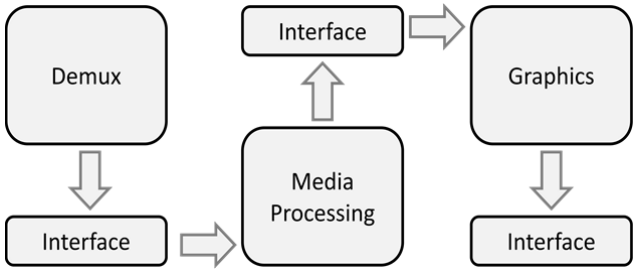


Fig. 2 - Component connections and data flow to media rendering.

The current version of Media processing provides a set of basic functions to other components. A high level description of it is described below:

- Resources allocation and media flow control;
- Load, select and show subtitles;
- Media stream receiving and decoding;
- Screenshots;
- Provides video stream information, like total duration, actual time position, resolution and frame rate per second;
- Support streaming video using Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), User Datagram Protocol (UDP) and Real-Time Transfer Protocol (RTP).

The current version of the component was implemented only to provide the desired functionalities. Wasn't applied any optimizations to improve the performance, reduce memory cost or processor usage, in the implementation.

2.1. LibVLC

LibVLC is a graphic library developed by VideoLAN under GNU General Public License (GPL) version 2. The choice for this library to implement the Media processing component is due to its large list of features, like:

- Compatibility with various media formats, including the standard H.264/AVC (defined in MPEG-4 Part 10), the audio standard MPEG Layer 2, MPEG Layer 3 (MP3), MPEG-4 part 3 (Advanced Audio Coded - AAC) and AC3;
- Portability to a wide range of operating systems, like Microsoft Windows, GNU Linux, Mac OS, BeOS and FreeBSD;

- Support to several video outputs, like DirectX, OpenGL, X11, Xvideo, SDL and Frame Buffer;
- Written in C language, offering a high performance needed for a Media processing.
- The library operate in a multithread environment, creating, joining and destroying threads when some events occurs, like the start (play) of the video stream.

2.2. FlexCM

The GingaCDN components were developed using the FlexCM component model. Each FlexCM component must specify the required interfaces and the interfaces provided to other components. The responsibility for connecting the components is done by FlexCM during the execution time.

Each component implementation has to specify two archives:

- Architecture: This file describes the essential data for execution, like the path to the dynamic library of each component and a unique identification for the component;
- Registry: Specifies which connections are used by the component, using the unique identification numbers defined in the component implementation.

This methodology helps the distributed development and also guarantees an easy integration process. The version used in Media processing implementation was the v0.2.

3. Tests and Results

The current version of Media processing was submitted to a video evaluation set in order to investigate the processor and memory usage in real use cases. The evaluation computer was a Intel Core 2 Duo 6300 (1.86Ghz) processor with 2 Gigabyte (GB) of RAM running the Ubuntu 9.10 operating system.

Three tests were performed for each sample, and the samples were captured every second using the Procps application, for a time of one minute. The component evaluated was compiled using the version 4.4.1 of GNU GCC compiler without any optimizations available in the compiler and used a default build of the libVLC, to evalue the component in an high compatible environment.

The samples were collected from three videos available in two different resolutions. Both videos are progressive (p) and follows the H.264/AVC standard. The results are being show in minimum (Min.), maximum (Max.) and average (Av.). The memory consumption is shown in Megabytes (MB) and the processor use in percentage.

The data includes the FlexCM loading platform, the libVLC objects allocation, the libVLC default demux and the X11 default rendering component for GNU Linux. The data for processor usage show the average values of all cores of the processor.

Three videos were used as benchmark. The STS-117 Launch video, named STS-117 [7]. The Speed and the Stormchasers videos [8]. The detailed description for 1080p videos can be viewed in Tab. 1. The 720p video details can be found in Tab. 2. Results obtained for 720p videos are presented in Tab. 3, and results obtained for 1080p videos are illustrated in Tab. 4.

Tab. 1 – Video Information: Size (in MB), the total duration (minutes and seconds), resolution, frame rate per second and container for 1080p videos.

Video Name	1080p				
	Size	Duration	Resolution	FPS	Container
Speed	128	2:07	1440x1080	23.98	WMV
Stormchasers	90.8	1:31	1440x1080	23.98	WMV
STS-117	67.4	1:08	1920x1080	29.97	WMV

Tab. 2 – Video Information: Size (in MB), the total duration (minutes and seconds), resolution, frame rate per second and container for 720p videos.

Video Name	720p				
	Size	Duration	Resolution	FPS	Container
Speed	96.5	2:07	1280x720	23.98	WMV
Stormchasers	68.8	1:31	1280x720	23.98	WMV
STS-117	35.2	1:08	1280x720	23.98	WMV

Tab. 3 – Memory cost and processor use for 720p videos.

Video Name	Memory Use			CPU Use		
	Min.	Max.	Av.	Min.	Max.	Av.
Speed	65.88	69.20	68.93	30.2	67	33.34
Stormchasers	67.51	70.27	69.47	31	78	32.85
STS-117	64.74	73.08	69.75	20.6	52	25.94

In terms of memory use, the data for 720p show a difference of 1.18% between Speed and the STS-117, the video with the greatest memory use. The low memory usage is important in low-cost equipment, like a set-top-box, to reduce the total price of the system. Moreover, the results show a reduced CPU use, important to multitask environments and also to a reasonable energy use for embedded systems.

Tab. 4 – Memory cost and processor use for 1080p videos.

Video Name	Memory Use			CPU Use		
	Min.	Max.	Av.	Min.	Max.	Av.
Speed	83.92	86.78	86	49.3	80.5	52.34
Stormchasers	84.17	87	86.12	42	84	47.95
STS-117	99.81	101.78	100.84	58.3	83	63.72

The memory use of each video between the lowest and the greatest are of 16.2%. This little variation shows a stability to reproduce a large amount of videos in that resolution. The increase CPU use of STS-117 can be explained with the video resolution (1920x1080), wider than the other two 1080p videos, and the FPS of 29.97, larger than the 23.98 of other two videos, requiring more CPU use to decode and render the frames in real-time.

Finally, it is important to say that it was not possible to compare the performance with other Media processing implementations at this time, because there is a lack of other components following the same specifications.

4. Conclusion and Future Works

The componentized version of Ginga middleware, supporting both procedural and declarative application, will play a main role for Digital TV applications developers because this kind of development will be more flexible. The developer can choose between procedural and declarative environment without any concern. This facilitated the development of applications for Digital TV, strengthening the technology at national level. Contributing to that, Brazil is beginning to exporting it to other countries in South America such as Peru, Chile, Argentina and Venezuela [9] which will increase the national demand for qualified professionals to work with this new technology.

Regarding future works, it is necessary to add support for audio streams and some other related features, like volume control, audio stream selector and so on. It is also intended to make performance comparisons with other Media processing component implementations and tests with other machine configurations, like to notebooks and computers with low memory capacity.

5. References

[1] ITU, “ITU-T Recommendation H.761: Nested Context Language (NCL) and Ginga-NCL for IPTV”, 2009.

[2] M. F. Moreno, “A declarative middleware for Interactive Digital TV Systems”. M.S Thesis, PUC-Rio, Rio de Janeiro, 2006. p.105. (In Portuguese).

[3] Applications Laboratory of Digital Video, Telemidia. “OpenGinga – Middleware reference implementation of Brazilian Digital TV”, openginga.org. 2008. [Online] Available: http://www.openginga.org/index.html. [Accessed: Mar. 12, 2010].

[4] L. E. C. Leite, et al, “FlexTV, a proposal for middleware architecture for Brazilian Digital TV System”. Journal of Computer Engineering and Digital Systems, 2005, vol.2, pp.30-49. (In Portuguese).

[5] S.M. Filho, et al, "FLEXCM - A Component Model for Adaptive Embedded Systems", in Proc. COMPSAC (1), 2007, pp.119-126.

[6] SUN MICROSYSTEMS, Silicon Graphics, Intel Corporation. “1.0 Programmers guide” May. 11, 1998. [Online] Available: http://java.sun.com/javase/technologies/desktop/media/jmf/1.0/guide/index.html [Accessed: Mar. 11, 2010].

[7] NASA, "NASA High Definition Video", nasa.gov, Dec. 16, 2009. [Online]. Available: http://www.nasa.gov/multimedia/hd/index.html. [Accessed: Mar. 11, 2010].

[8] MICROSOFT, "WMV HD Content Showcase", microsoft.com, 2004. [Online]. Available: http://www.microsoft.com/windows/windowsmedia/musicandvideo/hdvideo/contentshowcase.aspx. [Accessed: Mar. 11, 2010].

WIKIPEDIA, Mar. 10, 2010. [Online] Available: http://en.wikipedia.org/wiki/ISDB.

A Media Processing Implementation Using Xine-Lib for the Ginga Middleware

Rafael L. Pereira, Juliano L. Gonçalves, Julio C. B. Mattos, Luciano V. Agostini
{rpereira.ifm, juliano.lucas, julius, agostini}@ufpel.edu.br

Group of Architectures and Integrated Circuits – GACI
Federal University of Pelotas – UFPel
Pelotas – Brazil

Abstract

The Brazilian middleware for Digital TV, known as Ginga, is currently divided in two subsystems: the declarative part, named Ginga Nested Context Language (Ginga-NCL), and the procedural part, named Ginga-Java (Ginga-J). A project was created, named Ginga Code Development Network (GingaCDN), to develop a modularized and fully integrated middleware combining the two subsystems: Ginga-NCL and Ginga-J. One of the main objetives of this project is to implement a common core, named Ginga Common Core (GingaCC), to provide a set of methods to manipulate videos and channel tunning to be used by both GingaNCL and Ginga-J. This work shows the implementation of the Media Processing component (one of the components of the GingaCC) using the Xine-Lib and the FlexCM component mode.

1. Introduction

Ginga - the middleware for the Brazilian Digital Television System (SBTVD) – is an effort to create a middleware using two environments, the declarative environment, named Ginga Nested Context Language (GingaNCL)[1] and the procedural environment, known as Ginga-Java (Ginga-J)[2]. The Ginga-J is a procedural system to support programs written in Java Language. On the other hand, GingaNCL is based in the Nested Context Language (NCL), a declarative language developed by PUC-Rio and recommended by International Telecommunication Union (ITU) for use in the Internet Protocol Television (IPTV)[3] systems.

Nowadays, there are two environments to develop applications for the SBTVD: GingaNCL, an upgrade from a low-cost declarative middleware named Maestro [4] and OpenGinga[5], an upgraded version of FlexTV[6] used as reference for procedural middleware. Applications developed for GingaNCL can not run in OpenGinga, and vice-versa.

A commom core, named Ginga Common Core (GingaCC), is being developed to provide compatibility between GingaNCL and Ginga-J, producing a unique middleware for SBTVD. For this purpose, Ginga Code Development Network (GingaCDN) was created. This development network is composed by 13 Brazilian universities, coordinated by Federal University of Paraíba (UFPB), where each university are responsible to development certain number of components. The Media Processing is one of GingaCC components and it is responsible for the video processing. Figure 1 presents communication between the subparts and GingaCC.

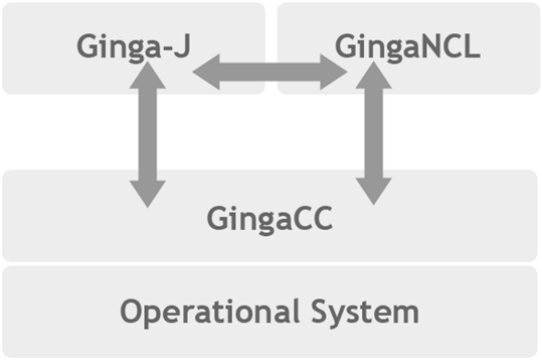


Fig. 1 – The Ginga middleware by GingaCDN project

The aim of this paper is to present the Media Processing implementation used in GingaCC. During this development process, the FlexCM is the model component used to make the middleware modularized and Xine-lib is a multimedia library used to implement this media processing version. The paper also shows some results in terms of CPU performance and memory usage.

This paper is organized as follows: Section 2 presents the Media Processing component functions and the resources used to implement this component (Xine-Lib and FlexCM). Section 3 describes the tests and results and section 4 concludes the paper and presents some future works.

2. The Media Processing Component

The Media processing component is one of the main components of GingaCC directly involved in rendering video streams. This component is involved in several tasks described below:

- Tuner – component responsible for the channel tuning and the capture of the Transport Stream that is transmitted in the channel. The tuner output is redirected to the Information Service;
- Information Service – component responsible to analyze the Transport Stream in order to obtain the stream information and add some relevant information to reproduction;
- Demux – component responsible for demux the streams, which is composed by the Transport Stream, using the information retrieved from Information Service component. The Demux output (video, audio and subtitle streams) is sent to Media processing;
- Media processing – component responsible to decode the stream received from Demux component. The output is sent to Graphics component;
- Graphics – The Graphics component is responsible to control and to show the decoded video in the display. This is the last component involved directly in video player.

The Media Processing componet was developed in C++ language using the Xine-lib[7]. To be integrated with other GingaCC components, the FlexCM [8] model component was used. The GingaCDN components are being implemented using encapsulation and only the interface is provided to other components. This approach makes easier the component integration and testing.

The Media processing implementation follows the Java Media Framework (JMF) version 1.0 [9]. The JMF is an API that specifies the architecture to synchronize and control audio, video and other time-based data structures like subtitles. The JMF 1.0 version specifies the media reproduction known as “Java Media Player”. The connections between the Media processing, the Demux and the Graphics components are presented in Figure 2.

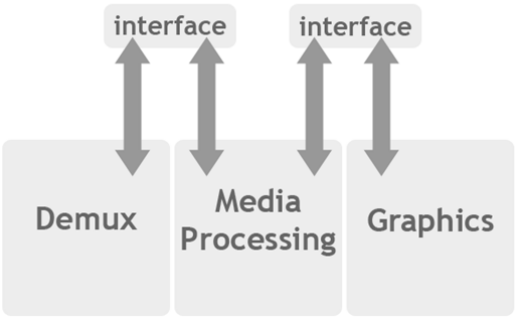


Fig. 2 - Component connections.

The implemented Media processing (current version) provides a set of basic functions to other components. These functions are:

- Resource allocation and media flow control;
- Load, select and show subtitles;
- Media stream receiving and decoding;
- Screenshots;
- Provide video stream information, like total duration, actual time position, resolution and frame rate per second;
- Support streaming video using Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), User Datagram Protocol (UDP) and Real-Time Tranfer Protocol (RTP).

2.1. Xine-Lib

Xine-lib is a free high-performance (GNU General Public License - GPL), portable and reusable multimedia playback engine[10]. The choice for this library to implement the Media processing component is due to the following features:

- Compatibility with several media formats, including the standard H.264/AVC (defined in MPEG-4 Part 10), the audio standard MPEG Layer 2, MPEG Layer 3 (MP3), MPEG-4 part 3 (Advanced Audio Coded - AAC) and AC3;
- Portability to a wide range of operating systems, like GNU Linux and FreeBSD;
- Written in C Language, providing high performance for a media processing implementation.

2.2. FlexCM

The GingaCDN components were developed using the FlexCM component model. Each FlexCM component must specify the required interfaces and the interfaces provided to other components. The responsibility for connecting the components is done by FlexCM during the execution time.

Each component implementation has to specify two files:

- Architecture: This file describes the essential data for execution, like the path to the dynamic library of each component and a unique identification for the component;
- Registry: Specifies which connections are used by the component, using the unique identification numbers defined in the component implementation.

This methodology helps the distributed development and guarantees an easy integration process. The version used in Media processing implementation was the 0.2.

3. Tests and Results

The current version of Media processing was submitted to a video evaluation in order to investigate the processor and memory usage in real use cases. The computer used to do the tests was an Intel Core 2 Duo 6300 (1.86Ghz) processor with 2 GigaByte (GB) of RAM running the Ubuntu 9.10 operating system.

Three tests were performed for each sample, and the samples were captured every second using the Procps application, for one minute. The component evaluated was compiled using the GNU GCC compiler version 4.4.1 without any optimizations available.

The samples were collected from three videos available in two different resolutions. Both videos are progressive (p) and follow the H.264/AVC standard. The results are being shown in minimum (Min.), maximum (Max.) and average (Av.). The memory consumption is shown in MegaBytes (MB) and the processor usage in percentage. The data includes the FlexCM loading platform, the xine-lib objects allocation, the xine-lib default demux and the X11 default rendering component for GNU Linux.

Three videos were used as benchmark: the STS-117 Launch video (STS-117), the Speed video and the Stormchasers video. Table 1 shows the detailed information about 1080p videos and Table 2 shows the detailed information about 720p videos.

Tab. 1 – Video Infomation: Size (in MB), the total duration (minutes and seconds), resolution, frame rate per second and container for 1080p videos

Video Name	**1080p**				
	Size	**Duration**	**Resolution**	**FPS**	**Container**
Speed	128	2:07	1440x1080	23.98	WMV
Stormchasers	90.8	1:31	1440x1080	23.98	WMV
STS-117	67.4	1:08	1920x1080	29.97	WMV

Tab. 2 – Video Infomation: Size (in MB), the total duration (minutes and seconds), resolution, frame rate per second and container for 720p videos

Video Name	**720p**				
	Size	**Duration**	**Resolution**	**FPS**	**Container**
Speed	96.5	2:07	1280x720	23.98	WMV
Stormchasers	68.8	1:31	1280x720	23.98	WMV
STS-117	35.2	1:08	1280x720	23.98	WMV

The test results are presented in Table 3 and Table 4 for 720p and 1080p videos, respectively. For 720p videos, there is not a significant difference between the three analyzed videos in terms of memory use. The memory usage variation was only 0.3% between the highest average and the lowest average, while the difference between a highest maximum use and the lowest minimum use of memory was only 1.4%.

The results in terms of CPU usage show a difference that can be noticed in the video speed. There is a difference between the minimum and the maximum CPU usage however on average the three videos shown a low cost, which is important to multitask environment and also to low energy usage for embedded systems.

The results for 1080p videos show a expected increase in terms of memory usage and CPU comparing to 720p videos. However, the results show a good performance in terms of memory and CPU usage in both 720p and 1080p videos making the Media Processing using Xine-lib feasible for a wide range of video resolutions.

Tab. 3 – Memory Cost and Processor Use for 720p vídeos.

Video Name	Memory Use			CPU Use		
	Min.	Max.	Av.	Min.	Max.	Av.
Speed	44.85	45.06	45.05	32.9	69	34.52
Stormchasers	44.76	45.03	44.97	30.8	49	32.06
STS-117	44.44	44.91	44.88	22.2	41	26.92

Tab. 4 – Memory Cost and Processor Use for 1080p vídeos.

Video Name	Memory Use			CPU Use		
	Min.	Max.	Av.	Min.	Max.	Av.
Speed	60.7	60.75	60.75	51.5	62.7	53.24
Stormchasers	60.33	60.65	60.52	44.7	63	48.77
STS-117	52.03	73.05	72.89	60	77	65.80

4. Conclusion and Future Works

The componentized version of Ginga middleware shows a flexible solution for developers to support declarative and procedural applications. Moreover, this componentized version enables to build a customized middleware for different solutions. This paper presented a Media Processing component implementation based on Xine-lib. The paper also showed results in terms of CPU performance and memory usage.

Regarding future work, it is necessary to add other features, like audio support, volume control, audio stream selector and so on. It is also intended to do performance comparisons with others Media processing component implementations and tests with other computer configurations, e.g. computers with low memory capacity.

5. References

[1] L. F. G. Soares; R. F. Rodrigues; M. F. Moreno, “Ginga-NCL: the declarative environment of the Brazilian digital TV Sytem”. Journal of the Brazilian Computer Society, 2007, pp.37-46

[2] G. L.S. Filho; L. E. C. Leite; C. E. C. F. Batista. Ginga-J: The procedural middleware for the Brazilian digital TV system”. Journal of the Brazilian Computer Society, 2007, vol 12, pp.47-56

[3] ITU, “ITU-T Recommendation H.761: Nested Context Language (NCL) and Ginga-NCL for IPTV”, 2009.

[4] M. F. Moreno, “A declarative middleware for Interactive Digital TV Systems”. M.S Thesis, PUC-Rio, Rio de Janeiro, 2006. p.105. (In Portuguese)

[5] Aplications Laboratory of Digital Video, Telemidia. “OpenGinga – Middleware reference implementation of Brazillian Digital TV”, openginga.org. 2008. [Online] Avaliable: http://www.openginga.org/index.html. [Acessed: Mar. 12, 2010].

[6] L. E. C. Leite, et al, “FlexTV, a proposal for a middleware architecture for Brazilian Digital TV System”. Journal of Computer Enginering and Digital Systems, 2005, vol.2, pp.30-49. (In Portuguese).

[7] The Xine Project, “A free vídeo player”, [Online]. Available: http://www.xine-project.org/home [Acessed: Mar. 23, 2010]

[8] S.M. Filho, et al, "FLEXCM - A Component Model for Adaptive Embedded Systems", in Proc. COMPSAC (1), 2007, pp.119-126.

[9] Sun Microsystems, Silicon Graphics, Intel Corporation. “1.0 Programmers guide” May. 11, 1998. [Online] Avaliable: http://java.sun.com/javase/technologies/desktop/media/jmf/1.0/guide/index.html [Acessed: Mar. 11, 2010].

[10] The Xine Project, “About xine”, [Online]. Available: http://www.xine-project.org/abou.t [Acessed: Mar. 24, 2010]

Proposal of a Diamond Search Design with Integrated Motion Compensation for a Half/Quarter-Pixel H.264/AVC Motion Estimation Architecture

[1]Gustavo Freitas Sanchez, [1,2]Robson Sejanes Soares Dornelles, [1]Luciano Volcan Agostini
{gsanchez.ifm, rdornelles.ifm,agostini}@ufpel.edu.br

[1]Federal University of Pelotas
[2]Federal University of Rio Grande do Sul

Abstract

This paper proposes a new way to improve the encoded digital video quality from Diamond Search Algorithm through the use of the small diamond search pattern for the previous iteration and so getting a better video quality than the regular Diamond Search. This proposed architecture also generates the samples necessary to perform the half and quarter pixels interpolation. This is useful to perform a search of a fractional vector with a quarter pixel precision at most and then accomplishing a better encoded video quality. The architecture also performs the motion compensation process during the motion estimation. It makes the video coding faster because it allows skip a step.

1. Introduction

With the growing demand of digital videos with high quality and high compression rate, the effort in research is increasing and then a lot of video coding standards are emerging. In this context the H.264/AVC [1] arises as the newest and most efficient video coding standard.

The main goal of the H.264/AVC standard is to ally high video quality to high compression rates. For doing this, it is necessary to perform extremely complex algorithms. This high complexity does not allow, at least in the current technology, software video coding solutions when it is needed to deal with real time (24 to 30 frames per second), mainly when encoding and decoding digital videos with high resolution (like HDTV 720p and 1080p). This way, hardware solutions are really necessary.

This work is part of the Brazilian effort in hardware solution for the Brazilian System of Digital Television (SBTVD) [2], since the H.264/AVC was chosen to be the standard of the SBTVD.

To encode digital videos in an efficient way, the video redundancies must be explored. There are 3 types of redundancies: temporal (similarity between neighboring frames), spatial (similarity within the same frame), entropy (redundancy in the binary encoding).

Fig. 1 shows the block diagram of the encoder, where the current frame is the frame that will be encoded, and the reference frames are already encoded. The motion estimation (ME) is responsible to exploit the temporal redundancy and is the focus of this work. The motion compensation is used to reassemble the encoded block and is also focus of this work. The intra prediction exploits the spatial redundancy and entropy encoder exploits the entropy redundancy. The standard also has a block of transform and quantization which is responsible for increasing compression on digital videos in exchange of a small loss of quality. The inverse transform and inverse quantization are used to reassemble the frame and the reduction filter effect of block is required by the standard.

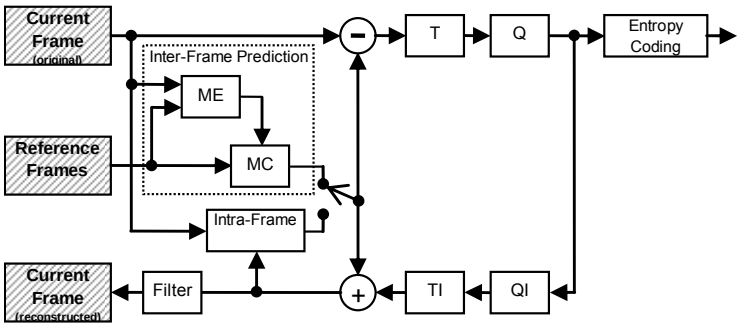


Fig. 1 - H.264/AVC encoder block diagram

This paper presents a proposal of an architecture for motion estimation based on the Diamond Search (DS) algorithm. Besides, it proposes some modifications to obtain better quality and speed during the encoding process by doing the motion compensation while the motion estimation is being performed. This architecture also generates the samples for the half and quarter pixels interpolation, which will increase the video quality.

The paper is organized as follows: section 2 presents how motion estimation works; the section 3 presents the Diamond Search algorithm and the proposal of this work; the section 4 shows the proposed architecture. The section 5 concludes this paper and show some future works.

2. Motion Estimation

The motion estimation uses the temporal redundancy to find the best way to represent a frame by using previously encoded data [3]. To do this, a frame is divided into several blocks. Each block is compared with blocks of the reference frame (already encoded) using a criterion of similarity. Then, when this block is found, it is created a motion vector for this position.

A search algorithm is used to go through the reference frame searching for the most similar block. The most known algorithms are: full search, that always finds the best block match, and some iterative algorithms such as diamond search and hexagon search. These algorithms find similar blocks, but not always the best [3].

A similarity criterion is used to compare the similarity between two blocks. One of the most used is the Sum of Absolute Differences (SAD) which is used in this architecture. Its equation is represented in eq. 1, where R represents the reference frame and O represents the original frame.

$$SAD(R,O) = \sum_{j=0}^{n-1}\sum_{i=0}^{n-1}|R(i,j) - O(i,j)| \qquad (1)$$

3. Diamond Search

The diamond search is an iterative search algorithm that achieves a good match between the block that will be encoded and the blocks from the reference frames. The algorithm can find a good match. However, the best match is not guaranteed because it does not calculate all the possibilities. Even so, it is possible to find a match with a little difference. This algorithm follows two patterns shown in Fig. 2 [5].

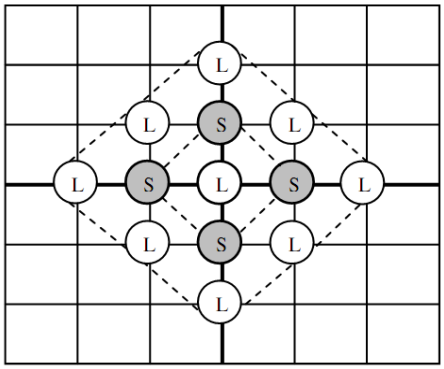


Fig. 2 - Diamond Search Patterns

Firstly, the pattern represented by the letter L, Large Diamond Search Pattern (LDSP), is processed applying a criterion of similarity between the blocks of these points and the block of the current frame. When the most similar block is not found in the center, the algorithm is reapplied using the position which obtained the better result as the new center. If the smallest error is found in the center, then a refinement is applied and the blocks are compared by the pattern represented by the letter S, Small Diamond Search Pattern (SDSP), which also compares the block in the center [5]. So, it is created a motion vector for this position.

This algorithm achieves lower quality results when compared to Full Search. However, the search can be done in a very short time, needing a lower number of operations.

This work proposes the use of two SDSP: one that is regularly used, and another to refine the previous iteration. Thus, the video can have an increase in quality, as this area is close to the vector that was chosen by the normal DS and nearby areas tend to be similar.

4. Proposed Architecture

This work proposes an architecture to perform the motion estimation with the DS search algorithm and two diamond search SDSP. The search area was considered 42x42 pixels, the similarity criterion chosen was SAD. The block size used was 8x8 pixels.

The architecture block diagram is represented in fig. 3, where the current frame memory contains the samples from the frame that should be encoded and the samples for interpolation, the reference frame memory

has the data that should be compared. The processing units (PUs) perform the SAD calculations for the blocks to be encoded and also the samples for interpolation.

The comparator finds the best block and generates a signal that tells which is the best block from the blocks that are being compared. The position updater starts in the center of the reference frame. As the architecture processes the data, the current vector points to the center of the new position that will be the center of the diamond search next iteration. Finally, the position updater SDSP generates the final motion vector.

The previous vector always holds the result of the previous iteration of the current vector and so it is possible to calculate the previous SDSP.

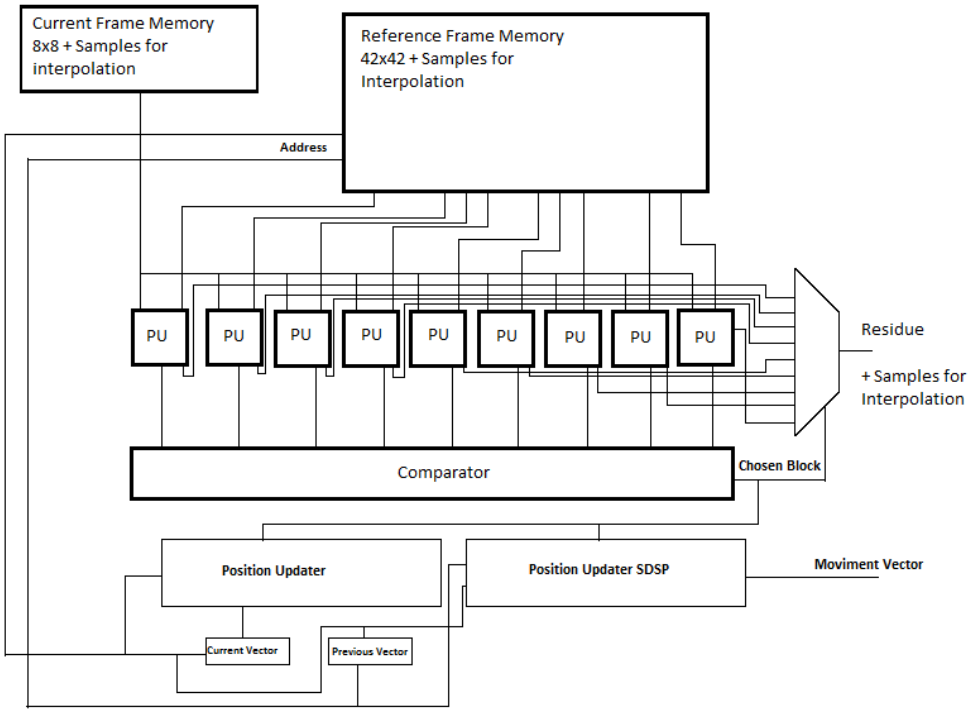


Fig. 3 - Diamond Search block diagram

Initially, the current frame memory and the reference frame memory receive the video data, then the data is compared in the processing units (PU), each PU is responsible for calculating one position of LDSP. The comparator finds the block that had the lowest error and sends a signal to the updater position, which informs where the new iteration will start from.

This position is used as the center where of the new DS. When the best block is found in the center, then the position updater SDSP generates the final motion vector.

The motion compensation is performed during the motion estimation. It is done by saving the values of the difference between the original block and the reference block. When the block is chosen the multiplexer chooses the PU that saved the residue and put it as an output.

This process also happens with the samples for the interpolation that are saved in the PU and then are sent to the output by the multiplexer according with the chosen block.

The architecture of the reference memory was described in order to optimize the use of the data already in it. As each vector is generated, the reference area needs to move. In each movement, a lot of data that were in memory, need only to be moved horizontally or vertically. The vectors from the current frame are generated in a spiral form as shown in Fig. 4 to get a better performance.

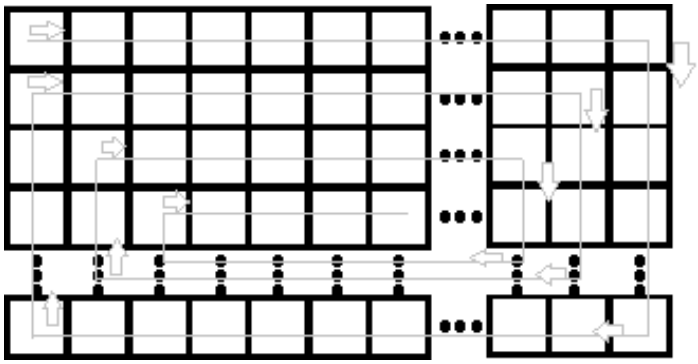


Fig. 4 - Flow encoding a frame

5. Conclusions and Future Works

This paper has presented a proposal of an architecture for a modified diamond search algorithm. This algorithm uses two SDSP: one as regularly used and the other one to refine the previous iteration. By doing this, it might be possible to get a better digital encoded video quality.

The proposed architecture also generates samples for the half and quarter pixels interpolation. It results in a big increase of encoded video quality. The architecture also performs the motion compensation while the motion estimation is being performed. It saves time in the whole encoding process, since after the motion estimation, it would be necessary to perform the motion compensation.

As future works are necessary to synthesize and validate the architecture. It is also important to evaluate the reference software and analyze the quality gain by using the idea of two SDSP. Besides, it is planned to integrate this architecture with a half pixel interpolator [4] and then integrate with another search algorithm for this interpolated area. After this, it is interesting to integrate with a quarter pixel interpolator and a quarter pixel search architecture, in order to have a complete quarter pixel motion estimation architecture.

6. References

[1] ITU-T Recommendation H.264/AVC (03/05): advanced video coding for generic audiovisual services, 2005.

[2] Brazilian Forum of Digital Television. ISDTV Standard. Draft. Dec. 2006 (in portuguese).

[3] I. Richardson, H.264 and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia. John Wiley&Sons, Chichester, 2003.

[4] M. Corrêa, "Desenvolvimento de uma Arquitetura para Interpolação de Half-Pixels segundo o padrão H.264/AVC". Proc. 16th Iberchip, 2010.

[5] L. Rosa, "SDS-DIC Architecture with Multiple Reference Frames for HDTV Motion Estimation". Proc. 24th SIM, 2009

NOCs and MPSoCs

Wire Length Evaluation of Dedicated Test Access Mechanisms in Networks-on-Chip based SoCs

[1]Alexandre Amory, [2]Cristiano Lazzari, [3]Marcelo Lubaszewski, [1]Fernando Moraes
alexandre.amory@pucrs.br

[1]PPGCC, Faculdade de Informática, PUCRS, [2] INESC-ID, Lisbon, Portugal
[3] PGMICRO, Departamento de Engenharia Elétrica, UFRGS

Abstract

The use of existing Networks-on-Chip (NoCs) for test data transportation has been proposed to avoid conventional dedicated Test Access Mechanism (TAM), reducing the chip global wiring length, however, it is not known how much wiring is saved by reusing NoCs as TAMs. This paper addresses this problem by presenting a wire length estimation method used to evaluate the cost of dedicated TAMs for NoC-based SoCs. The proposed method does not required layout information for the test optimization, thus, the test architecture can be defined earlier in the design flow. The experimental results demonstrate that dedicated TAMs require, on average, 24% of the global wires. On the other hand, results can vary depending on the SoC, from 5% to 58%, demonstrating the need of a fast wire length estimation to help the designer to decide the best test architecture for his design: dedicated TAM or NoC TAM.

1. Introduction

With the scaling of microchip technology, computation is becoming cheaper than communication. The main reason is that global wires do not scale as transistors do [1]. Global wires can be found in the chip-level communication infra-structures such as buses. Networks-on-Chip (NoCs) [2] may replace global buses in near future due to scalability and parallel communication features. NoCs alleviate the issues related to long global wires because NoCs consist of shared and segmented wires [3]; sharing wires reduces the number of global wires, while segmenting wires reduces their sizes.

Modular testing has been proposed as a solution to test such complex SoCs [4]. The conceptual model for modular testing consists of: test wrappers, used to switch between functional and test modes; and Test Access Mechanisms (TAMs), used to transport test data from/to the test pins to/from the Core-Under-Test (CUT). The most common practice for TAM design is to include dedicated and global test buses used only for test data transportation. Since these TAMs consist of long global wires, dedicated test buses are also subject to the same interconnect problems such as signal integrity, delay, and power dissipation. In an attempt to avoid the long global wires related to the TAMs, Cota et al. [5] proposed the use of the NoC to transport test data. Doing so, the NoC would avoid long global wires both in functional and in test modes.

The problem is that it is not know how much wiring could be actually saved by using NoCs as TAMs. The goal of this paper is to evaluate the amount of wiring required to implement dedicated TAMs in a NoC-based SoC with regular network topologies. For this purpose a new wire length estimation model is presented. As far as we know, this is the first paper to evaluate the amount of wiring required to implement dedicated TAMs. The prior works with closest motivation are related to test scheduling algorithm which optimizes both test length and TAM wire length [6, 7]. However, they do not actually evaluate the amount of TAM wires and they require layout information to perform the optimization, which means that first there should be a complete layout of the design to finally generate the test solution. The proposed wire length estimation model does not require layout information, thus, the test architecture and optimization can be concluded earlier.

This paper is organized as follows. Section 2 introduces the proposed wire length estimation method. Section 3 evaluates the wire length of dedicated TAMs for several SoCs. Section 4 presents the conclusion of the paper.

2. Wire Length Estimation Model for Dedicated TAMs

This section presents the proposed model used to estimate the wire length required to implement dedicated TAMs.

2.1. Introduction to the Proposed Model

The proposed model assumes that the SoC is represented by tiles [1] which are evenly distributed in the entire SoC area such that the distance between any two neighbor tiles is the same. Each tile can have zero or more cores and exactly one network interface. The rest of the system (clock and reset tree, test wires, and NoC)

[1]The use of tiles is common in physical synthesis of NoC-based systems [8].

are distributed among the tiles. This description is coherent to *homogeneous NoC-based systems*, where the tiles have similar logic, like in a homogeneous MPSoC system. This subject is further discussed in Section 2.3.

The proposed method counts the minimal number of hops required to reach all modules within a dedicated TAM. The wire length between cores within the same tile is supposed to be zero, while the wire length between cores in different tiles is equivalent to the number of hops between these two tiles. Fig. 1 represents a NoC-based SoC using three dedicated TAMs, and it is used to illustrate the proposed wire length estimation method. Each box represents a tile which consists of one router (identified by the number outside parentheses) connected to zero or more cores (identified by the number within parentheses).

The total number of wires to implement the set of TAMs of a SoC is defined as

$$w_{SoC} = \sum_{i=1}^{n} (h_i + 1) \times w_i \times 2 \quad (1)$$

where n is the number of TAMs, h_i is the minimum number of hops to implement TAM$_i$ plus one hop representing the wires from the input test pins to the first core of the TAM. Finally, w_i is the width of the TAM$_i$. Since there must be wires also for the test responses, then, the number of wires is multiplied by two.

Let us assume the following TAM assignment, illustrated in Fig. 1, created by a conventional test scheduling algorithm: TAM1={c1,c5,c6,c8,c9}, TAM2={c4,r01,r11,r12,r02}, TAM3={c0,c2,c3,c7,r00 ,r10,r20,r21,r22} assuming 16 test pins to connect the chip to the ATE. The width of these TAMs are 7, 5, 4 wires, respectively.

For instance, TAM1 has five cores where two of them are located in the tile 01 and the remaining cores are located in tiles 11, 10, and 20, thus, the minimum distance between these four tiles is three hops (see the continuous fat line in Fig. 1). Since the width of TAM1 is seven test wires, then it results in (3+1)×7×2=56 wires to implement the TAM1. The minimum number of hops for TAM2 and TAM3 are 4 and 5 hops, respectively. Finally, the total TAM wiring for this example is 154 wires ((3+1)×7×2=56 for TAM1, (4+1)×5×2=50 for TAM2, and (5+1)×4×2=48 for TAM3).

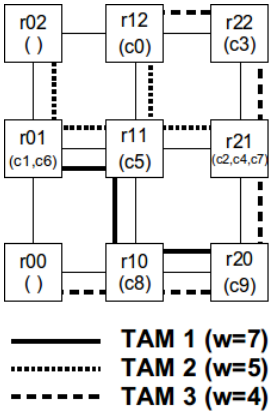


Fig. 1 - Example of a d695 SoC connected to a NoC used to estimate the wire length required to create the dedicated TAMs. *rXX* represent the router and *cX* represent the cores connected to the router. *W* represents the TAM width.

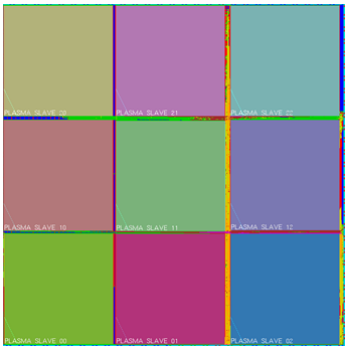


Fig. 2 - Layout of the d695 SoC with dedicated TAMs. The squares are the HeMPS tiles while the horizontal and vertical lines are the Hermes NoC with dedicated TAMs.

2.2. Wire Length Estimation Model

Finding the minimum number of hops h_i between the cores of a TAM i is equivalent to the Minimum Rectilinear Steiner Tree problem which is *NP*-complete [9]. The problem can be modeled as follows: given N points in the plane, find a minimum length tree of rectilinear edges which connects the points.

2.3. Limitations and the Scope of the Model

The actual TAM wire length in a chip also depends on the layout congestion. Congested layouts might require longer TAM wiring from one point to another than non-congested layouts. Since the proposed model does not capture layout congestion, the model represents the shortest wiring length required to design dedicated TAMs. The actual TAM wiring is expected to be longer than the estimated wire length. On the other hand, this model is much easier to estimate wire length since it does not require physical synthesis of large NoC-based SoCs.

The model is best suit for homogeneous SoCs where all tiles have similar sizes and the NoC has a regular topology, like mesh or torus. Although it is well known that there are other types of SoC architecture, an homogeneous SoC with regular NoC is still the most popular design style, where the proposed wire length estimation model could be applied. For instance, homogeneous SoCs have been used in both academic and commercial chip designs [3, 10-12]. In addition, it makes sense to use this model since most of the prior work

on testing NoC-based SoCs is based on regular NoC topology. It makes the proposed model suitable for measuring wire length of TAMs.

2.4. Evaluation of the Model

Let us take the d695 SoC presented in Fig. 1 as an example to compare the *actual* and the *estimated* wiring for dedicated TAMs.

As calculated in Section 2.1, the *estimated* number of test wires required to implement dedicated TAMs is 154. In an *i*-by-*j* mesh, there are $2\times(i\times(j-1)+j\times(i-1))$ channels. For example, the system d695 is a 3-by-3 mesh, thus, it has 24 channels of 32 bits or $24\times32=768$ wires[2]. Thus, according to the proposed model, close to 154/768 = 20% *of the global wires of the chip are required to implement the dedicated TAMs*.

Layout analysis is required to evaluate the *actual wiring* for dedicated TAMs. We implemented in VHDL the system presented in Fig. 1 using dedicated TAMs. The tiles are supposed to be hardcore while the NoC and the dedicated TAMs are softcore. Both the tile and the NoC are based on the HeMPS MPSoC [13] configured with buffers of size 16 and 32-bit channel width. The system has been synthesized to the library UMC 130nm.

The layout consists of two steps. The first step is to create the blackbox of the HeMPS tile with a 32-bit MIPS processor, network interface, DMA, and 16KB dual-port memory. It resulted in a box of $2560\times2550\mu m$ of area. The second step connects the tile to the HeMPS NoC (named Hermes), which is automatically generated by the Atlas environment [13], to the blackbox. Finally, the dedicated TAMs, depicted in Fig. 1, are included into the SoC.

After the SoC setup, Cadence™ tools were used for logic and physical synthesis. Fig. 2 illustrates the resulting layout of the SoC based on dedicated TAMs. The resulting wires are classified into *four classes of wires*: local, global, clock, and TAM wires. "Local wires" are required to implement the internal router logic. "Global wires" are used to connect the routers to each other and the router to the tile, excluding TAM and clock wires. "Clock wires" are used to implement clock and reset trees. Finally, "TAM wires" represent the dedicated TAMs. Tab. 1 shows the distribution of wire length among these types of wires.

Tab. 1: Distribution of wires in a NoC-based SoC with dedicated TAMs.

local wire length (%)	**global wire length (%)**	**clock wire length (%)**	**TAM wire length (%)**
67.32	19.42	8.11	5.15

The overhead of dedicated TAMs is small (5.15%) compared to the total wire length of the NoC. However, it consists of 5.15/19.42 = 26.5% of the global wires. Recall that the proposed model estimated that 20% of the global wires would be used to implement dedicated TAMs. The difference between the actual (26.5%) and estimated (20%) TAM wiring is due to routing congestion which is not captured in the proposed model.

3. Experimental Results

3.1. Experimental Setup

The first step is to build the NoC-based SoCs for the evaluation. The following systems from ITC'02 SoC Test Benchmarks [14] have been modified to include a NoC (the NoC size, i.e. the number of routers, for each system is in parentheses): d281 (3,3), d695 (3,3), g1023 (4,3), h953 (3,3), p22810 (5,5), p34392 (4,4), p93791 (6,5), u226 (3,3). The size of the NoC has been selected based on the number of cores of the system. The SoCs f2126 and q12710 have been excluded because they have only four cores and the SoC a586710 has only seven cores. It does not make sense to evaluate TAM wire length in such small SoCs. The t512505 SoC has been excluded because it has a *bottleneck core*, core 31, which requires about 88% of the entire SoC test data.

There is also the so called 'big(9,9)' SoC which has been created to test the scalability of the proposed model. This SoC is placed in a 9×9 mesh with 117 cores. These cores are the cores of the five biggest ITC'02 SoC Test Benchmarks, which are the SoCs p22810, p34392, p93791, t512505, and a586710.

The ITC'02 SoC Test Benchmarks were modified to include the NoC into the SoC. First, each core receives two OCP-like ports, one port to receive and the other to send data from/to the NoC. Each port has 39 control terminals and 32 data terminals plus the original number of terminals of a core defined in the ITC'02 SoC Test Benchmark. For example, the original module 1 of d281 SoC has 60 inputs and 26 outputs, thus, the modified module has $60+26+(2\times(39+32))$ terminals.

Second, the routers and the NoC are generated. For the sake of simplicity we assume that all routers in a system are identical, i.e. they have five bi-directional ports and the same number of test patterns. The number of terminals of a router is $5\times(2\times(2+32))$ (two control terminals and 32 data terminals multiplied by five bi-directional ports). Moreover, the router has 50 scannable flip-flops related to internal control logic. Recall that the test scheduler considers the routers as soft-cores.

Third, the cores of each SoC are placed on the NoC. Ten random placements have been generated for each SoC because the placement has an impact on the TAM wire length. The cores have been placed in the NoC randomly such that if the number of cores is greater than the number of routers, it makes sure that all routers

[2]This wire count does not consider control wires used to implement the protocol.

have at least one core and no router receives more than two cores. It also makes sure that all placements are different from each other. At this point the SoCs have been generated. Next, the TR-Architect [4] is used for SoC test scheduling.

3.2. Wire Length Savings

This section uses the model presented in Section 2 to evaluate the amount of wiring spent in systems based on dedicated TAMs. In other words, *it evaluates the amount of wiring that could be saved by using NoC TAM*.

First, we calculate the number of wires to implement a NoC with 32-bit width channels for each SoC. For instance, system d695 has nine routers, thus, it has 24 channels of 32 bits, then, the NoC requires 24×32=768 wires. The column "NoC Wires" of Fig. 3 presents the amount of wires to implement the NoC channels. The parameter w_{max} represents the number of test pins available for the test architecture.

SoC	w_{max}	NoC wires	avg TAM wires	diff (%)	best TAM wires	diff (%)	worst TAM wires	diff (%)
d281	16	768	255	33.23	232	30.21	272	35.42
	24	768	205	26.64	164	21.35	234	30.47
	32	768	147	19.19	130	16.93	156	20.31
	40	768	208	27.06	196	25.52	230	29.95
	48	768	216	28.07	184	23.96	244	31.77
	56	768	242	31.56	202	26.30	264	34.38
	64	768	277	36.02	226	29.43	302	39.32
d695	16	768	202	26.35	168	21.88	212	27.60
	24	768	257	33.46	228	29.69	288	37.50
	32	768	397	51.74	370	48.18	452	58.85
	40	768	222	28.91	170	22.14	284	36.98
	48	768	398	51.88	344	44.79	440	57.29
	56	768	330	42.97	268	34.90	410	53.39
	64	768	445	57.94	356	46.35	534	69.53
g1023	16	1088	243	22.33	216	19.85	262	24.08
	24	1088	277	25.50	266	24.45	302	27.76
	32	1088	283	26.03	240	22.06	324	29.78
	40	1088	337	31.01	274	25.18	374	34.38
	48	1088	428	39.38	390	35.85	466	42.83
	56	1088	361	33.16	304	27.94	422	38.79
	64	1088	156	14.36	148	13.60	164	15.07
h953	16	768	56	7.34	50	6.51	62	8.07
	24	768	56	7.34	50	6.51	62	8.07
	32	768	56	7.34	50	6.51	62	8.07
	40	768	56	7.34	50	6.51	62	8.07
	48	768	56	7.34	50	6.51	62	8.07
	56	768	56	7.34	50	6.51	62	8.07
	64	768	56	7.34	50	6.51	62	8.07
p22810	16	2560	415	16.23	374	14.61	458	17.89
	24	2560	419	16.35	382	14.92	438	17.11
	32	2560	518	20.22	464	18.13	530	20.70
	40	2560	582	22.74	536	20.94	628	24.53
	48	2560	419	16.38	366	14.30	490	19.14
	56	2560	628	24.52	570	22.27	672	26.25
	64	2560	910	35.55	848	33.13	968	37.81
p34392	16	1536	274	17.85	250	16.28	304	19.79
	24	1536	227	14.79	214	13.93	236	15.36
	32	1536	192	12.49	160	10.42	216	14.06
	40	1536	283	18.44	238	15.49	312	20.31
	48	1536	431	28.05	372	24.22	498	32.42
	56	1536	431	28.05	372	24.22	498	32.42
	64	1536	431	28.05	372	24.22	498	32.42
p93791	16	2880	499	17.34	454	15.76	532	18.47
	24	2880	451	15.66	402	13.96	522	18.13
	32	2880	903	31.37	834	28.96	1010	35.07
	40	2880	1005	34.90	904	31.39	1088	37.78
	48	2880	897	31.14	812	28.19	994	34.51
	56	2880	1154	40.06	1080	37.50	1290	44.79
	64	2880	1324	45.97	1256	43.61	1380	47.92
u226	16	768	96	12.50	96	12.50	96	12.50
	24	768	144	18.75	144	18.75	144	18.75
	32	768	192	25.00	192	25.00	192	25.00
	40	768	240	31.25	240	31.25	240	31.25
	48	768	288	37.50	288	37.50	288	37.50
	56	768	350	45.57	350	45.57	350	45.57
	64	768	414	53.91	414	53.91	414	53.91
big	16	9216	1159	12.58	1104	11.98	1224	13.28
	24	9216	1668	18.10	1578	17.12	1758	19.08
	32	9216	479	5.20	278	3.02	690	7.49
	40	9216	776	8.42	664	7.20	898	9.74
	48	9216	382	4.14	338	3.67	436	4.73
	56	9216	804	8.72	418	4.54	1182	12.83
	64	9216	1139	12.36	902	9.79	1566	16.99
average	—	—	—	24.56	—	21.97	—	27.10

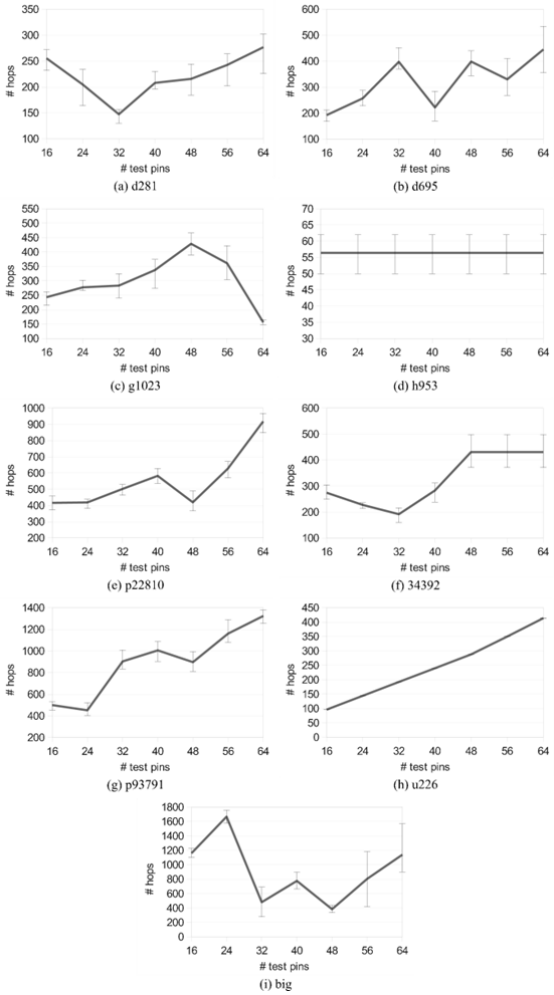


Fig. 3 - Wire length for dedicated TAMs. The line represents the average wire length for the ten placements

while the error bars represent the wire length for the worst and best placements. The number of hops (the distance between two adjacent tiles) is used as a relative wire length unit. The actual wire length in μ can be easily calculated by multiplying the tile length by the number of hopes determined by the proposed model.

Second, the TAM wire length of a system based on dedicated TAM depends on how the cores are placed into the NoC. For this reason we evaluate ten placements for each system. The column "TAM Wires" of Fig. 3 presents the average/best/worst TAM wire length for each system considering different number of test pins. The column "diff" of Fig. 3 represents the relative number of wires to implement dedicated TAMs compared to the number of wires of the 32-bit NoC. For instance, the 202 wires (see Fig. 3, SoC d695, w_{max}=16) correspond to 26% (202/768), of wires of a 3x3 NoC considering channels of 32-bit width. The illustrations of Fig. 3

represent the same results of the table, but, it is more intuitive and less detailed. The rest of this section discusses these results individually.

The number of test pins does not change the wire length of h953 SoC because TR-Architect generated the same scheduling for different number of test pins [3]. On the other hand, different placements cause a small wire length variation in the same SoC (see the error bar for the best and worst placements). The opposite happens for the u226 SoC. It can be observed that more test wires cause more TAM wire length, however, different placements have no effect on wire length. It happens because, for the same number of test pins, the test scheduler generated the same scheduling despite of the different placements.

The g1023 SoC has reduced TAM wiring when the number of test pins is 54 and 64. It happens because, in this SoC, the test scheduler assigns almost one core per TAM. These cases require a smaller amount of wiring since there are wires only between the chip test pins and the CUT test ports. Most other SoCs have TAMs with more than one core.

Other cases, which have not been mentioned, fall in one of the following situations: the test scheduler assigns most of the test wires to a TAM with only one core or it assigns few test wires to a TAM with most cores of the SoC. In these cases the TAM has wires only from/to the test pin to the/from the CUT, requiring wide and short wires. The remaining cores of these systems are tested by narrow but long TAMs. The combination of TAMs with wide and short wires and TAMs with narrow but long wires leads to shorter TAM wire lengths, globally. For this reason these systems require fewer wires.

The average results for the average/best/worst placements are, respectively, 24.6%, 22%, and 27% (bottom of Fig. 3). It means that, on average, *about 24.6% of the SoC global wires are used to implement dedicated TAMs*. For some systems, like d695 with 64 test pins, the TAM wiring can be about 58% of the SoC global wires. Note that the model is optimistic, as explained in Section 2.3. It means that the actual wiring for dedicated TAMs is larger.

Finally, Fig. 4 has been generated by grouping the results in Fig. 3 (columns 5, 7, and 9) in terms of number of test pins and taking the average results. It presents the average usage of TAM wires (for the average, best, and worst placements) per number of test pin considering all SoCs. It shows that the TAM wire length increases as the number of test pins increases, demonstrating that dedicated TAM might not be viable for large number of test pins. It also shows that the error bars are increasing as the number of test pins increases. It means that the impact of placement on the wire length tends to increase as the number of test pins increases.

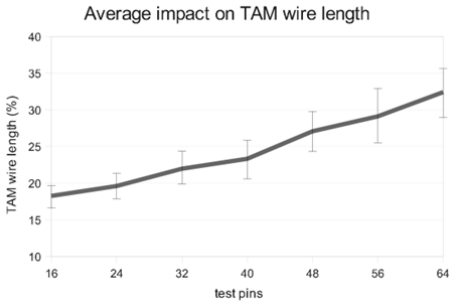


Fig. 4 - Average amount of test wiring compared to the global wires. The line represents the average percentage of wires used to implement dedicated TAMs. The error bars represent the average percentage for the worst and best placements.

4. Conclusion

The previous papers about testing NoC-based SoCs claimed that NoC TAM saves global wiring but it was not known how much wiring was actually saved. This paper addresses this problem by proposing a wire length estimation model used to evaluate the amount of wiring required to implement dedicated TAMs. To the best of our knowledge, this is the first paper to estimate the amount of wiring saved by using NoC as TAM.

The proposed model has been used to evaluate several ITC'02 SoC Test Benchmarks SoCs, including a large SoC with 117 cores. It has been concluded that the TAMs increase the total number of global wires of the chip in 24%, on average, but the variation can be large, depending on the SoC; in some cases it can be 58% while for others cases it can be less than 10%. With the proposed model the designer can quickly decide, without requiring information from the physical synthesis, whether NoC TAM is a good approach for his design.

5. Referências

[1] ITRS, International Technology Roadmap for Semiconductors, Semiconductor Industry Association, 2005.

[2] T. Bjerregaard and S. Mahadevan, "A survey of research and practices on network-on-chip," ACM Computing Surveys, vol. 38, no. 1, 2006.

[3] This is not a limitation of TR-Architect, but a feature of the SoC. Goel and Marinissen [4] have proved that this SoC reaches the theoretical lower bound with less than 16 test pins.

[3] F. Angiolini, P. Meloni, S. Carta, L. Raffo, and L. Benini, "A layout-aware analysis of networks-on-chip and traditional interconnects for MPSoCs," IEEE Transactions on CAD of Integrated Circuits and Systems, vol. 26, no. 3, pp. 421–434, 2007.
[4] S. K. Goel and E. J. Marinissen, "SOC test architecture design for efficient utilization of test bandwidth," ACM TODAES, vol. 8, no. 4, pp. 399–429, Oct. 2003.
[5] E. Cota, C. A. Zeferino, M. Kreutz, L. Carro, M. S. Lubaszewski, and A. A. Susin, "The impact of NoC reuse on the testing of core-based systems," in Proc. VTS, 2003, pp. 128–133.
[6] S. K. Goel and E. J. Marinissen, "Layout-driven SOC test architecture design for test time and wire length minimization," in Proc. DATE, 2003, pp. 738–743.
[7] E. Larsson and H. Fujiwara, "Test resource partitioning and optimization for SOC designs," in VLSI Test Symposium, 2003, p. 319.
[8] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," IEEE Circuits and Systems Magazine, vol. 4, no. 2, pp. 18– 31, 2004.
[9] B. T. Preas and M. J. Lorenzetti, Physical Design Automation of VLSI Systems. Benjamin Cummings Publishing Company, 1988.
[10] A. Banerjee, R. Mullins, and S. Moore, "A power and energy exploration of network-on-chip architectures," in Proc. NoCs, 2007, pp. 423–425.
[11] B. Li, L. Peh, and P. Patra, "Impact of process and temperature variations on network-on-chip design exploration," in Proc. NoCs, 2008, pp. 423–425.
[12] S. R. Vangal et al., "An 80-tile sub-100-w teraFLOPS processor in 65-nm CMOS," IEEE Journal of Solid-State Circuits, vol. 43, no. 1, pp. 29–41, 2008.
[13] E. A. Carara, R. P. Oliveira, N. L. V. Calazans, and F. G. Moraes, "HeMPS - a framework for NoC-based MPSoC generation," in Proc. ISCAS, 2009, pp. 1345–1348.
[14] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in Proc. ITC, 2002, pp. 519–528.

Model-based Power Estimation of NOC-based MPSOCS

Luciano Ost[1], Guillerme Guindani[1], Leandro Soares Indrusiak[2], Fernando Moraes[1]

[1]*Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)*
Av. Ipiranga, 6681 - prédio 32 - Porto Alegre - Brazil - CEP 90619-900
{luciano.ost, guilherme.guindani, fernando.moraes}@pucrs.br

[2]*University of York*
YO10 5DD, York, United Kingdom
leandro.indrusiak@cs.york.ac.uk

Abstract

Networks-on-Chip (NoCs) can use more than one third of the power budget of the chip they are embedded in. Thus, design decisions that impact NoC power dissipation can be crucial to the success (or failure) of a product, more so for battery-powered embedded systems. This paper covers the integration of a power estimation model into an abstract model of a NoC-based MPSoC. Results present the design space exploration of a system with 4 real applications running simultaneously. The integration of the power estimation model into the proposed design flow enabled to analyze different design parameters to reach the most power-efficient architecture.

1. Introduction

NoC-based MPSoCs are a trend for future portable systems (e.g. mobile internet devices - MIDs) that require high performance while maintaining low power dissipation. Fourth generation (4G) systems are examples of MIDs with limited power budget (battery operated), which must be efficiently used for executing several high performance applications. Examples of expected 4G applications for future portable systems are: (*i*) three dimensional (3D) and holographic gaming, (*ii*) 16 megapixel smart cameras and (*iii*) high-definition (HD) camcorders [1]. In this scenario, the impact of the power dissipation by the NoC interconnect becomes a critical challenge in the design space exploration of such systems [2]. For example, NoC infrastructure of two real systems reported by [3] and [4] are responsible for 36% and 28% of the total power dissipation, respectively.

The detailed estimation of NoC power dissipation at transistor or gate levels is very time-consuming due to their size and complexity. Thus, simple and accurate high level models became necessary to achieve acceptable results within the time-to-market frame of complex systems. Modeling at higher abstraction levels is a common practice to increase and simplify development and validation of complex systems as MPSoCs. The simulation speed, the improved observability, and debugging capabilities provided by higher-level models reduce design decisions that impact NoC power dissipation, which can be crucial to the product life-time [3].

This paper presents the integration of a power estimation model into an actor-oriented model of a NoC-based MPSoC, aiming to enable fast design space exploration and to provide an accurate estimation of the power dissipated by the NoC on each design alternative. The proposed approach allows the co-validation of complex applications (modeled as UML sequence diagrams) mapped onto the platform model, considering power constraints early at the design process [5].

2. Rate-based Power Model

The *rate-based power model* considers data volume, but computed as a *transmission rate* within a given sample period; and accuracy is guaranteed from a physical calibration step, which defines the power dissipation for each transmission rate [6]. Such model is highly customizable, and can be easily applied to different NoC architectures and technologies.

The rate-based power estimation model comprises of two steps: *calibration* and *application*. The *calibration* step (step 1 in Fig. 1) defines the relevant model parameters. This step starts with the synthesis of the NoC router, generating an HDL description of this router mapped to the target technology. This new HDL code replace one router of the original NoC description for the next simulations (2). This NoC description is then simulated (3) with different traffic scenarios, each of them with a fixed injection rate (4). The switching activity of each simulation run is analyzed by Synopsys PrimePower estimation tool (5), which computes the average power dissipation of each router element: (*i*) buffers (responsible for at least 80% of the average power dissipation); (*ii*) internal crossbar and (*iii*) control logic.

After the calibration phase, a power dissipation table is generated for each injection rate and for each router

element (6). Using linear approximation, an equation that gives the power dissipation as a function of the injection rate is obtained for each table.

In the *application* step (7), the NoC is simulated to obtain the reception rate at each buffer (8). This is measured with monitors inserted at each router buffer. The monitors count received flits in a parameterizable sample window. For each reception rate, the associated power dissipation (***Pbuffer***) is annotated, applying the equations obtained in the calibration step (9). The power dissipation of the control logic (***Pcontrol***) and the crossbar (***Pcrossbar***) are obtained using the average buffers reception rate.

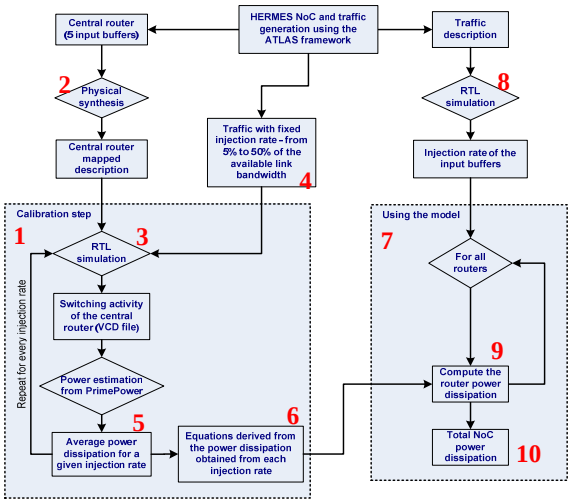


Fig. 1 - Rate-based power estimation model flow.

The power dissipation of a router is given by Equation (1), where *m* represents the number of sampling periods and *n* is the number of buffers in this router. The NoC average power dissipation is given by the summation of the dissipation values of every router.

$$P_{avg} = \sum_{k=1}^{m} \frac{\sum_{i=1}^{n} Pbuffer_{k_i}}{n} + Pcrossbar + Pcontrol \quad (1)$$

3. Actor-oriented Power Model

The proposed power estimation method can provide basic figures, but it needs a proper platform model to adjust those figures according to the power consumption patterns of each application. This work integrates that power model to an actor-oriented platform model, which is abstract enough to allow fast simulation but provides sufficient information to the power estimation algorithm. It also enables application modeling using multiple models of computation (e.g. finite-state machine, process networks, discrete events), allowing applications to be described using time and concurrency primitives which better reflect their nature.

An actor-oriented NoC model was developed for this work using was reference model the HERMES infrastructure [7]. It has a 2D mesh topology and wormhole switching, and all its buffers, arbiters and routers are modeled as actors, which communicate by exchanging tokens. Ptolemy II was used as the modeling framework here, but other environments supporting actor semantics could be used as well. The flit-by-flit transmission of packets over the NoC is modeled by token exchanges by the actors. A 1-to-1 mapping of flits to tokens can provide high accuracy, but it leads to long simulation times. Instead, in this work the packet payload is abstracted using PAT [8], which reduces simulation time while still obtaining accurate results for latency and throughput.

To support the rate-based power estimation model, the proposed actor model also includes the following features:

1. Each buffer computes its average reception rate – *avrr*, according to Equation 2, where: *recPkts*, number of received packets in the sample window; *flit*, is the flit size; *T*, the clock period; and *sw* the sample window, in clock cycles.

$$avrr = \frac{recPkts \times pktSize \times flit}{T \times sw} \quad (2)$$

2. The power dissipation of the links (LinkPD) is calculated according to the following Equations:

$$link_{BPD} = C_{load} \times fNoC \times Vcc^2 \quad (3)$$

$$Link_{PD} = \left(link_{BPD} \times \left(w \times \alpha\right)\right) \times avrr \quad (4)$$

where:

C_{load} *represents the total switching capacitance of the wires*
fNoC *is the NoC frequency*
w *is the number of the wires used for data transmit ion*
α *is the link switch activity*

3. A monitor collects the average reception rate of each buffer, avrr (Equation 2), and the switching activity of its associated link, which are sent to PowerScope at the end of each sample period. The power dissipation is obtained applying avrr to the individual power equations (6 in Fig. 1, Pbuffer, Pcrossbar and Pcontrol). The power dissipation of the router is then computed from Equation 1.

PowerScope generates graphics and a report including energy consumption, maximum, minimum and average power per router. PowerScope uses the following power parameters: (*i*) switch control base dissipation; (*ii*) switch control variable dissipation; (*iii*) buffer base dissipation; (*iv*) buffer variable dissipation; (*v*) link switch activity.

4. Results

In terms of *accuracy,* results indicate that the difference error in the average power dissipation between both actor-oriented and RTL models is negligible [9]. These results prove that the proposed simplified NoC power estimation model can accelerate the power estimation (when compared to RTL models) due to the design flexibility, setup and debugging features, without any loss of accuracy.

In terms of evaluation time, the *rate-based* power estimation is slower than *volume-based* estimation models (e.g. Hu [10] that applies simple equations) and faster than *commercial tools* (e.g. Synopsys PrimePower), that has to generate the switching activity for the entire NoC, which can be unfeasible [9]. Considering Synopsys PrimePower as reference model, the rate-based model (VHDL implementation) has an average error (relative difference) of 5% while the volume-based model (algorithmic model) presented an error up to of 50% for power estimation analysis [9].

Four applications were modeled (using actors and UML diagrams) in Ptolemy II: (*i*) *HDTV*, comprising end-to-end transmission of 10 HDTV channels, modeled as 2 application blocks, with frames obtained from real application traces; (*ii*) VOPD (*Video Object Plan Decoder*), modeled as 12 application blocks; (*iii*) *MPEG4 decoder*, modeled as 12 application blocks [3]; and (*iv*) *an automotive application,* modeled as 10 application blocks [5].

The NoC infrastructure has the following parameters: 6x6 mesh topology, XY routing algorithm, 32-bit flit size, packets with 128 flits and handshake control flow. The rate-based power model was calibrated using the XFAB XCMOS 0.18 µm (XC018) 1.8V technology, adopting clock-gating, and a 250 MHz clock frequency.

The MPSoC power estimation was obtained for different application characteristics and two different mapping heuristics. The simulation scenario has all four applications executing simultaneously in the same platform for one second and the design space exploration varies:

- switching activity in the NoC links: 10%, 20%, 30%, 40% and 50%;
- mapping heuristic : random (reference worst-case mapping) and GI (greedy incremental).

Fig. 2 shows the NoC average power dissipation for two different mapping heuristics, when varying the link switching activity. As expected, the impact of the mapping heuristic on the average power dissipation can be clearly seen. The power dissipation increases with the increase of the switching activity: for a switching activity of 50%, the difference between GI and random mapping reaches 48.16%. Such results show the importance of profiling an application's average switching activity, and using it to guide design choices. The heuristic GI presents less NoC average power dissipation because applications are mapped in one region, minimizing network congestion.

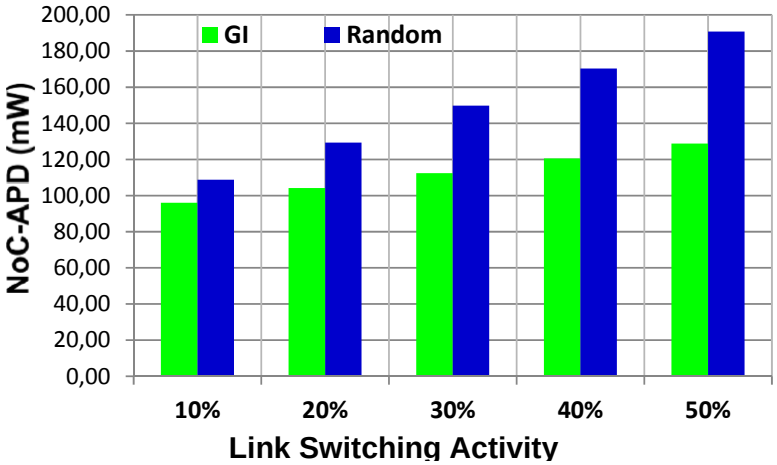


Fig. 2 – NoC average power dissipation for two mappings and link switching activity.

5. Conclusions and Future Work

The most promising technique to explore the complex design space of NoC-based MPSoC platforms is to build simpler, more abstract models of applications and platform components, and to evaluate the impact of alternative compositions on performance and power dissipation. The accuracy and speed of such evaluation must be high, and the effort to build and compose such models must be very low, so that they can provide meaningful results early on the design flow.

This paper addressed an import issue in this scenario: the accuracy of power estimation of high-level NoC-based MPSoC models. By integrating the rate-based power model into an abstract model that can be use to obtain accurate NoC power results of multi-applications mapped onto NoC-based MPSoCs platforms.

Additional work will also be done on extending the power estimation model so that it considers also the power dissipation due to the switching activity in the router buffers.

6. References

[1] Krenik, B. 4G wireless technology: When will it happen? What does it offer? In: IEEE Asian Solid-State Circuits Conference (A-SSCC'08), 2008.

[2] Atitallah, R. B.; et. al. MPSoC power estimation framework at transaction level modeling. In: Int. Conference on Microelectronics (ICM'07), 2007.

[3] Lee S. E. at. al. A high level power model for Network-on-Chip (NoC) router. Computers & Electrical Engineering, 35(6), 2009.

[4] Kahng, A.; et. al. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In: Design, Automation and Test in Europe (DATE'09), 2009.

[5] Määttä, S. et al. Validation of Executable Application Models Mapped onto Network-on-Chip Platforms. In: IEEE Symposium on Industrial Embedded Systems (SIES'08), 2008.

[6] Guindani, G. et al. NoC Power Estimation at the RTL Abstraction Level. In: Computer Society Annual Symposium on VLSI Design (ISVLSI'08), 2008.

[7] Ost, L. et al. A Simplified Executable Model to Evaluate Latency and Throughput of Networks-on-Chip. In: Symposium on Integrated Circuits and Systems Design (SBCCI'08), 2008.

[8] Moraes, F. et al. HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip. Integration the VLSI Journal, 38(1), 2004.

[9] Ost, L.; et. al. A high abstraction, high accuracy power estimation model for networks-on-chip. In: Symposium on Integrated Circuits and Systems Design (SBCCI'09), 2009.

[10] Hu, J.; Marculescu, R. Energy-aware mapping for tile-based NoC architectures under performance constraints. In: Asia South Pacific Design Automation Conference (ASP-DAC'03), 2003.

Implementation and Evaluation of a Congestion Aware Routing Algorithm for Networks-on-Chip

Leonel Tedesco, Thiago Gouvea da Rosa e Fernando G. Moraes
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - prédio 32 - Porto Alegre - Brazil - CEP 90619-900
{leonel.tedesco, fernando.moraes}@pucrs.br

Abstract

The goal of this work is to propose and evaluate an adaptive source routing algorithm, where the path between source and target PEs may be modified due to congestion events. The proposed method requires QoS session establishment and traffic monitoring. A QoS session establishes a connection between two IPs, applying application constraints. Traffic monitoring carries congestion information to the target, leading to a global view of the routing path. Evaluated performance figures include latency, traffic distribution and the delay to switch to a new path. For hot-spot traffic scenarios, the average latency is reduced by 10%. The proposed routing method also achieved a better network occupation. The delay to switch to a new path defines how fast the algorithm reacts to a congestion event. The proposed algorithm modifies the path in average after few messages, conduction the flow to a new path with minimal latency.

1. Introduction

The heterogeneity of applications on the emerging embedded systems is an important feature to consider on the MPSoC design. Processing elements, which perform distinct functions and produces different traffic patterns, bring dynamicity and unpredictability to the overall chip communication events. In addition, different QoS requirements and levels are found 0, which are usually specified in terms of throughput, latency and deadlines 0. Due to this dynamic behavior, the treatment of unpredictable events and QoS constraints should be carried out at run time. Monitoring units, which collect statistical traffic information, are largely employed in NoCs. Such units can notify some modules of the system that abnormality in the network traffic has occurred. If the monitored parameter is out of its bounds, some action is taken. Examples of actions driven by monitoring include: data injection control 0, dynamic priorities for QoS packets 0, dynamic buffer allocation 0, adaptive routing algorithms 000.

The major part of the state of the art proposals for adaptive routing algorithms considers local traffic monitoring, i.e., each router analyses their immediate neighbors to choose the output port. This approach presents a limited view of the network traffic, being suitable for traffic workloads with a high degree of locality, where nodes communicate with those that are closer to them. In traffic patterns with lower locality, the local monitoring may induce the routing of a given traffic to other congested areas. *The main contribution of this work is a method that uses information collected on the source-target path to execute adaptive source routing*. To avoid hot spots produced by dynamic traffic events, the proposed method is summarized as follows: (*i*) traffic monitoring, done in a distributed way, with information collected on the routing path; (*ii*) transmission of congestion information to the traffic source; (*iii*) modification of the path to the target at the source, if a congestion path is detected.

2. Monitoring and Congestion Detection

Two packet classes are employed: DATA and ALARM packets. ALARM packets are used to notify the source router the congestion points in the source-target path. The ALARM packet is a high priority packet, containing only 1 flit. Three tables store congestion information: (*i*) SCT (Source Congestion Table), inserted at the source NI; (*ii*) RCT (Router Congestion Tables), available in all routers of the NoC; (*iii*) TCT (Target Congestion Table), inserted at the target NI. The SCT contains, for each hop, 3 fields: *hop* number; *path*, which identifies the output port taken at each hop; *cong*, which indicates if the hop is congested or not. The RCT has a set of entries, each one for a given QoS flow passing through the router. Each entry stores the flow identification (*flow number*), the hop number (*identification*), and the metric used to identify congestion. The TCT stores the congestion level at each router of the source-target path.

The first packet of the flow establishes a session between the source-target pair. Each session contains one or more messages, and each message are composed by fixed size packets. Although a session is established, there is no resource reservation, as in circuit switching. This first packet initializes the field identification of all RCTs in the path, and the TCT.

After session initialization, each DATA packet is transmitted with a different *ident* value (varying from 1 to the number of hops in the path). When a router in the path receives a DATA packet, it verifies the *ident* field, and if its matches with the router address, *ocup* information is inserted into to packet. This field is filled with

the value of the parameter adopted for QoS evaluation. This parameter can be, for example, the amount of used buffer slots, the output data rate or the average time spent by packets to traverse the router.

At the end of each message, the NI compares the TCT congestion levels to a given threshold, previously defined according to the QoS requirements of the application that generates the flow, back propagating an ALARM packet. If there is no congestion, all congestion bits of the ALARM packet are zero. Otherwise, the ALARM packet contains the address (hop number) of the congested routers. When the source router receives the ALARM packet, a new path is computed (if it is necessary) and used for the next message. No packet reordering mechanism is necessary since (*i*) all packets of a given message use the same path; (*ii*) packets belonging to different messages are not mixed, because an ALARM packet must be received before the transmission of a new.

3. Path Computation Algorithm

The definition of the new path adopts the following assumptions: (*i*) minimal routing, all paths have the same number of hops to the target; (*ii*) the new path should use, if it is possible, routers near to the oldest path to minimize the impact of the new flow in other existing flows; (*iii*) the *x* direction has higher priority when computing a new path; (*iv*) the search in the *x* direction continues until a *y* column in the present *x* position is congestion free. Two data structures are used in the proposed method, being localized in the NI of the source router. A *matrix* stores the congestion history of the routers that were part of the paths computed by the routing algorithm. Considering that *adaptive minimal routing* is used, these routers are those inside the rectangle defined the by addresses of the source-target pairs. A *vector* is used to store the current congestion level of the path.

The method for adaptation is exemplified by the Fig. 1. The objective is to obtain a partial path with lines and columns of the mesh without congested routers. The adaptive algorithm starts seeking non-congested routers in the *x* direction (as in the XY routing algorithm). The *y* direction is taken when a congested neighbor in the *x* direction is congested. If so, the next hop must be the next located in the *y* direction, as illustrated in (a). If not, routers in the *x* direction are analyzed. The next step is to define the search space for the present y address. In (b) all routers in y=2 are not congested. The columns for each router on the current y coordinate are analysed. For each router in an *x* position, it is verified all routers which belong to the correspondent *x* coordinate, and that can make part of the new routing path. If there is at least one congested router, the column related to its *x* coordinate is considered not congestion free. The goal is to find the closest congestion free column to the target router. Fig. 1 (b) illustrates that there are two columns that are congestion free, in the *x* coordinates 1 and 3.

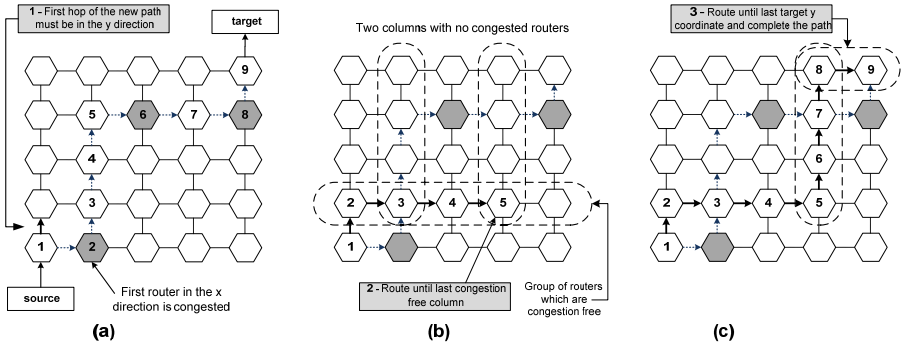


Fig. 1 – New path definition algorithm.

Once an *x* coordinate congestion free is found, the path is incremented with the horizontal and vertical routing directions, until the y coordinate of the target is reached. In the example shown in Fig. 1(b), the path is completed with 3 routes on the *x* direction. In Fig. 1(c) is shown 3 routes in the *y* direction. This is done between the lines 24 and 33. Once the partial path is completed, it is necessary to complete the path until the target node, if the target is not already reached, which may occur. Due to the fact that an alignment in *x* or *y* direction with the target router will be achieved after the execution of *new path* function, only routing in the vertical or horizontal direction will be done. Fig. 1(c) illustrates a path completion until the target, with one routing hop in the *x* direction (in the example, the *east* direction). Before the adoption of the new path, a special DATA packet is sent to the target using the previous path to clean the congestion tables. The new path is than initialized with a DATA packet to open a new routing session, identifying the routers of this path. Considering that minimal routing is adopted, the TCT on the target remains with the same size.

4. Results

The implementation of the presented methods is in SystemC TLM. The main features of the used NoC are are: (*i*) source routing; (*ii*) wormhole packet switching; (*iii*) flit size equals to 32 bits; (iv) end-to-end credit

based flow control. Fig. 2 illustrates the spatial traffic distributions. A CPU, responsible to control the traffic generators, is placed in the NoC central position. The QoS flows, generated by S nodes, execute the method for adaptive routing, generating each one 500 16-flit packets. Traffic targets, labeled as T, analyze the congestion condition of the QoS paths and generate ALARM packets. In both traffic distributions QoS flows are S1→T1 and S2→T2, and the shadowed routers generate disturbing flows. Each router monitors the time each flit waits in the input buffers before being injected into the network. Such spent time is defined as *flit time*. The latency of a flit is the addition of all *flit times* in the path. For the scenarios with disturbing traffic, a router with flits waiting more than 2 clock cycles to be injected into the network is considered congested. Each disturbing source generates 250 16-flit packets.

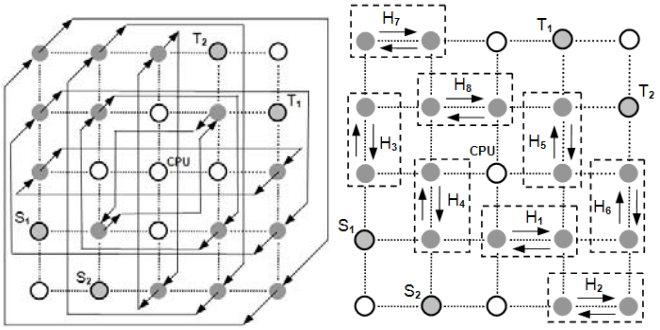


Fig. 2 – Spatial traffic distributions, complement and hot-spot (higher locality).

4.1. Latency Evaluation

Latency evaluation only considers packets of QoS flows S1→T1 and S2→T2. Table 1 presents the obtained average latency and standard deviation values, in clock cycles. The proposed routing method does not decrease the average latency values for the complement traffic distribution. Two reasons explain such behavior: traffic evenly distributed inside the network, leading to an absence of paths to be explored by the adaptive routing; overhead for path changing.

Table 1 – Packets Latency Results for QoS flows, in clock cycles (AV: Average, SD: Standard Deviation).

Flow	Message size for QoS flows (packets)	*Complement Traffic Distribution*				*Hot-spot Traffic Distribution*			
		Without the proposed method		**With the proposed method**		Without the proposed method		**With the proposed method**	
		AV	SD	**AV**	**SD**	AV	SD	**AV**	**SD**
S1 → T1	8 (S1)	65.9	19.3	**63.6 (-3.5%)**	**20.5**	59.2	9.5	**53.6 (-9.5%)**	**9.7**
	16 (S2)	66.2	19.3	**63.8 (-3.6%)**	**19.9**	59.2	9.5	**52.8 (-10.8%)**	**10.0**
	32 (S3)	65,7	19.2	**64.4 (-1.9%)**	**21.9**	59.3	9.5	**54.2 (-8.6%)**	**10.3**
S2 → T2	8 (S1)	47.1	1.4	**48.6 (+3.2%)**	**5.2**	58.4	11.5	**51.1 (-12.5%)**	**8.9**
	16 (S2)	47.2	1.9	**48.7 (+3.2%)**	**5.1**	57.9	11.2	**51.2 (-11.6%)**	**9.1**
	32 (S3)	47.3	2.6	**48.3 (+2.1%)**	**4.5**	57.9	11.2	**51.8 (-10.5%)**	**9.5**

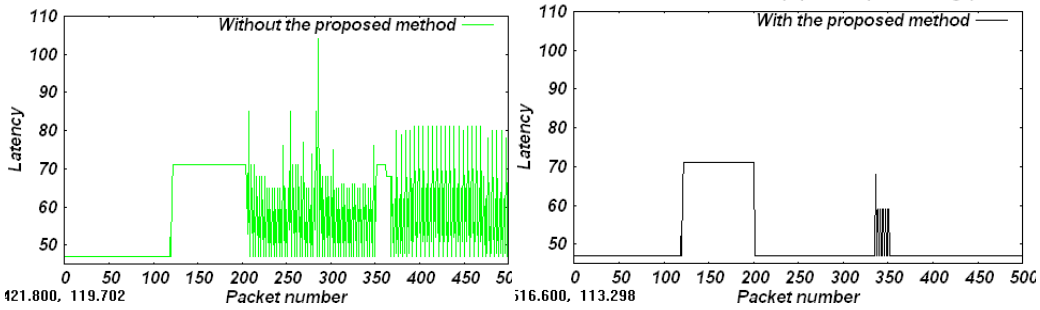


Fig. 3 – Latency values for S2→T2 flows, message with 8 packets.

On the other hand, for the *hot-spot* scenario it is possible to observe that the average latency decrease is in average 10% and the latency standard deviation in average 7%. In this scenario, congestion on specific regions of the network where introduced, and the method proposed detected and avoided these regions. Fig. 3 illustrates the latency for each packet for flow S2→T2. It is possible to note that from packet number 200, the latency reach its minimum value (Fig. 3(b)). In Fig. 3(a) the packets that passed through congestion points arrive to the target node with variable latency values, also introducing jitter.

4.2. Reaction to congestion events

The hot-spot scenario is the one which allows a greater path exploration. The routing mechanism reaction is the amount of packets sent with an undesired QoS level. We evaluate this metric according to the number of packets inside each message, i.e, the *granularity* of the algorithm. The worst case is for QoS messages with 32 packets (compare Fig. 3(b) to Fig. 4). This shows that long messages leads to a later treatment of congestion events, i.e., the reaction of the algorithm to congestion events take a longer time.

As shown in Fig. 4, the shortest message reaches minimal latency in packet 200. However, some greater latency values are observes from packets 330 to 350. The use of longer messages, as shown in Fig. 4, conducts to a slower reaction time. As the number of monitored events is also increased with longer messages, the algorithm is able to find a less congested path, suppressing the *noise* observed in Fig. 3.

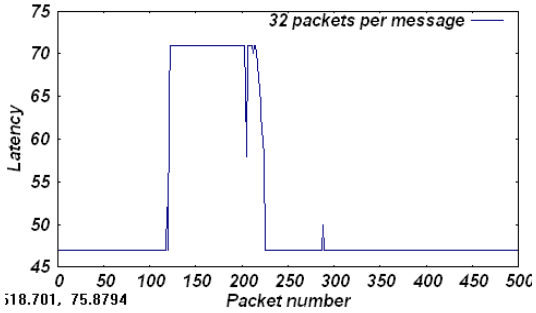


Fig. 4 – Latency values for S2→T2 flows, message with 32 packets.

5. Conclusions and future work

The original contribution of this research work is a new method for adaptive routing to be used in networks on chip. This method takes routing decisions based on the congestion path of each QoS flow, bringing to the routing algorithm a global view of the path being routed, not only neighbors routers status, as the state of the art proposals. The average and standard deviation on the packet latency show the effectiveness of the method when compared with a fixed routed flow. Reduction on the packet average latency and standard deviation was reached for the hot-spot traffic pattern. The use of messages of variable sizes also allowed the evaluation of the reactivity time of the algorithm.

Future works include comparison of the proposed method with other routing algorithms (which can use source or distributed routing). In addition to performance, the cost of the method in terms of area and power will be conducted. Experiments using other traffic distributions and real traffic scenarios are also part of future works.

6. References

[1] Marculescu, R. et al. “Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 28(1), 2009, pp 3-21.

[2] Tedesco, L. et al. “Application Driven Traffic Modeling for NoCs”. In: SBCCI’06, pp. 62-67.

[3] Ogras, U. Y.; Marculescu, R. “Analysis and Optimization of Prediction-Based Flow Control in Networks-on-Chip”. ACM Transaction on Design Automation of Electronic Systems, v.13(1), 2008, article 11, 28p.

[4] Mello, A. et al. “Rate-based Scheduling Policy for QoS Flows in Networks on Chip”. In: VLSI-SOC 2007, pp. 140-145.

[5] Nicopoulos, C.A et al. " ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers". In: MICRO’06, pp. 333-346.

[6] Hu, J.; Marculescu, R. “Dyad – Smart routing for networks on chip”. In: DAC’04, pp. 260-263.

[7] Lotfi-Kamran, P. et al. “BARP-A Dynamic Routing Protocol for Balanced Distribution of Traffic in NoCs”. In: DATE’08, pp. 1408-1413.

[8] Gratz, P. et al. “Regional Congestion Awareness for Load Balance in Networks-on-Chip”. In: HPCA’08, pp. 203-214.

Flow Oriented Routing for NoCs

Everton Carara, Fernando G. Moraes
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - prédio 32 - Porto Alegre - Brazil - CEP 90619-900
{everton.carara, fernando.moraes}@pucrs.br

Abstract

Several NoC routing schemes proposals targeting overall performance optimization are available in the literature. However, such proposals do not differentiate flows. The goal here is to demonstrate that adaptive routing algorithms can be used in flows with temporal constraints, enabling an enhanced degree of path exploration. Flows with no temporal constraints should use a constrained, deterministic version of the same adaptive routing. The main contribution of this work is to expose the routing algorithm at the IP level. The obtained results show significant gains in latency, throughput and jitter for hotspot scenarios.

1. Introduction

Several works propose routing schemes to achieve the advantages of deterministic/adaptive and minimal/non-minimal routing algorithms on a same router/NoC. In common these approaches have as their main goal to avoid congested regions using alternative paths provided by some adaptive routing scheme. In general, the router congestion level is obtained by input buffers monitoring later spread to adjacent routers by means of dedicated sideband signals. When a new packet arrives to a given router, neighbors congestion levels are considered to help the routing algorithm avoiding congested routers in the path.

Nilsson et al. [1] propose the *Proximity Congestion Awareness* (PCA) scheme using the hot-potato routing scheme in the Nostrum NoC [2]. At each router, the routing decision is based on the neighbors' congestion signals, designated *stress values*. These values can be computed as the average number of packets coming from the neighbors over a sample period. A similar approach is presented by Ye et al. [3], where a proposed contention look-ahead routing scheme combines the advantages of wormhole switching and hot-potato routing. Congestion signals, allied to a penalty scheme are used to choose whether to send the packet towards a *profitable route* (minimal path) or a *misroute* (non-minimal path).

DyAD (*Dynamic Adaptive Deterministic*) [4], is a well-known routing scheme which combines the advantages of adaptive and deterministic routing algorithms. Also based on neighbors' router congestion levels, the router switches between adaptive and deterministic routing. In the presence of neighborhood congestion, the router uses adaptive routing, otherwise it employs deterministic routing. The used routing algorithm is the *odd-even* [5] in two minimal versions: adaptive and deterministic. Other two routing schemes very similar to DyAD are presented in [6] and [7]. In the former [6], the difference is the use of only two versions of the adaptive odd-even routing algorithm: minimal and non-minimal. The approach suggests the use of non-minimal routing when neighborhood congestion is detected, otherwise it suggests the use of minimal routing. The latter work [7] presents the DyXY routing scheme based on the XY routing algorithm. In this minimal routing scheme, while the packet is not aligned in X or Y axes, the next router is chosen based on the neighbors' congestion levels.

In all reviewed works, there is no flow differentiation implied. The main goal of them has been to improve overall performance as evaluated for example through latency and throughput experimental results. The efficiency of all methods is questionable, due to their reduced design space exploration. This paper proposes a new routing scheme, called *flow oriented routing*, to increase the performance of *specific* flows. In common with previous methods, our work also combines adaptive and deterministic routing algorithms. The main difference is the possibility to specify at run time if the routing will be adaptive, for QoS flows, or deterministic, for BE flows. This approach exposes the routing scheme to the IP level, allowing its combination with other mechanisms to improve soft QoS support.

2. Flow Oriented Routing

Flow oriented routing is a feature that can be added to any adaptive routing algorithm. The basic condition is that there exists a deterministic version of the selected adaptive algorithm. It can be proved that such a version always exist, by fixing a single path between each source/target pair from all paths usable by the adaptive algorithm. The proposed scheme allows the NoC to *simultaneously* provide adaptive and deterministic routing. As adaptive routing offers alternative paths, it can be applied to high priority flows while low priority flows are routed deterministically.

An important problem in adaptive routing schemes is the output port selection metric. As reviewed before, the common policy has been to use neighborhood congestion level as decision metric. As discussed earlier, such metric does not ensure a congestion-free path, since this is local information and may lead packets to not

locally visible congested areas. To reduce the area overhead, and keep the implementation as simple as possible, this work does not adopt local congestion detection. When more than one output port is available, the selected port is the one which leads to the shortest path to the target (non-minimal routing algorithm is adopted). If all output ports lead to shortest paths, the first free port is selected. The area overhead with regard to a router with congestion detection is very small (less than 1%), while the presented related works have an area overhead around 5%.

Several adaptive routing algorithms can be used, such as the turn model ones [8]. This work suggests as case study the Hamiltonian routing algorithm [9], due to its simplicity and the possibility to use it for multicasting without incurring in deadlock risk.

2.1. Hamiltonian Routing Algorithm

A Hamiltonian path for a graph is any path that visits every graph vertex exactly once. In the Hamiltonian routing algorithm, each NoC router receives a label. In a NoC with *N* routers, the label assignment is based on the router position on a Hamiltonian path, where the first router in the path is labeled *0* and the last one is labeled *N-1*. Fig. 1 illustrates a possible label assignment to routers based on a Hamiltonian path in a 4x4 mesh NoC. The labeling process divides the network in two *acyclic* and *disjoint* subnetworks. The *high-links subnetwork* (solid lines) contains all links whose direction is from lower-labeled routers to higher-labeled routers, and the *low-links subnetwork* (dashed lines) contains all links whose direction is from higher-labeled routers to lower-labeled routers.

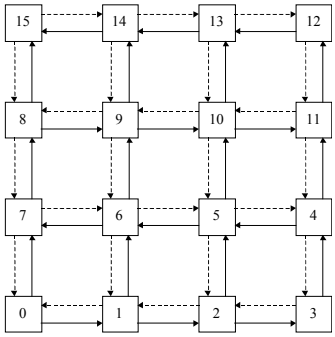


Fig. 1 - Example of label assignment based on a Hamiltonian path in a 4x4 mesh.

The *non-minimal partially adaptive* version of the Hamiltonian routing algorithm works as follows. A packet in a router with a label lower than the destination router is forwarded to any higher label neighbor router, which has a label lower or equal to the destination router. Consider, for example, the source router 6 and 14 as the target router. The possible paths taken by the packet are: {6, 9, 14}, {6, 9, 10, 13, 14}, {6, 9, 10, 11, 12, 13, 14}, {6, 7, 8, 9, 14}, {6, 7, 8, 9, 10, 13, 14} and {6, 7, 8, 9, 10, 11, 12, 13, 14}. In a similar way, when a packet is in a router higher than the destination router, it is forwarded to any lower neighbor router, which has a label higher or equal to the destination router. Consider, for example, the source router 13 and 7 as the destination router. The possible paths taken by the packet are: {13, 10, 9, 8, 7} and {13, 12, 11, 10, 9, 8, 7}.

To create a *minimal deterministic* version from the partially adaptive Hamiltonian routing algorithm, the forwarding condition can be restricted to "forward to the higher/lower neighbor router, which has a label lower/higher or equal to the destination router" (depending on the source and target labels). In the examples 6→14 and 13→7 the paths are respectively {6, 9, 14} and {13, 10, 9, 8, 7}.

In this paper, the Hamiltonian routing algorithm is used simultaneously in the two presented versions (*i*) non-minimal partially adaptive and (*ii*) minimal deterministic. To allow the routing engine to differentiate packets, one available bit of the packet header is defined as the *routing bit*. This routing bit is used to specify which version of the routing algorithm is executed.

3. Results

This section presents the evaluation of the flow oriented routing in a real NoC-based MPSoC platform. The evaluated performance figures are latency, throughput and *jitter*. The transmission of a given flow through the NoC may modify the original flow rate, inducing variable latency values, resulting in missed deadlines at the target IP. Jitter is this instantaneous variation in latency, and must be minimized in applications with QoS constraints.

3.1. NoC in a MPSoC

HeMPS [11] is a homogeneous NoC-based MPSoC platform. Its main hardware components are the Hermes [10] NoC and the Plasma-IP, which wraps a mostly-MIPS open source processor called Plasma, a network interface, a DMA module and a private RAM memory. The system contains a *master* processor responsible for managing system resources and *slave* processors responsible for tasks execution. In this experiment, the Hermes NoC was modified to support the flow oriented routing.

The target application is a MJPEG decoder operating on gray-coded images, partitioned into 9 tasks. The MJPEG task mapping is illustrated in Fig. 2, considering the deterministic version of the Hamiltonian routing algorithm. The Plasma-IPs connected to routers 3 to 7 execute only system debug, corresponding to a disturbing traffic. This scenario characterizes a hot spot region between tasks IVLC2 and IVLC3.

The MJPEG tasks communicate in a pipeline fashion. The *Start* task reads a compressed data stream and continuously sends 266-flit packets to task IVLC1. The remaining tasks inject 138-flit packets into the network. The disturbing tasks inject 50-flit packets. Due to the software tasks execution time (CPU bound), the used link bandwidth is very small (less than 3%).

Three scenarios are evaluated: (*i*) all flows routed deterministically; (*ii*) all flows routed adaptively; (*iii*) only MJPEG application flows routed adaptively.

Table 1 presents the latency and throughput results for all MJPEG flows. Latency results are in clock cycles and throughput corresponds to a percentage of the total link bandwidth. Total time is the time spent to finish the application execution in clock cycles. The total execution time without disturbing traffic is 2,051,939 clock cycles (ideal execution time).

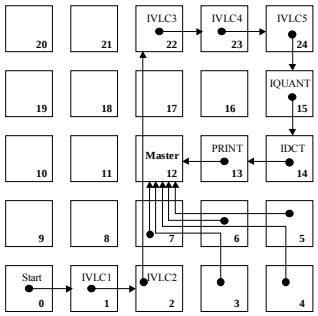


Fig. 2 - MJPEG mapping and flow spatial distribution.

Table 1 – Latency and throughput results for MJPEG running on the HeMPS MPSoC.

	Routing					
	Deterministic		Adaptive		Flow Oriented	
	Latency	Throughput	Latency	Throughput	Latency	Throughput
Start → IVLC1	477.57	2.37%	479.6	2.64%	477.4	2.7%
IVLC1 → IVLC2	349.5	1.23%	349.4	1.28%	349.5	1.41%
IVLC2 → IVLC3	**1621.2**	**1.23%**	**1668.66**	**1.28%**	**377**	**1.41%**
IVLC3 → IVLC4	350	1.23%	349.9	1.28%	349	1.41%
IVLC4 → IVLC5	349.5	1.23%	349.4	1.28%	350	1.41%
IVLC5 → IQUANT	349	1.23%	349	1.28%	349	1.41%
IQUANT → IDCT	349.5	1.23%	349.5	1.28%	349.5	1.41%
IDCT → PRINT	350	1.23%	350	1.28%	350	1.41%
Total time (cycles)	**3,932,709**		**2,209,991**		**2,078,633**	
Execution time overhead	**91.66 %**		**7.7 %**		**1.3 %**	

Comparing adaptive and deterministic routing, the latency of flow IVLC2→IVLC3 is not reduced, since the disturbing traffic competes with this flow in both scenarios. The advantage of adaptive routing with regard to deterministic routing is in the total execution time reduction. In the deterministic scenario, due to higher contention, IVLC2 injects packets only after some debug tasks finish and contention reduces. This is why it takes a long execution time with an average latency similar to deterministic routing. The flow oriented routing can avoid the hot-spot region, reducing latency to values near to other MJPEG flows, with increased throughput. The execution time overhead here is only 1,3%.

Fig. 3 presents the jitter evaluation. As in the previous experiment, the use of adaptive routing for all flows does not suppress jitter. On the other hand, flow differentiation can suppress jitter, as illustrated in this experiment.

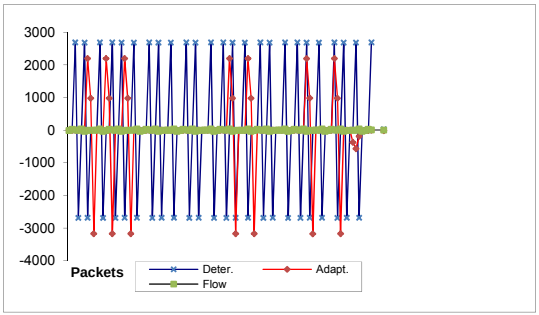


Fig. 3 – MJPEG experiment jitter evaluation, in clock cycles.

A naïve interpretation could assume that in a real MPSoC with most tasks running in software only a small interference of disturbing traffic would occur, due to the huge available NoC bandwidth and small injection rates (as shown in columns *throughput* of Table 1). The presented experiment demonstrates that in a real MPSoC, disturbing traffic can severely impact latency, execution time, as well as jitter, important performance figures in applications with QoS constraints.

4. Conclusions

The main contribution of this work is to display the benefits of exposing the NoC routing algorithm to the IP level, in the same way as achievable through arbitration (use of priorities) and switching method (circuit or packet switching). The proposed method is *general*, not targeted to a specific NoC, neither to a specific routing algorithm. A typical wormhole packet switching NoC and the Hamiltonian routing were employed only to demonstrate the effectiveness of the method in practice.

The main idea behind flow oriented routing is to restrict path exploration to flows with performance requirements. There is no reason to enable BE flows to be routed adaptively, since they can in this way unnecessarily disturb flows with QoS constraints. Therefore, the evaluation of new routing proposals should change the way to evaluate performance, since for example reducing overall latency could not correspond to getting closer to fulfilling real-time requirements in a MPSoC with multiple applications running simultaneously.

5. References

[1] Nilsson, E.; Millberg, M.; Oberg, J.; Jantsch, A. *"Load distribution with the proximity congestion awareness in a networks on chip"*. In: Design Automation and Test in Europe (DATE), 2003, pp. 1126-1127.

[2] Kumar, S.; Jantsch, A.; Soininen, J.-P.; Forsell, M.; Millberg, M.; Oberg, J.; Tiensyrja, K.; Hemani, A. "*A Network on Chip Architecture and Design Methodology"*. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2002, pp.105-112.

[3] Ye, T.; Benini, L.; Micheli, G. *"Packetization and routing analysis of on-chip multiprocessor networks"*. Journal of Systems Architecture, 50(2-3), 2004, pp. 81-104.

[4] Hu, J.; Marculescu, R. "*DyAD-Smart Routing for Networks-on-Chip*". In: Design Automation Conference (DAC), 2004, pp. 260-263.

[5] Chiu, G. *"The Odd-Even Turn Model for Adaptive Routing"*. IEEE Transactions on Parallel and Distributed Systems, v.7(11), 2000, pp. 729-738.

[6] Sobhani,A; Daneshtalab, M.; Neishaburi, M.; Mottaghi, M.; Afzali-Kusha, A.; Fatemi, O.; Navabi, Z. "*Dynamic Routing Algorithm for Avoiding Hot Spots in On-chip Networks*". In: International Conference on Design and Test of Integrated Systems in Nanoscale Technology (DTIS), 2006, pp. 179-183.

[7] Li, M.; Zeng, Q.; Jone, W. *"DyXY – A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Networks on Chip"*. In: Design Automation Conference (DAC), 2006, pp. 849-852.

[8] Glass, C. J.; Ni, L. M. *"The Turn Model for Adaptive Routing"*. Journal of the Association for Computing Machinery, 41(5), 1994, pp. 874-902.

[9] Lin, X.; McKinley, P. K.; Ni, L. M. "*Deadlock-free Multicast Wormhole Routing in 2-D Mesh Multicomputers*". IEEE Transactions on Parallel and Distributed Systems, 5(8), 1994, pp. 793-804.

[10] Moraes, F.; Calazans, N.; Mello, A.; Moller, L.; Ost, L. "*HERMES: an Infrastructure for Low Area Overhead Packet-switching* Networks on Chip". Integration the VLSI Journal, 38(1), 2004, pp. 69-93.

[11] Carara, E.; Oliveira, R.; Calazans, N; Moraes, F. "*HeMPS - A Framework for NoC-Based MPSoC Generation*". In: IEEE International Symposium on Circuits and Systems (ISCAS), 2009, pp. 1345-1348.

Adaptive Buffer Size Based on Flow Control Observability for NoC Routers

[1]Anelise Kologeski, [1]Caroline Concatto, [1]Débora Matos, [1]Fernanda Kastensmidt, [1]Luigi Carro, [1]Altamiro Susin, [2]Márcio Kreutz
{alkologeski, cconcatto, debora.matos, fglima, carro, susin}@inf.ufrgs.br, kreutz@dimap.ufrn.br

[1]UFRGS – Federal University of Rio Grande do Sul (PGMICRO, PPGC), Porto Alegre, Brazil
[2]UFRN –Federal University of Rio Grande do Norte, Natal, Brazil

Abstract

The communication among cores of a MPSoC having reusable interconnections is being provided by Networks-on-Chip (NoCs). To meet the need for communication, a solution proposed is to have an adaptive router, in which each channel can have a different buffer size according to the system requirements. In this situation, if a channel has a communication rate smaller than its neighbor, it may lend some of its buffer units that are not being used and ensures performance during the execution of different traffic flow. This paper proposes a mechanism that verifies, during run time, the behavior of the data traffic in the adaptive router. From the observability of the data flow, the system uses a control equation that adapts itself to provide an appropriate buffer depth for each channel to sustain performance with minimum power dissipation.

1. Introduction

Nowadays, embedded devices use several Processor Elements (PEs) to run one application. The PEs can integrate complex Multiprocessor System-on-Chips (MPSoCs), which are responsible by run completely an application. To guarantee the communication network between the PEs, there are Network-on-Chips (NoCs) [1]. The NoC aims to solve the interconnections problem of older techniques, providing high parallelism, scalability and reusability. NoC designs typically target a specific application, and hence it is customized at design time to achieve best energy and performance. However, if the architecture allows dynamically reconfigurable network, then is possible get best results as latency, throughput and power dissipation, avoiding efficiently the redesign and handling for some situations not foreseen at design time [2].

One example of changes in the communication flows is shown in fig.1 (a), the task graph of Xbox360 [3]. The task graph contains a platform with several PEs, and each PE has a throughput and bandwidth to communicate with other PE. Xbox is a video game, and hence the communication pattern can change according to the user behavior. When the user is saving the game, nine PEs are used, when photos are read from the USB1 others seven PEs are used and when the system is uploading some configuration, five PEs are used, as depicted in fig. 1(b), 1(c) and 1(d), respectively. The communication rates are expressed in Gbps and the arrows indicate the communication flow.

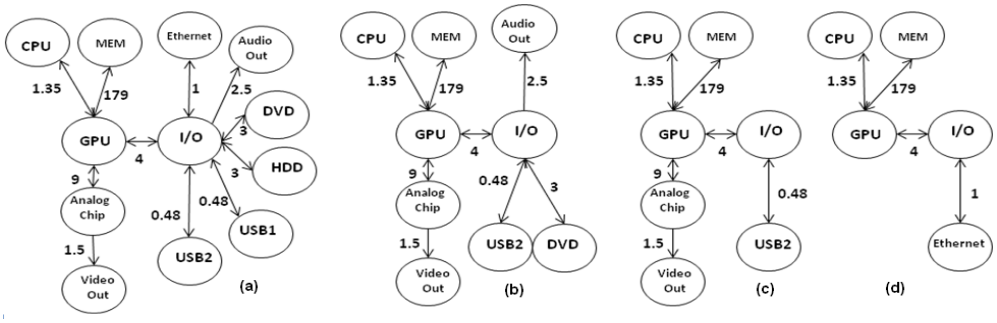


Fig. 1 - Four possibilities of traffic behavior using Xbox 360.

In [4] the buffer size of the NoC can change at run-time according to the need of the system, showing a better power-performance product, but without any internal control flow. Therefore, we propose a set of sensors and an automatic control mechanism to dynamically calculate the buffers depth for each input channel of the router, according to the network traffic and target application.

This paper is organized as follows. After presenting the related works in section 2, in section 3 we present the architecture of the adaptive router and the control system that is the main proposal of this paper. The performance and synthesis results are shown in section 4, and we conclude our paper in section 5.

2. Related Works

In the literature some works present solutions that address the need for adaptability. In [5] the authors propose an adaptive architecture with runtime observability. The adaptive process only occurs in the presence of a fault, and hence no performance or power reduction can be obtained during the normal system operation.

In [6] given the traffic of a target application and the total budget of the available buffering space, an algorithm optimizes the allocation of buffering resources across different router channels, while matching the communication characteristics of the target application. However, buffer sizing is developed at design time for each target application.

In [7] a NoC architecture with bidirectional channels that can be self-reconfigurable at run-time is proposed. This proposal allows transmit flits in the two directions, in order to increase the performance. The allocation of the channel is based only in the request. The main problem of this approach is that it does not consider the target application to distribute channels for each router, and it does not have a priority scheme to distribute these channels. If BiNoC uses the channels in only one direction at a time, this can decrease the performance of the network, since a channel with lower traffic get the channel before the router with a higher traffic.

The related works presented in this paper do not consider at the same time the traffic behavior and a policy to distribute the resources. This proposal cover exactly this gap, presenting an adaptive buffer slots allocation strategy based in a policy to distribute the buffer slots at run-time, according to the requirements of each channel of the router.

3. Adaptive Router Architecture

The proposed router architecture is a VHDL soft-core, with configurable channel width, input buffer depth and routing information width. The router used in this architecture is the RASoC (*Router Architecture for SoC*), and it composes the network-on-chip called SoCIN (*SoC Interconnection Network*) [8].

There is a limit in the increase of the buffer depth since in the worst case of communication scenario may compromise area and power. In this situation, if a channel has a communication rate smaller than its neighbor, it may lend some of its buffer slots that are not being used. This architecture is able to sustain performance due to the fact that not all buffers are used all the time. Each channel can have flits stored on its own buffer or in the left or right neighbor channels [4]. Fig. 2 shows an example of the buffer reconfiguration in a router according to a needed of bandwidth in each channel. First, a buffer depth with the same size for all channels is defined in design time. In this case, we defined the buffer size equal to 4, as illustrated in fig. 2(a). After, the traffic in each channel is verified and a control string defines the buffer depth needed in each channel, as showed in fig. 2(b). The distribution of the buffer words among the neighbor channels is realized as showed in fig. 2(c).

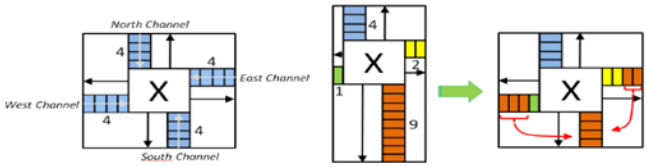


Fig. 2 - (a) Buffer depth original (b) necessary for traffic (c) and adaptive.

For sustain performance even for situations not foreseen at design time, the main strategy presented here is to observe the traffic behavior of the application, and size the buffer depth according to the traffic verified in the channels at run-time.

The traffic controller was implemented to each input channel of the router. Each input channel has a buffer that contains a FIFO to store the flits. In FIFO is controlled the input flow, the handshake it and the routing of the flits that arrive in the input channel. The *Buffer Depth Controller (BDC)* is new in this work and encloses three others blocks, they are the *Monitor*, *Integrator* and the *Buffer Slots Allocation (BSA)*, and are presented in the follow.

3.1. Monitor

The *Monitor* block observes the traffic of the channel, being basically a counter. Each one of the channel monitors verifies how many packets pass through its channel, and when the sum of all packets from a router reaches the limit value, a timer is activated. The timer defines the time in which the router must reallocate the buffer depth for each channel. When this value is reached, the buffers reallocation is made among the channels.

3.2. Integrador

The *Integrator* calculates the new buffer depth for each channel, according to the traffic behavior and the application. For the reallocation of buffers we use the simple first order control equation (1):

$$buffer\_need_{new} = (\alpha) \times buffer\_need_{old} + (1 - \alpha) \times traffic\_rate \quad (1)$$

The α value is defined according to the application, being a value between 0 and 1. To calculate $buffer\_need_{new}$ is used shifts for multiplication and sums. Equation 1 considers the past traffic in conformity with α, i.e., higher α values indicate that the past has a greater weight, and lower α values favor the instantaneous traffic occurrence. The buffer slot number used until the moment is $buffer\_need_{old}$, and *traffic_rate* indicates the traffic rate in the channel, measured by the quantity of packets that passing by it.

As the allocation of buffers is done with the borrow/lending process among the adjacent neighbor channels, the maximum number of buffer slots that each channel can take will be 3 times the original buffer depth defined in design time. If more than one neighbor channel needs to borrow some buffer slots, an arrangement among the channels will be necessary.

3.3. Buffer Slots Allocation

This block is the responsible to define and rearrange the final buffer depth for each channel of the router based in results obtained with the *Integrator* block. To allocate the appropriate buffer depth, the *Buffer Slot Allocation (BSA)* block receives the calculated buffer depth by *Integrator* of each adjacent channel, but this computed depth is not always possible. When two adjacent channels need to borrow buffer slots, they need to do it according to some priority and available free buffer slots in the neighborhood.

The *BSA* block is active after of the *Integrator* calculates the new buffer depth for each channel. It will allocate buffers following a policy. Firstly it checks if the buffer depth required is greater than the buffer depth defined in design time. In the affirmative case, the algorithm tries to borrow buffer slots from the right neighbor. A buffer slot is only borrowed when it is not needed by its own channel. When the channel cannot borrow buffer slots from the right neighbor, it tries to borrow buffer slots from the left neighbor.

It is possible that a channel needing a larger buffer size does not receive whole the buffer slots required. The new buffer depth will be the original depth defined at design time, plus the buffer slots that could be obtained from the neighbors or less depth borrowed. As each channel has a *BSA* block run in parallel, this system allows the rearrangement of buffers of the router in the same time. It guarantees that the sum of all buffer slots will never be greater than the sum of the buffer slots defined at design time for each channel.

The required time for the buffer slots allocation policy for each channel is very small. From the moment that the result of equation 1 is ready, in the worst case, when the channel needs to borrow buffer slots from the right and left neighbors to reach the required depth, is four clock cycles.

4. Results

4.1. Performance Evaluation

In our experiments were used a fixed-length packet with 80 flits and an 8 bits link size. We analyzed four situations for the Xbox 360, as shown in fig.1, and we used a 3x4 NoC mapped according to the need for communication. We call each one of these traffics as Xbox1 (v0), Xbox2 (v1), Xbox3 (v2) and Xbox4 (v3), respectively. A cycle-accurate traffic simulator described in Java was utilized to evaluate the average latency and the throughput of the network. Fig. 3 shows the results of latency for the adaptive and homogeneous router. The homogenous router presents all channels with a fixed buffer size equal to 4. The adaptive router uses a buffer depth allocator with α equal to 0.125. In addition, the buffer depth is monitored and changed at run time when 128 packets pass through a router.

In this experiment we considered the possibility of traffic shown in fig. 1 running in a sequence of events: first runs Xbox1 (v0), then run Xbox2 (v1), after Xbox3 (v2) and finally Xbox4 (v3). Each initial buffer size is the depth obtained in the last traffic behavior. We can observe in fig. 3 that, with for the adaptive router, the average latency was reduced and that the buffer depth influences the average latency of the network. The adaptive router shows approximately 75 % of reduction in the average latency when compared with homogenous router with buffer size equal to 4.

Following the same configuration previously adopted, we verified the throughput of these experiments. Fig. 4 shows the throughput results for the homogenous and proposed router considering the traffic behavior of Xbox1 and all 60 channels of 3x4 NoC. We observed that the throughput increases on average 4.5 times using the adaptive router with the buffer depth allocator instead than the homogeneous router. This experiment shows that for the same buffer size, the adaptive router is more utilized and presents a higher throughput.

4.2. Area, Power and Frequency Results

We used the ModelSim tool to simulate the code. We analyzed the average power consumption results to a CMOS 0.18um process technology using the Synopsys Power Compiler tool. Table I presents the results obtained with this analysis. The power results were obtained using the maximum frequency of each architecture. In this case, to reach the same average latency obtained with the adaptive router, the homogeneous router needs much larger buffers. For the four traffic patterns used of the Xbox application, the homogeneous router needs to have approximately a buffer depth equal to 9, in comparison to a buffer depth equal to 4 in the adaptive router with α equal to 0.125.

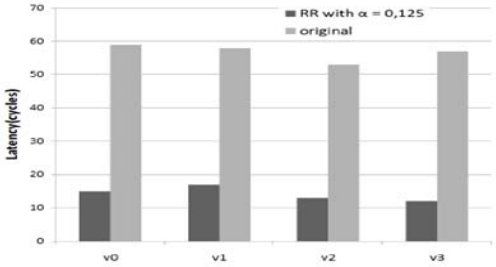


Fig.3 - Latency results for homogeneous and our proposed routers using the traffic behaviors for Xbox 360 with buffer depth equal to 4.

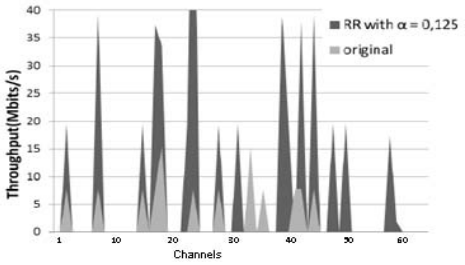


Fig.4 - Throughput results for homogeneous and our proposed routers using Xbox1 with buffer depth equal to 4.

The adaptive router reduces the power consumption by 28% on average, and it uses 55% smaller buffer depths to reach the same performance results. For the same average latency, the frequency of the adaptive router is higher than the homogeneous router, however the area consumption is larger, due to the extra hardware used to define the buffer depth allocation at run-time.

Tab.1 – Results to the adaptive and homogeneous router architecture.

Architectures	***Buffer Depth***	***Area (μm^2)***	***Maximum Frequency (MHz)***	***Power Consumption @ Max. Freq (mW)***
Homogeneous Router	4	79.130	265	6,48
	9	138.175	222	13,33
Adaptive Router with α = 0,125	4	254.957	224	9,49

5. Conclusion

In this paper we shown an automatic controller to dynamically allocate the buffer depth according to the traffic measured in each channel of the router. The buffer depth is obtained from a borrowing/lending process among the adjacent channels. The proposed architecture presents gains in throughput and average latency when we consider homogeneous and adaptive architectures. A same NoC can be used to different traffic behaviors, without presenting any performance loss.

As a future work we are planning to do more experiments with other NoC topologies and with different applications, study fault tolerance in architecture and analysis a best time for update of the buffer depth, as well as study the use of clock gating to save power.

6. References

[1] Dall'Osso, M., Biccari, G., Giovannini, L., Bertozzi, D., Benini, L., Tavel, P., "Xpipes: A latency insensitive parameterized network-on-chip architecture for multiprocessor SoCs," Proc. 21st International Conference on Computer Design. ICCD'03. 2003. pp. 536-539.

[2] Matos, D., Concatto, C., Kologeski, A., Kreutz, M., Carro, L., Kastensmidt, F. and Susin A., "Adaptive Router Architecture Based on Traffic Behavior Observability," Proc. 2nd International Workshop on Network on Chip Architectures, International Symposium on Microarchitecture, 2009, pp. 17-22.

[3] Andrews, J., Baker, N., "Xbox 360 System Architecture," IEEE Micro, 2006, Vol. 26. no. 2, pp. 25—37.

[4] Concatto, C., Matos, D., Carro, L., Kastensmidt, F., Susin, A., Kreutz, M., "NoC Power Optimization Using a Reconfigurable Router," IEEE Computer Society Annual Symposium on VLSI, 2009, pp. 235 – 240.

[5] Al Faruque, M.A., Ebi, T., Henkel, J., "ROAdNoC:Runtime Observability for an Adaptive Network on Chip Architecture," In IEEE/ACM International Conference on Computer-Aided Design, (ICCAD 2008), pp. 543-548.

[6] Jingcao, H., Ogras, U. Y., Marculescu, R., "System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design," In IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, 2006, pp. 2919-2933.

[7] Ying-Cherng Lan, Shih-Hsin Lo, Yueh-Chi Lin, Yu-Hen Hu, Soa-Jie Chen., "BiNoC: Bidirectional NoC Architecture with Dynamic Self-Reconfigurable Channel," In 3rd ACM/IEEE International Symposium on Networks-on-Chip, 2009, pp. 266-275.

[8] Zeferino, C., Susin, A., "SoCIN: A Parametric and Scalable Network-on-Chip," 16th Symposium on Integrated Circuits and System Design,2003, pp. 169-174.

Crosstalk Fault Tolerant NOC - Design and Evaluation

Alzemiro H. Lucas, Alexandre M. Amory, Fernando G. Moraes
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - prédio 32 - Porto Alegre - Brazil - CEP 90619-900
{alzemiro.silva, alexandre.amory, fernando.moraes}@pucrs.br

Abstract

The innovations on integrated circuit fabrics are continuously reducing components size, which increases the logic density of systems-on-chip (SoC), but also affect the reliability of these components. Chip-level global buses are especially subject to crosstalk faults, which can lead to increased delay and glitches. This paper evaluates different crosstalk fault tolerant approaches for Networks-on-chip (NoCs) links such that the network can maintain the original network performance even in the presence of errors. Three different approaches are presented and evaluated in terms of area overhead, packet latency, and residual fault coverage. Results demonstrate that the use of CRC coding at each link is preferred, when minimal area overhead is the main goal. However, each one of the methods presented here has its own advantages and can be applied depending on the application.

1. Introduction

Besides the communication infrastructure, an important SoC design challenge is the degradation of the signal integrity on long wires. Coupling capacitances tends to increase with the reduced components size. Faster clocks and lower operation voltage makes the delay induced by crosstalk effects even more critical, being the major source of errors in nanoscale technologies 0. Another noise sources that can produce data errors 0 are electromagnetic interference, radiation-induced charge injection and source noise.

Compared to buses, NoCs provide more opportunities to implement fault tolerance techniques for intra-chip communication. For instance, a NoC has multiple paths for any pair of modules, which can be exploited to improve the fault tolerance of the communication by using adaptive routing algorithms. Techniques based on codification for error detection/correction can also be applied for NoCs. Other approaches include place and route techniques to avoid routing of bus lines in parallel, changes in the geometrical shape of bus lines and addition of shielding lines between two adjacent signal lines. However, those techniques require advanced knowledge on electric layout design, and they are executed later in the design flow.

Considering these issues, bus encoding techniques represents a good tradeoff between implementation costs and design time to minimize crosstalk effects 0, and it is a technology independent mechanism to increase reliability on intra-chip communication. Table 1 summarizes the related work.

Table 1 – Comparison of related works.

Reference	Method	Implementation/ Evaluation	Metrics	Types of Faults
ZIM03 0	- CRC/Hamming on links - Fault Model - QoS	Implementation	- Residual error rate	Transient
BER04 0	- Source CRC - Switch-to-switch retransmission	Implementation	Not presented	Transient
VEL04 0	- Parity/Hamming on links - QoS	Implementation	- Latency - Power	Transient
MUR05 0	- End-to-end retransmission - Switch-to-switch retransmission: -Flit level -Packet level - Correction + Detection	Evaluation	- Latency - Power - Residual error rate	Transient
GRE07 0	- End-to-end retransmission - Switch-to-switch retransmission: -Flit level -Packet level -Retransmission with and without priority	Evaluation	- Message arrival probability - Average detection time - Average correction time	Transient

This paper evaluates error recovery mechanisms to increase the reliability of NoC links, making it resilient against crosstalk faults. As a design constraint, the evaluated mechanisms must be able to keep the NoC performance (latency, throughput, and bandwidth) in case of errors. Additional design constraints include low area overhead, high fault coverage, and minimum delay.

2. Crosstalk Fault Tolerant NoC Architectures

This paper presents three different strategies for fault tolerance on NoC links. All methods use as reference design the Hermes NoC 0.

2.1. NoC with link CRC

Figure 1 illustrates the first fault tolerant architecture implemented, which protects only the NoC links. This strategy provides router-to-router flit-level error detection and retransmission. This figure represents two adjacent routers, the sender and the receiver routers, and a link between them. Modifications compared to the non-fault tolerant design are in grey. It includes a CRC encoder at the sender, a CRC decoder and an error flip-flop at the receiver, additional signals *crc_in* and *error_out* in the link, and slight modifications in the input buffers.

When the sender sends a flit to the receiver, it encodes the CRC in parallel due to its combinational logic. The CRC is sent through the *crc_in* signal to the receiver, which decodes it and test for faults. If there is no fault, the flit (not the CRC) is stored in the receiver buffer. If some fault arrives, the flit is not stored into the buffer, and an error is signaled, through the *error_out* signal, back to the sender, which retransmits the last flit. This approach enables error recovery in one clock cycle.

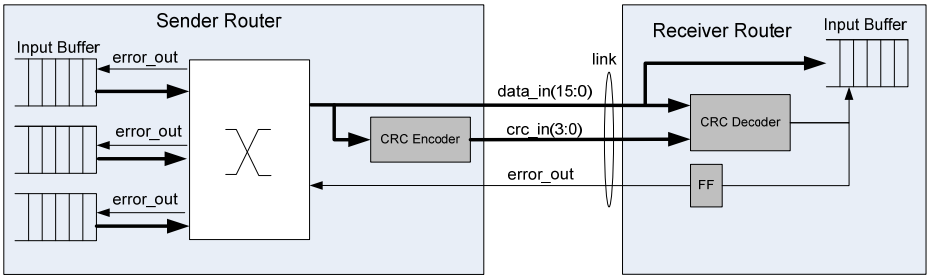


Figure 1 - Block diagram of the fault tolerant NoC design based on CRC for links.

The benefit of this approach is that the buffers and the buses width inside the router remain unchanged, saving silicon area. Only the external router interface receives new signals for error detection and recovery. The impact in silicon area, power, and delay is smaller. There is no impact on the latency when the network has no faults. Under a faulty condition the latency is incremented by one clock cycle only, which is an advantage compared to an approach based on end-to-end retransmission. This approach also provides the following additional advantages:

- It is not necessary to store full packets at each router, enabling the use of wormhole packet switching, reducing area, power and latency compared to store-and-forward or virtual cut- through;
- This method is faster compared to full packet retransmissions, since once an error is detected, the flit can be retransmitted in the next clock cycle;
- Error detection occurs before routing decision, making the network resilient against misrouting due to header flit errors;
- Flits to be retransmitted are available at the sender routers buffers, inducing smaller area overhead.

2.2. NoC with source CRC

This section presents the second fault tolerant NoC architecture, which is illustrated in Figure 2. This figure illustrates a path from the source router to a given router located in the path to the destination router. Modifications compared to the non-fault tolerant design are in grey. It includes a CRC encoder at the sender node, a CRC decoder and an error flip-flop at the receivers (intermediate routers and destination node), additional signals *crc_in* and *error_out* in the links, and a slight modification in the input buffers.

The main modification compared to the design in Section 2.1is that the CRC encoder is located only at the Network Interface (NI) of the module connected on the router local port. The CRC bits became part of the flit, increasing the width of all buffers of the network, as well as the internal buses of the routers, so that the CRC bits are carried through the network as part of the packet. As can be seen, the *data_out* signal have now 20 bit instead of 16 as the previous network, because this signal includes the *crc_out* signal, presented before in Figure 1.

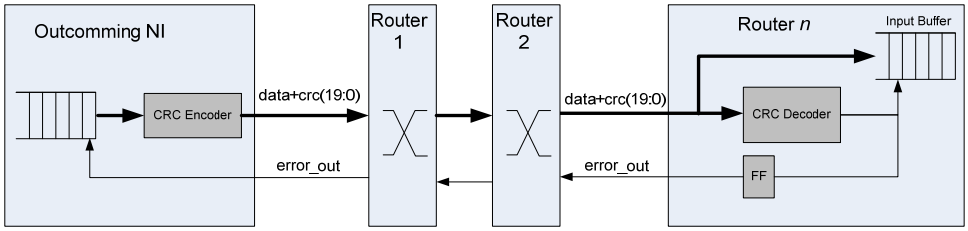


Figure 2 - Block diagram of the fault tolerant NoC design based on source CRC coder.

This network has the advantage of using four less CRC modules at each router (local port does not have CRC). On the other hand, it increases the buffer width, which increases the silicon area, the power consumption, and the delay of routers. This approach uses the same mechanism to recover the corrupted data from the input buffers at the previous router, thus, this network presents the same latency as the previous network to retransmit corrupted flits (one clock cycle).

Another advantage of this approach is that it can not only protect the links, but also protect certain internal logic of the router. It is possible to detect transient or even permanent faults in certain internal modules of the routers like the buffers and the crossbar, however, it cannot detect faults in most of the router control logic. Another limitation is that, if the fault is a bit-flip in a buffer, the error cannot be recovered. Therefore, other techniques for fault tolerance should be adopted.

2.3. NoC with Hamming on Links

Figure 3 presents a simplified structure of the network with Hamming code on links. This figure represents two adjacent routers (sender and receiver) where the link has been changed to carry Hamming parity bits. The sender has a combinational Hamming encoder at the output ports and the receiver has a combinational Hamming decoder at the input ports. Unlike the previous networks, none of the internal modules of the routers were changed.

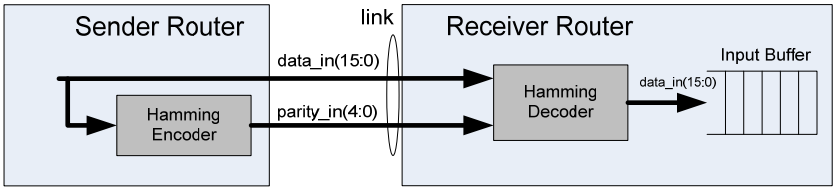


Figure 3 - Block diagram of the fault tolerant NoC design based on Hamming code.

The incoming flits of the receiver are first decoded and then saved in to the buffer. If there is a single fault the Hamming decoder corrects it transparently, without need of flit retransmission, thus, unlike the previous networks, this approach does not add latency to the network in a faulty situation. Note that there is no error signal from the receiver to the sender like the previous networks.

3. RESULTS

3.1. Area Overhead

Four networks have been implemented and synthesized using Cadence Encounter RTL Compiler (0.35um standard cells library) to evaluate the silicon area. Table 2 shows the results for a router with 5 ports and for an 8x8 network.

Table 2 - Area results for an 8x8 network (FT means Fault Tolerant).

	Original NoC	FT NoC Link CRC		FT NoC Source CRC		FT NoC Hamming	
	# of Cells	# of Cells	%	# of Cells	%	# of Cells	%
Router w/ 5 ports	3537	4025	13.8	4145	17.2	4149	17.3
- Buffer	547	590	7.8	630	15.3	547	0
- FT Logic	0	256	-	144	-	612	-
Total NoC cells	208864	236672	13.3	243968	16.8	243136	16.4

The total standard cell area increased 13.3% for the FT NOC with CRC in the links. This area overhead is due to the addition of CRC encoders and decoders at each router port.

The area overhead of the NoC with CRC computed at the source router is 16.8%. The area overhead for this

NoC comes from the increased buffer size (*flit* + CRC bits). As previously mentioned, this network can also provide some protection to transient faults on internal modules of the routers, justifying its use in designs where router fault tolerance is required.

The network with Hamming on links presented an area overhead of 16.4% compared to the original network. This overhead is due to the higher complexity of the Hamming decoding circuitry. The advantage of this technique is the error correction without retransmission, not interfering in the network latency in the presence of faults.

In conclusion, these results point out that a more complex code would probably not be an affordable fault tolerance technique for a NoC similar to Hermes 0. Perhaps, more complex NoCs could afford complex codes.

3.2. Latency Impact

Latency is evaluated using the following test scenario: (*i*) spatial traffic distribution: random destination; (*ii*) temporal traffic distribution: normal distribution, with an average injection rate of 20% and 10% of the available link bandwidth.

Table 3 presents results for the first scenario, with an average injection rate equal to 20% of the available link bandwidth. Each router sends 100 48-flits packets, resulting in 6,400 transmitted packets. Two error injection rates are adopted: 0.0717% (1,181 injected errors) and 2.03% (33,323 injected errors). As expected, both CRC architectures do not add extra latency in the absence of faults, and present the same average latency. The average latency increases 1.8% and 13% for a 1,181 and 33,323 injected faults respectively. The network with Hamming code does not add extra latency for error protection, however with higher error injection rates some faults are not corrected (residual faults).

Table 3 - Average latency, in clock cycles, for an injection rate equal to 20%.

Network	Transmitted Packets	Error Injection Rate (%)	Injected Errors	Avg. Packet Latency (clock cycles)	Latency Variation (%)
Original	6,400	0	0	839.39	0
Link CRC	6,400	0	0	839.30	0
		0.0717	1,181	854.77	1.8
		2.0300	33,323	948.47	13.0
Source CRC	6,400	0	0	839.30	0
		0.0717	1,181	854.77	1.8
		2.0300	33,323	948.47	13.0
Hamming	6.400	0.0717	1,181	839.39	0
		2.0300	33,323	839.39	0

The previous scenario with 20% of injection rate corresponds to a worst-case scenario where the network is congested. An injection rate of 10% of the available bandwidth, the second simulation scenario, corresponds to a more realistic NoC traffic behavior. In this simulation, each router sends 200 48-flits packets, resulting in 12,800 transmitted packets. Two error injection rates are adopted: 0.113% (3,689 injected errors) and 2.23% (72,392 injected errors). The results presented in Table 4 shows smaller latency values, due to the smaller congestion inside the network (such average value is near to the minimal latency value, 70 clock cycles). The average latency is in practice the same, with or without error injection at lower injection rates.

Table 4 - Average latency, in clock cycles, for an injection rate equal to 10%.

Network	Transmitted Packets	Error Injection Rate (%)	Injected Errors	Avg. Packet Latency (clock cycles)	Latency Variation (%)
Original	12,800	0	0	101.08	0
Link CRC	12,800	0	0	101.08	0
	12,800	0.113	3,689	101.11	0
	12.800	2.230	72.392	101.77	0

3.3. Residual Fault Analysis

This section shows the effectiveness of both methods to protect the network against crosstalk faults. In the context of this work, residual faults are flits that cannot be corrected due to the limitation of the code used to protect the network. Neither CRC or Hamming codes are able to detect 100% of the error patterns that can occur in a 16 bit bus, however CRC can detect flits with error in multiple bits, and Hamming is limited to errors on a single bit.

In this work we use Maximal Aggressor Fault Model (MAF) [1] to simulate fault injection on the network buses. To analyze residual faults using CRC and Hamming codes, the saboteur module is parameterized to inject faults varying the number of MAF model conditions to increase the error injection rate. The simulated

scenario considers a 5x5 network, with each IP sending 200 packets to a random destination, using 15% of the link available bandwidth. Table 5 shows the results of these simulations.

Table 5 – Residual fault analysis.

MAF Conditions	Transmitted flits	Injected errors	CRC Residual Faults		Hamming Residual Faults	
d_r	806,021	341	0	0%	0	0%
d_r, d_f	808,716	444	0	0%	0	0%
d_r, d_f, g_n	794,816	896	0	0%	0	0%
d_r, d_f, g_n, g_p	809,575	16,727	14	0.08%	389	2.32%

As expected, the CRC coding presents a lower residual fault rate, since its fault coverage is higher than the Hamming code. When all conditions of the MAF model are verified, a 2.32% and 0.08% residual fault rate is observed for the Hamming and CRC codes, respectively.

4. Conclusions and Future Work

The goal of this paper was to evaluate different crosstalk fault tolerant methods for network links such that the network can maintain the original network performance even in the presence of errors. Among the three evaluated architectures, the CRC applied at each link is the recommend method to protect the network against crosstalk effects. The *source CRC* penalizes area and power. On the other hand, the *source CRC* enables to protect some internal router components, since data transmitted through the router is protected. The assumed advantage of the Hamming codification, no retransmission required, presented a smaller fault coverage and higher area overhead compared to CRC, however it can be an interesting alternative for some applications, where a small number of data errors can be tolerated and the latency needs to be minimal.

It is possible to enumerate the following future works: (*i*) explore a *source Hamming* architecture, verifying the feasibility to use it for internal router protection; (*ii*) develop new methods to protect the router, minimizing the use of classical redundant approaches (TMR); (*iii*) evaluate and propose adaptive routing algorithms for faulty routers.

5. References

[1] Cuviello M.; et al. “Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects”. In: *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD’99)*, pp. 297-303, 1999.

[2] H. H. K. Tang, K. P. Rodbell, “Single-event upsets in microelectronics fundamental physics and issues”. In: *Materials Research Society Bulletin*, vol. 28, pp. 111–116, 2003.

[3] Bertozzi D.; “The Data-Link Layer in NoC Design”. In: Micheli G., Benini L.; Networks on chips: Technology and Tools. Ed. Morgan Kaufmann, 408 p, 2006.

[4] Zimmer H.; Jantsch A. “A Fault Model Notation and Error-Control Scheme for Switch-to-Switch Buses in a Network-on-Chip”. In: *Hardware/Software Codesign and System Synthesis* (*CODES+ISSS’03)*, pp. 188-193, 2003.

[5] Bertozzi D.; Benini L. “Xpipes: A Network-on-chip Architecture for Gigascale Systems-on-Chip”. IEEE Circuits and Systems Magazine, vol. 4, no. 2, pp. 18-31, 2004.

[6] Vellanki P.; et al. “Quality-of-Service and Error Control Techniques for Network-on-Chip Architectures”. In: *Great Lakes Symposium on VLSI (GLSVLSI’04)*, pp. 45-50, 2004.

[7] Murali S.; et. al. “Analysis of Error Recovery Schemes for Networks on Chips”. IEEE Design and Test of Computers, vol. 22, no.5, pp. 434-442, 2005.

[8] Grecu, C.; et al. “Essential Fault-Tolerance Metrics for NoC Infrastructures”. In: *IEEE International On-Line Testing Symposium (IOLTS 2007)*, pp 37-42, 2007.

[9] Moraes F.; et al. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. Integration, the VLSI Journal, vol. 38, pp. 69-93, 2004.

Arithmetic and Digital Signal Processing

Radix-2 Decimation in Time (DIT) FFT Implementation Based on Multiple Constant Multiplication Approach

[1,2]Sidinei Ghissoni, [3]Eduardo Costa, [4]José Carlos Monteiro, [4]Cristiano Lazzari
[1]Ricardo Reis
{sghissoni,reis}@inf.ufrgs.br, ecosta@ucpel.tche.br,{jcm,lazzari}@inesc-id.pt

[1] Universidade Federal do Rio Grande do Sul - UFRGS
[2] Universidade Federal do Pampa - UNIPAMPA
[3] Universidade Católica de Pelotas - UCPEL
[4] Instituto de Engenharia e Sistemas de Computadores - INESC-ID

Abstract

This paper proposes the implementation of a radix-2 Decimation in Time (DIT) FFT using the Multiple Constant Multiplication (MCM) approach. The MCM problem has been largely applied to the reduction of the multipliers in digital filters. In MCMs, the operations over the constants are implemented by using addition/subtractions and shifts rather than with general multipliers. In FFT filters, the butterfly algorithm plays a central role in the complex multiplications by constants. Thus, the use of the MCM in the butterflies can reduce significantly the number of real and imaginary multiplications by constants. It can be obtained by sharing the twiddle factors of the butterflies as much as possible. In this work, we have implemented four stages of 16 bit-width butterfly radix-2 with decimation in time for a 16-point FFT, by using both the MCM and the general Booth multiplier. For each stage of the real and imaginary parts of the butterflies we were able to apply the sharing of partials coefficients using MCM. The results were obtained by synthesizing the circuits in the CADENCE Encounter RTL Compiler tool for the UMC130nm technology. Ours results show that reductions of 60% in area, 47% in delay, and 69% in the number of cells can be achievable by using the MCM in the butterflies of the FFT.

1. Introduction

The Fast Fourier Transform (FFT) is an important algorithm used in many DSP applications, such as audio and video process, wireless communication, and it is also found in modules of WLAN chips. The various existing FFT algorithms use the partitioning of the set of input samples into sequences of half of the length of the original sequence. This is done recursively until there are only sequences of length 2, where the input samples cannot be more partitioned. This algorithm is named radix-2 FFT [1]. Partitioning the input sequence into more than two subsequences leads to higher radices of the FFT algorithm, what leads to the increase of the number of arithmetic operations. The complexity of the FFT architecture design is mainly given by the multiplication of the inputs by a large number of coefficients.

In the last decades, a large amount of researches has focused the implementation of efficient multipliers, in order to optimize area, delay and power consumption of this arithmetic operator. However, in the case of the FFT circuit, where a large amount of complex multiplications are performed by the butterflies, even the use of these efficient multipliers has not enabled optimizations in fully-parallel FFT architectures.

The multiplication of a set of constants by a variable, i.e., the Multiple Constant Multiplication (MCM) approach has enabled significant impact on the design of Digital Signal Processing (DSP) area. In the MCM operation, each constant is implemented using only addition/subtraction and shift operations rather than using a general multiplier for each constant. A large amount of algorithms has been proposed to optimize the multiplier block in digital filters by using the MCM approach [2]. However, only a few of them has been applied to FFT. Thus, we propose to optimize a radix-2 DIT butterfly by using the MCM approach in order to allow for further implementation of an efficient FFT architecture. We have used the algorithm of [2] for the generation of the tree of adder/subtraction and shift operations.

This paper is organized as follows: in Section 2, we present an overview of Multiple Constant Multiplications with FFTs algorithms. In section 3 is the presented the application of this approach MCM in a butterfly. In Section 4 some result comparisons between butterflies using MCM and booth multiplier are presented. Finally in Section 5, we present the conclusions of this work and some ideas for future works.

2. Application of the MCM Problem in the FFT Architecture

The Fast Fourier Transform (FFT) algorithm is a simpler form to calculate the Discrete Fourier Transform (DFT) efficiently. In the last years, many algorithms have been proposed for the improvement of performance of the FFT butterflies. Equation 1 presents the radix-2 butterfly with decimation in time [1], where the N^2 multiplications obtained in the direct DFT are reduced to $\log_2 N$ multiplications. This aspect enables a real increase of computational performance in the solution of the Fourier transform. The FFT can reduce the

computational complexity of the DFT, because its butterfly can process the calculation of two samples at a time.

Several other algorithms were developed to further reduce the computational complexity of the butterfly, such as radix-4, split-radix, radix-2^2, radix-2/4/8, and higher radix versions. However, all of them are based on the butterfly of [1].

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^K, K = 0,1,N-1 \quad (1)$$

where $W_n = e^{-j\frac{2\pi}{N}}$ is the twiddle factor.

2.1 Related Work

In the last ten years, several researches have been proposed algorithms for the optimization of the FFT architectures. In [3], the authors proposed a new radix-2/4/8 algorithm, which can effectively minimize the number of complex multiplications in pipelined FFTs. In [4], an optimization of FFT architecture based on multirate signal processing and asynchronous circuit technology is proposed. In [5], solutions based on parallel architectures for high throughput and power efficient FFT cores are presented. Different combinations of hybrid low-power techniques are exploited, such as the use of multiplierless units, which replace the complex multipliers in the FFTs, the use of low-power commutators, which is based on advanced interconnection, and the use of parallel-pipelined architecture. The method also uses MCM for the sharing of various multipliers that are located in the same stage of the hybrid architectures. However, this methodology is not efficient and it is not able to obtain the same reduction of area that we present in this work, and in some cases, area penalties can be observed.

In [6], it is proposed the optimization of the twiddle factors using trigonometric identity for few points of FFT architecture. The presented methodology proposes the replacement of the adders of the circuits by the use of multiplexers. Based on the same idea, the work of [11] proposes a Low-Complexity Reconfigurable Complex Constant Multiplication for FFTs for the reduction of area for a larger number of points (32 points). This new methodology proposed by [6] and [11] was compared against the works [7-10], where reductions in terms of the number of adders could be achievable. Although the authors of [6] and [11] do not present power and delay results in their work, they comment that probably these metrics may lead to large circuits, due to the limitation of the proposed architecture where only perform the FFT computation in serial form.

Although there are several techniques to reduce the complexity of the FFT architectures, only a few of them uses the MCM approach for the sharing of the real and imaginary twiddle factors in the butterflies. In our work, we presented the use of MCM algorithm proposed in [2] in order to reduce the complexity of the radix2 butterfly of the FFT with decimation in time.

3. Implementation of the 16-point Radix-2 DIT FFT

In fig. 1, it is presented the flow for the fully-parallel 16-point radix-2 FFT architecture with decimation in time. The FFT is divided into four stages and each one of them is composed by eight butterflies. The butterflies allow the calculation of complex terms, where one complex addition, one complex subtraction and one complex multiplication are involved in the butterfly block. The blank rectangles shown in fig. 1 represent circuit registers. The multipliers present in the butterflies were implemented by using both Booth multiplier and MCM approach.

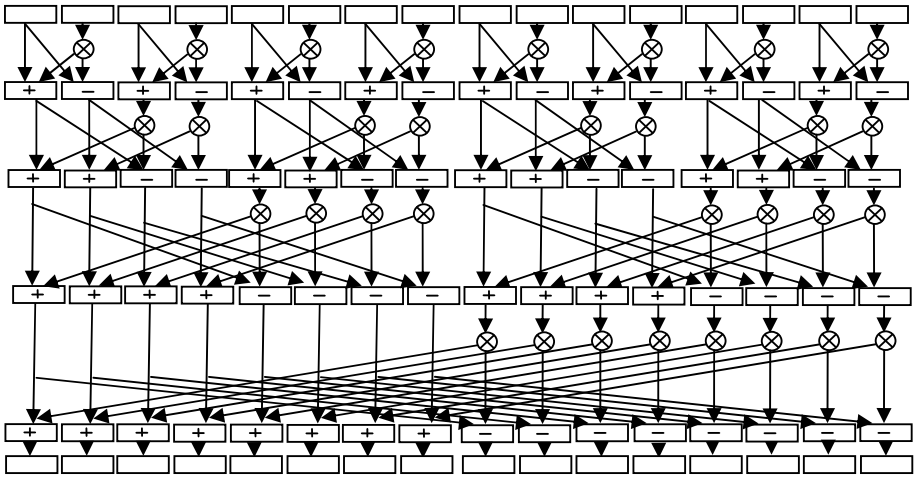


Fig. 1 –16-point radix-2 DIT FFT architecture

In a 16-point Fully-parallel FFT implementation, 32 real and 32 imaginary terms are performed in the butterflies (4 stages with 8 butterflies). Thus, the architecture presents large hardware requirements in terms of arithmetic operators. In order to reduce the number of operators, we have applied the MCM approach to the twiddle factors of the butterflies. Particularly, we have used the algorithms of [2], which enable the optimization of the number of operations under a delay constraint. Firstly, the algorithm removes the repeated twiddle factors in each stage. After the initial checking, it is possible to observe that, in fact, in the 16-point radix-2 DIT algorithm, only three different coefficients are generated (by considering that the negatives coefficients are easily obtained by the positive ones using the 2´s complement operation). Fig. 2(a) shows the structure of the butterfly for the radix-2 DIT algorithm and fig. 2(b) shows its resultant twiddle factors multiplications obtained by the application of the MCM algorithm presented in [2].

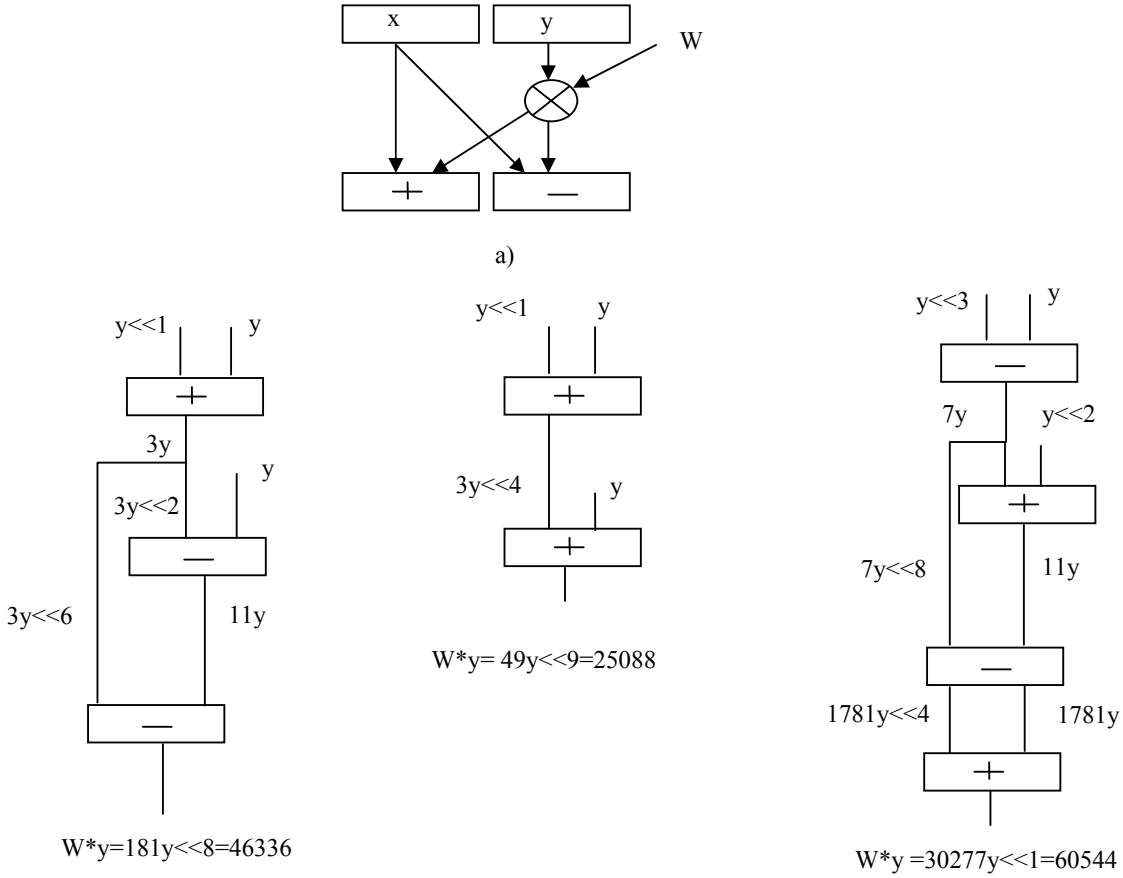


Fig. 2 – a) Structure of the butterfly for the radix-2 DIT algorithm. b) MCM approach applied to the twiddle factors.

4. Results

This section shows the obtained results of a 16-point and 16-bit radix-2 DIT FFT implemented in Vhdl. The Tab. 1 present area and delay results obtained from the implementation of fully-combinational booth multipliers and optimized multipliers based on the work proposed in guided from the MCM algorithm of [2]. The logic synthesis was performed with the CADENCE Encounter RTL Compiler tool for the UMC 130nm technology.

Tab.1 – 16-point and 16-bit Radix-2 DIT FFT results

Circuits	Area (mm2)	N° Cells	Path Delay(ns)	Adders/ Subtractors(16bits)	Multipliers (16 bits)
Booth16	0,405	25114	29	256	128
MCM	0,159	7713	15,5	456	0

In Tab. 1, it is possible to observe the large reduction in area that was obtained in the FFT architecture when using MCM. In fact, this occurs mainly due to the large reduction on the number of multipliers that is enabled by the use of MCM. As can be observed in Tab. 1, while no multipliers are used in the FFT when the MCM technique is used, 128 multipliers are used for the architecture with booth multiplier. In fact, this large number of multipliers is explained because 32 butterflies are used in the FFT architecture and each butterfly requires four multipliers for the real and imaginary parts calculation. On the other hand, the number of adders/subtractors is larger the FFT with MCM, because multipliers are implemented only with adders, subtractors and shifts. It is important to highlight that shifts are implemented by rearranging interconnections, and they are "free" in terms of occupied area. This aspect (the replacement of the general multipliers in series by adders/subtractors with a reduced logic depth) has also allowed a significant increase of performance in the FFT with MCM, as can be seen in Tab. 1, since the critical path is largely reduced in this architecture.

Power consumption is not taken into account in this work because it is not considered for the MCM algorithm applied[2]. Anyway, we expect to obtain the same reduction on power consumption as we have obtained on timing and area by the application of techniques based on power optimization.

5. Conclusions and Future Works

In this work we have applied the MCM approach in the butterflies of a 16-point radix-2 DIT FFT. The obtained results show that the use of the MCM can reduce area and increase performance of the fully-parallel FFT significantly, when compared to implementations with general booth multipliers. It is explained due to the fact that the multipliers can be replaced by adders/subtractors and shifts, when using the MCM approach. We used the algorithm proposed in [2] for the MCM implementation. Since this algorithm reduces the number of operations under a delay constraint, we were also enabled to increase the performance of the FFT significantly.

Although in this paper we have not taken into account the power consumption of the FFT, we are convinced that the large impact on area reduction provided by the MCM can allow a low power FFT implementation. Thus, as future work, we intend to verify the low power aspect that can be obtained by the use of the MCM in the fully-parallel FFT implementation.

6. References

[1] J. W.;Cooley, J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation", [S.l.], v.19, n.90, p.297–301, 1965.

[2] Aksoy, L., Costa, E., Flores, P. and Monteiro J., 2008. Exact and Approximate Algorithms for the Optimization of Area and Delay in Multiple Constant Multiplications, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(6), 1013-1026.

[3] L. Jia, Y. Gao, J. Isoaho, and H. Tenhunen, "A New VLSI-Oriented FFT Algorithm and Implementation," Proc. of Eleventh Annual IEEE Int'l ASIC Conference, 1998, pp. 337-341.

[4] K. Stevens and B. Suter, "A Mathematical Approach to a Low Power FFT Architecture," IEEE Int'l Symp. on Circuits and Systems, vol. 2, 1998, pp. 21-24.

[5] W. Han, T. Arslan, A. T. Erdogan and M. Hasan, "High-performance low-power FFT cores," ETRI Journal, vol. 30, no. 3, pp. 451–460, June 2008.

[6] J.-E. Oh and M.-S. Lim, "New radix-2 to the 4th power pipeline FFT processor,"IEICE Trans. Electron, vol. E88-C, no. 8, pp. 1740–1764, Aug. 2005.

[7] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," Circuits, Systems and Signal Processing, vol. 25, no. 2, pp. 225–251, Apr. 2006

[8] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," Circuits, Systems and Signal Processing, vol. 25, no. 2, pp. 225–251, Apr. 2006.

[9] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in Proc. IEEE Int. Symp. Circuits Syst., New Orleans, LA, 2007, pp. 1097–1100.

[10] Y. Voronenko and M. P¨uschel, "Multiplierless multiple constant multiplication," ACM Trans. Algorithms, vol. 3, no. 2, May 2007.

[11] F. Qureshi, Gustafsson, F. O." Low-complexity reconfigurable complex constant multiplication for FFTs " Circuits and Systems. ISCAS 2009. IEEE International Symposium on, pp. 1137-1140 May 2009.

Synthesis-Based Dedicated Radix-2 DIT Butterflies Structures for a Low Power FFT Implementation

[1]Mateus Beck Fonseca, [2]Eduardo A. César da Costa, [1]João Baptista dos S. Martins
beckfonseca@mail.ufsm.br, ecosta@ucpel.tche.br, batista@inf.ufsm.br

[1] Federal University of Santa Maria (UFSM)
[2] Catholic University of Pelotas (UCPEL)

Abstract

This paper addresses the architectural exploration of dedicated structures of Radix-2 Decimation in Time (DIT) butterflies for a low power Fast Fourier Transform (FFT) implementation. In the FFT computation, the butterflies play a central role, since they allow the calculation of complex terms. In this calculation, involving multiplications of input data with appropriate coefficients, the optimization of the butterfly can contribute for the reduction of power consumption of FFT architectures. In this paper different dedicated structures for the 16 bit-width pipelined radix-2 DIT butterfly operating on 100MHz are implemented, where the main goal is to minimize both the number of real multipliers and the critical path of the structures. For the logic synthesis of the implemented butterflies it was used Cadence tool Encounter RTL Compiler with XFAB MOSLP 0.18µm library. Cells count, Area and power consumption results are presented for the synthesized butterflies. For the power consumption, the results are obtained after using 100 000 inputs vectors at the inputs of the butterflies.

1. Introduction

Power consumption in VLSI Digital Signal Processing Systems (DSP) has gained special attention mainly due to the proliferation of high-performance portable battery-powered electronic devices such as cellular phones, laptop computers, etc. In DSP applications, Fast Fourier Transform (FFT) is one of the most widely used algorithms [1].

FFT is the largely implementation of the Discrete Fourier Transform (DFT), because this algorithm needs less computation, achieved by a recursive implementation of an operator named butterfly. This operator performs the calculation of complex terms, which involves the multiplication of input data by appropriate coefficients [2].

FFTs can be computed with Decimation in Time (DIT) or with Decimation in Frequency (DIF) techniques. However, in this paper, only the implementation of DIT butterflies structures is taken into account.

Sixteen mathematical forms of describing a complex multiplication are presented in [2]. However, some of these structures are quite the same, since the only difference is in the operations signal or in the input positions. Thus, for the sake of hardware complexity analysis, only a smaller number of structures are different and therefore actually need to be compared.

In this work, the main goal is to reduce the number of real multipliers in the original DIT butterfly, which is composed by four real multipliers to compute one complex multiplication.

Since the reduction in the number of multipliers is the main aspect to the power consumption reduction in the butterflies, we have proposed a novel structure by using three multipliers. As will be shown, although the new structure is not able to reduce the power consumption in the butterfly yet, it allows the employment of other techniques that can contribute for the power reduction in the proposed structure.

For the synthesis of the butterflies, Cadence tool Encounter RTL Compiler with XFAB MOSLP 0.18µm library has been used with clock constraints at 100MHz. Area and power results are presented for the butterflies. The results show that from the five analyzed structures, the structure with four multipliers is the one that presents the lowest power consumption, due to the smaller critical path along the structure.

The rest of the paper is organized as follows. The next section makes an overview of relevant work related to our own. In Section III we present the main aspects of the FFT algorithm with Decimation in Time. In Section IV we present the implemented structures of the butterflies, including our proposed structure. Area and power results for the butterflies obtained from logic synthesis tool are presented in Section V. Finally, in Section VI we conclude this paper, discussing the main contributions and future work.

2. Related Work

Most of the FFT implementations presented in literature use four real multipliers to perform the complex multiplication of the radix-2 butterfly. On the other hand, there are few works reporting the use of less than four real multipliers. For instance, in [3][3] sixteen ways to perform a complex multiplication using three real multipliers are presented. Although the analysis is focused on digital filters and not on FFT implementations, it can be naturally applied to the structures of the butterflies, which are composed by complex multiplications. The work of [4] introduces an unified approach for the development of the radix-2/4 decimation-in-time Fast

Hartley Transform (FHT) and FFT algorithms. However, performance and power results from the proposed structures are not presented. An FFT/IFFT design based on DIT algorithm is also presented in [5]. However, the results are only focused on FPGA, where power consumption is not considered.

The works of [4] and [5] compare radix-2 DIT FFT butterflies in terms of area, delay and power consumption for one structure using three and four real multipliers. The butterfly structures are implemented based on CORDIC, Distributed Arithmetic and bit parallel multiplier. The butterflies were synthesized onto a 0.11μm technology and the authors concluded that the butterfly based on bit parallel multiplier is power efficient, but only for modest FFT sizes (less than 512 points).

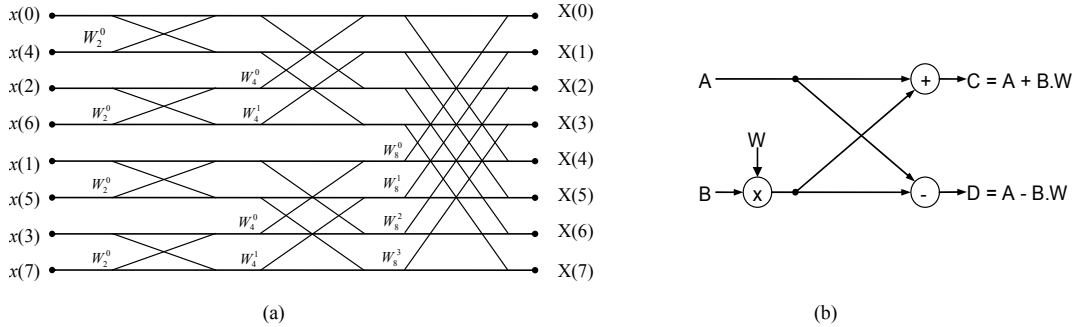
In this work we explore the different ways of implementing butterflies based on the strategies of [3]. We also propose a new form of implementing the butterfly by using three real multipliers. As in [4] and [5], we investigate the implementation of radix-2 DIT butterflies using three and four real multipliers, but we have done it for all the structures using only three multipliers. We explore the tradeoff between the number of multipliers used in the butterflies and the reduction of the critical path of the structures. However, as the opposite of the works of [6] and [7], which use bit parallel multiplier, Distributed Arithmetic (DA) and CORDIC, we are using into the butterflies, the own multiplier synthesized by the Cadence tool.

3. Radix-2 FFT with Decimation in Time Algorithm

The main goal of the FFT algorithms is to compute the Discrete Fourier Transform efficiently [3]. The FFT $X(k)$ of a signal $x(n)$ can be computed using (1), where W_N is named twiddle factor, i is the imaginary component and N is the number of points of the FFT.

$$\begin{aligned} &X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{k \cdot n}, \\ &W_N = e^{-i2\pi / N} \\ &\forall\ 0 \le k \le N-1 \quad e \quad \forall\ 0 \le n \le N-1 \end{aligned} \tag{1}$$

In [4] it is proposed a flow to process the FFT algorithm. This flow can be seen in fig. 1(a) for the example of an 8-point radix-2 FFT with decimation in time.



(a) (b)

Fig. 1 - (a) 8-point radix-2 DIT FFT flow and (b) Butterfly structure of the radix-2 DIT

The hierarchical computational blocks in the FFT structure are stages, groups and butterflies. Each stage requires the computation of groups, and each group requires the computation of butterflies. The butterfly plays a central role in the FFT computation. For the radix-2 FFT algorithm with decimation in time, the butterfly allows the calculation of complex terms according to the Fig. 1(b). The number of butterflies which are needed to calculate an N-point FFT is given by $(N/2) \log 2\ N$.

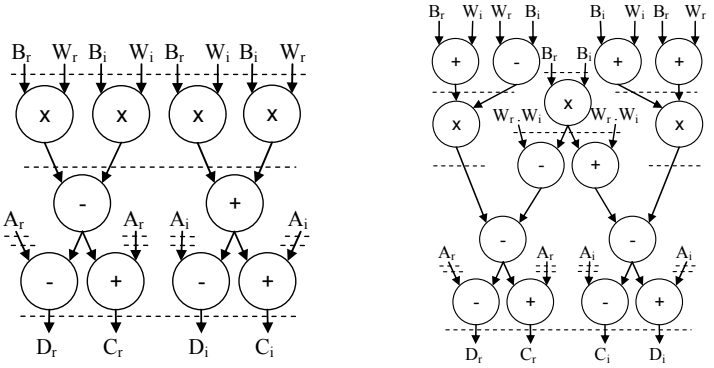
According to the structure shown in fig. 1(b), the butterfly is composed by addition, subtraction and multiplication, where these operations involve complex numbers. These arithmetic operations enable the calculation of the real and imaginary parts. The original structure of the butterfly involves four multipliers, three adders and three subtractors for the calculation of the real and imaginary parts. In this work, we implement and test some butterfly structures and we propose a new form of representing it.

4. Radix 2 DIT Butterfly Structures

Butterfly structures have one complex multiplication operation, and it can be done with three or four real multipliers. In this section we present the original structure of the butterfly. The structures by using some way of representing complex multiplication of [3] and the new proposed structure for the butterflies are also presented in this section.

4.1. Dedicated Butterfly Structures

Fig. 2 (a) shows the most common structure of the radix-2 DIT butterfly for FFT with four multipliers and six adders/subtractors operators based on the structure shown previously in Fig. 1(b). The dotted lines in fig. 2 and fig. 3 indicate the place of the registers used to create the pipelines.

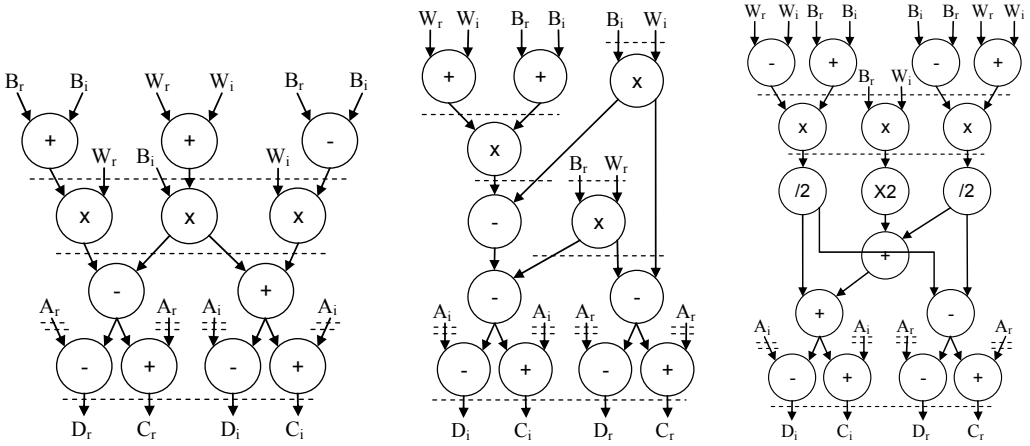


(a) (b)

Fig. 2 - (a) The original butterfly structure – Structure A, and (b) The proposed butterfly structure - Structure E

A new proposed form of representing the radix-2 DIT butterfly for FFT with three multipliers is presented in fig. 2(b). In this text we will report it as structure E. As it can be observed, this structure has one less multiplier (compared against the original structure A) and the same critical path presented by the structure B (fig. 3(a)), i.e. four circuits in the critical path. However, the new structure needs a previous multiplication that should be done and stored in Read Only Memory (ROM). Since the real and imaginary twiddle factors *Wr* and *Wi* are stored in ROM, thus their product (*Wr* * *Wi*) can be stored in ROM memory too.

According to [3], there are sixteen forms to compute a complex multiplication using three real multipliers. However, the hardware construction of some of them is very similar, since only swapping signals in sum operations and/or inputs positions are needed. Thus, it is possible to create three groups of different structures in physical implementation from the sixteen ones. These three groups of structures are represented in fig. 3.



(a) (b) (c)

Fig. 3 - The butterflies structures using the complex multiplication schemes of [2];
(a) Structure B; (b) Structure C; (c) Structure D

As can be observed in fig. 3, all the presented solutions lead to a structure with three multipliers. The number of adders and subtractors are almost the same in the structures. The structure D (fig. 3(c)) presents a decomposition of multiplication by 2 (represented by X2) and division by 2 (represented by /2). However, no additional hardware is required for these operations, since they can be implemented by using only wires. The main difference between the structures is in the critical path which is composed by four circuits in the structure B and five circuits in the structures C and D (not considering the X2 and /2 blocks in the critical path of the structure D, since they are only composed by wires).

5. Synthesis Results

All the developed architectures were described in Verilog with one and two pipelining stages and synthesized to XFAB MOSLP 0.18µm 1.8V library in typical operation conditions, using the Cadence Encounter RTL Compiler tool. The Leakage power for each structure was obtained in the order of nano Watts

(nW) scale, and thus suppressed in the results. Cells, area (in terms of number of cells and μm^2) and power consumption results, after switching activity analysis using 100 000 inputs vectors, are presented in Tab. 1.

The presented results in Tab. 1 show that the structure A presents the less power consumption among the structures. However, we have observed that the use of two levels of pipelining has lead to the significant power consumption reduction from the structures from B to E. This occurs because with the use of two levels of pipelining there is a combination of a large reduction in the critical path and the use of one less multiplier in these structures. In this way, the structure C could be an interesting choice, since it presents less area with almost the same power consumption of the structure A, when two levels of pipelining are taken into account.

Although our proposed structure E has not enabled gains in terms of area and power results, it has a potential for optimizations, because this structure presents the regularity aspect (presented by the structure A) and it allows the use of more efficient adder circuits, such as adder compressors.

Tab.1 – Synthesis Results for the 16-bit pipelined Butterflies Structures

	1 Pipelining stage			2 Pipelining stages		
Structure	**Cells**	**Area (μm^2)**	**Power (mW)**	**Cells**	**Area (μm^2)**	**Power (mW)**
A	1765	45126	5.931	1958	47951	6.287
B	1943	42313	10.577	1710	45058	7.049
C	1895	41695	8.135	1841	45802	6.470
D	2279	44213	11.668	1928	48019	7.829
E	2388	48339	11.571	2118	53015	8.293

6. Conclusions

In this work several architectures for the pipelined radix-2 DIT FFT butterflies were presented. The implemented butterflies structures were compared in three main scopes: i) the original structure by using four real multipliers; ii) some way of representing complex multiplication of [3], where three real multipliers are used; iii) one proposed structure with three real multipliers. The results we presented show that although the structure A drives the best power consumption, all the other architectures are potentially power consumption reduction, when two levels of pipeline are used, because they enable: i) the use of one less multiplier, and ii) the large reduction of the critical path. Although the proposed structure E has not presented reductions in terms of area and power consumption, its regular structure can allow some optimizations. Thus, as future work, we intend to test the use of more efficient full custom adder compressors in our proposed structure. We also intend to extend the tests of this work for some other larger radix-4 and radix-8 FFT butterflies.

7. References

[1] J. Cooley and J. Tukey, "An algorithm for the machine calculation of the complex Fourier Series" *Math. Comput.*, vol 19, 1965, pp. 297-301.

[2] Oppenheim and R. Schafer. "Discrete-Time Signal Processing". Prentice Hall Signal Processing Series. 1989.

[3] Wenzler and E. Lüder, "New structures for complex multipliers and their noise analysis," *in Proc. IEEE ISCAS*, vol. 2, Seattle, WA, 1995, pp. 1432–1435

[4] S. Bouguezel, M. Ahmad, and S. Swamy. An approach for computing the radix-2/4 DIT FHT and FFT algorithms using an unified structure. *IEEE International Symposium on Circuits and Systems. ISCAS* 2005. pp. 836-839. vol. 1. 2005.

[5] C. Concejero, V. Rodellar, A. Marquina, E. Icaya and P. Vilda. FFT/IFFT Design versus Altera and Xilinx Cores. *2008 International Conference on Reconfigurable Computing and FPGAs. ReConFig'08.* pp. 337-342. 2008

[6] J. Takala and K. Punkka. Scalable FFT Processors and Pipelined Butterfly Units. *Computer Systems: Architectures, Modeling and Simulation. Lecture Notes in Computer Science – LNCS*. vol. 3133. pp. 373-382. 2004.

[7] J. Takala, and K. Punkka. Scalable FFT Processors and Pipelined Butterfly Units. *Jornal of. VLSI Signal Process*. 43, 113-123 (Jun. 2006), 113-123.

Implementation of Adders Circuits Using Residue Number System - RNS

Alexsandro O. Schiavon, Eduardo, A. C. da Costa, Sérgio J. M. de Almeida
alexschiavon@gmail.com, {ecosta,smelo}@ucpel.tche.br

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

Abstract

With the growing in utilization of arithmetic operators in areas such as Digital Signal Processing – DSP, the study of techniques to improve the performance of these circuits, such as the encoding of the operands, has been taken into account. Thus, the main goal of this work is the study of adders using Residue Number System – RNS. This encoding technique uses the residues of the numbers in order to perform the mathematics operations. As the RNS divide the number in parts, and process each part in a separated way, it enables a reduction in the number of bits and a parallelization of the operations. In the case of addition operation, this aspect contributes for a significant reduction of carry propagation, which allows for an improvement of performance in this operation. For the circuits of DSP area, such as digital filters, that uses a large amount of additions and multiplications, the use of RNS can be a good alternative to improve the performance of these circuits. However, as will be presented in this work, the conversion stages between the binary and RNS representations represent a challenge for the improvement of performance in the arithmetic operators, because the gain in performance which is obtained in the modular addition stage is generally lost in the conversion steps.

1. Introduction

The Residue Number System - RNS is a non-weighted number system [1] It consists in the decomposition of a given binary number into smaller slices, such that the computation can be implemented in parallel [2]. This characteristic makes the RNS a good choice to increase the performance of some types of operations, such as addition, subtraction and multiplication. The main factor to determinate the performance of the RNS circuits is related to the choice of the *moduli* set. In this choice some aspects must be taken with special care. The *moduli* should be small in order to minimize the number of bits, but it must cover all the dynamic range of RNS [3]. This tradeoff must to be considered in order to implement a good hardware. In this paper, we use the classic *moduli* set $(2^n - 1, 2^n, 2^n + 1)$, so that the hardware was all implemented using combinational logic circuits. However, there are other possibilities in literature, as for example the implementation based on look-up tables using ROM memory [4].

The RNS operation can be separated into three main stages. The first stage is composed by the forward conversion, where a binary number is converted to a RNS number. In the conversion step, residues are generated and when they are ready to be read, the second stage begins, and it represents the modular sum. In this stage the residues that come from the converter are added in parallel. The last stage is the reverse conversion. The conversion process from residue to binary numbers is a very demanding task. Therefore, the design of an efficient converter from residue to binary is one of the most important tasks in order to increase the use of RNS-based applications and processors [5]. As should be emphasized, while in the work of [3], the converters were implemented based on the architecture suggested by [6], in our work, the converters are implemented according to the equations of [4]. As will be presented, the circuits based on these equations are more efficient.

2. RNS representation

Assuming that: $X = (x1,x2,x3, ...)_{RNS(m1,m2,m3, \ldots)}$, where, $xi = X \bmod mi$.

Each *mi* represents a *modulus* and all of them together represent the *moduli*. In this representation, the term *xi* is obtained by dividing a binary number by its correspondent *modulus* (*mi*). The *moduli* is represented by the pairwise of prime numbers. It guarantees that the dynamic range there will be not duplicated numbers in the RNS representation. The dynamic range is obtained by multiplying all the *modulus* as it is shown below [3][4][7][8].

$M = m_k\text{-}1 \times m_k\text{-}2 \times \ldots m_0$, where M is the dynamic range

It means that to cover all dynamic range of a binary number of 16 bits, for example, M must to be larger than 65535. In the case of working with *moduli* set $(2^n - 1, 2^n, 2^n + 1)$ and 16 bits, the *n* term in the *moduli* set should be 6 bit-width or larger than it. However, in the case where the *moduli* set is larger than 6, the circuit implementation should not to be the smaller one and its performance could not be improved.

In tab. 1 it is shown some radices of operation in RNS and its number of bits that is needed to find the dynamic range [3].

Tab. 1 – Radices and its dynamic range for the RNS operation

Radices	8 bits	16 bits	32 bits
$(2^n-1, 2^n, 2^n+1)$	$(2^3-1, 2^3, 2^3+1)$	$(2^6-1, 2^6, 2^6+1)$	$(2^{11}-1, 2^{11}, 2^{11}+1)$
$(2^{n-1}-1, 2^n-1, 2^n)$	$(2^{4-1}-1, 2^4-1, 2^4)$	$(2^{6-1}-1, 2^6-1, 2^6)$	$(2^{12-1}-1, 2^{12}-1, 2^{12})$
$(2^n-1, 2^n+1, 2^{2n}+1)$	$(2^3-1, 2^3+1, 2^6+1)$	$(2^5-1, 2^5+1, 2^{10}+1)$	$(2^9-1, 2^9+1, 2^{18}+1)$

3. RNS operations

RNS operations starts with a forward conversion, where it is processed a conversion from a conventional number to RNS ones. After the forward conversion the residues are processed in a parallel way. The results of each parallel addition are the results of operation in RNS. In order to obtain the number back to the binary representation, it is needed a reverse conversion. All this process is presented in fig. 1 [4].

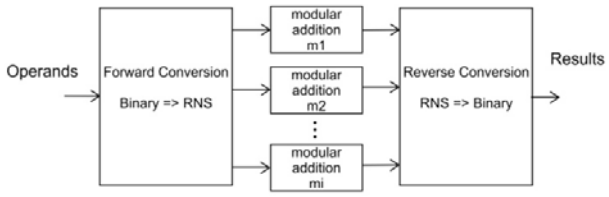


Fig. 1 – RNS process

3.1 Forward conversion

The conversion process is the main bottleneck in the RNS operation and this aspect has limited the use of this representation in some applications from microelectronics and informatics areas. It occurs mainly due to the additional hardware that is needed to implement this stage. In the forward conversion, the binary number is separated in three parts of *n*-bits [3][4], according to the equations 1, 2 and 3 [4].

$$B1 = \sum_{j=2n}^{3n-1} x_j 2^{j-2n} \quad (1)$$

$$B2 = \sum_{j=n}^{2n-1} x_j 2^{j-n} \quad (2)$$

$$B3 = \sum_{j=0}^{n-1} x_j 2^{j} \quad (3)$$

According to [4], the residues *r1* and *r2* can be calculated as follows in equations 4 and 5.

$$r1 = |B1 - B2 + B3|_{2^n+1} \quad (4)$$

$$r2 = |B1 + B2 + B3|_{2^n-1} \quad (5)$$

These equations represent the modular additions, and they must to be corrected when their results get over the *modulus* value. As can be observed in the equations 4 and 5, while in the residue *r2* only positive numbers are considered, the residue *r1* can assume positive and negative values.

3.2 Modular addition

Modular addition is different from a conventional addition, because the result may get over the *modulus* value, and when this occurs, it must to be detected and corrected. Fig. 2 shows a simplified circuit to perform the addition of the *modulus* $2^n - 1$ and $2^n + 1$, where the addition of the *modulus* 2^n (*r2*) represents a simple sum, with the most significant bit being discarded.

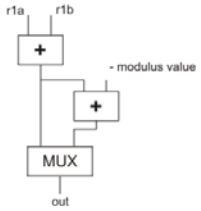


Fig. 2 – Modular Addition

3.3 Reverse conversion

The reverse conversion represents the most complex stage of the RNS process. For the implementation of this stage it could be used the CRT (Chinese Remainder Theorem) [4] or the Mixed-Radix Conversion (MRC) [4]. In this work, we are using the first theorem. All the other methods are only variations of these two methods, whose variations depends on the type of *moduli*-set chosen or certain properties that can be easily adapted to suit the particular approach chosen [4]. From the CRT theorem it is possible to represent two equations to describe all the reverse conversion for the *moduli* $(2^n-1, 2^n, 2^n+1)$. These equations are presented below.

$$|X|_M = \left|\frac{m2*m3}{2}*x1 - m1*m3*x2 + \frac{m1*m2}{2}*x3\right|_M \quad (6)$$

$$|X|_M = \left|\frac{M}{2} + \frac{m2*m3}{2}*x1 - m1*m3*x2 + \frac{m1*m2}{2}*x3\right|_M \quad (7)$$

The equations 6 and 7 are similar the only difference between them is that while the first one is used when the sum of *x1+x3* terms is even, the second one is used when this sum is odd. The implemented circuit based on the equations 6 and 7 is shown in fig. 3.

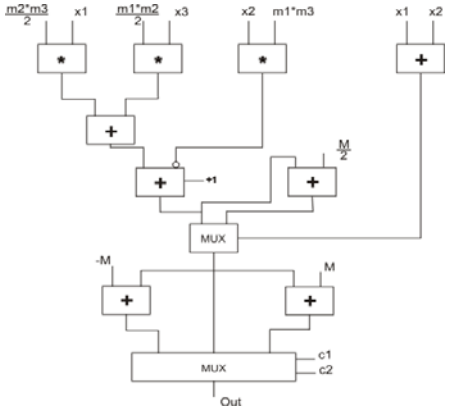


Fig. 3 – Reverse Conversion

In the circuit shown in fig. 3, the terms *x1* and *x3* are multiplied by the *modulus* (*m2*m3*)/2 and (*m1*m2*)/2 respectively. The partial results are subtracted from the multiplication between *x2* and *m1*m3*. In this point the first multiplexer needs to decide what result will be put at the output. It will be decided according to the result of the sum of *x1+x2*. If the sum is even, the value must not to be modified. However, if it is odd, a sum with *M*/2 needs to be done in the circuit. In the second multiplexer it is decided which value goes to the output (if the results need some correction or not). If the result is larger than zero and smaller than the dynamic range (*M*), the results does not need to be corrected and it goes to the output of the circuits. However, if this value is larger than *M*, it must to be subtracted by *M* in order to keep the results with the correct value. If the result is smaller than zero, a sum of *M* must to be done in order to also keep the results with the correct value too.

4. Experimental Results

The developed architectures were described in VHDL and synthesized to TSMC 0.18um standard cells technology, using Leonardo Spectrum from Mentor Graphics tool. Ripple Carry (RCA) and Carry Save (CSA) adders were used for the implementation of the modular addition stage. Tab. 2 shows the delay comparisons between some of the adders circuits present in literature, such as Ripple Carry and Carry Look Ahead, against the RNS adder by using RCA and the most efficient CSA in the modular addition step. We have also compared our implementations against the RNS adder of [3]. All the circuits were implemented for 16 bit-width.

Tab. 2 – Delay values for of 16-bit adder circuits

	CLA	RCA	RNS [3]	Our RNS	
				RCA in the Modular addition	CSA in the modular addition
Delay (ns)	2,01	2,59	12,24	9,87	9,39

As can be observed in tab.2, even using a more efficient adder (CSA) in the modular addition step, our RNS adder presents significant more delay value than the RCA and CLA adders. It occurs due to the large amount of additional hardware that is needed in the conversion steps. In particular, the reverse conversion is the one that present the more complex structure leading to the increase of delay in the RNS operation, as can be seen in tab. 3. As can be also observed in tab. 2, our RNS adders present less delay value than the circuit proposed in [3]. This occurs because we are using the most efficient equations of [4] for the implementation of the converters. This aspect can be observed in tab. 3, where the modular addition and conversion stages are compared against the circuits of [3].

Tab. 3 – Delay values for the stages of the RNS circuits

	RNS [3]		Our RNS			
	Total Conversion	Modular Addition	Forward Conversion	Reverse Conversion	Total Conversion	Modular Addition - CSA
Delay (ns)	9,9	2,34	2,31	5,65	7,96	1,43

The results presented in tab. 3 prove that the reverse conversion is the main bottleneck in the RNS adder circuit. In fact, the large delay value presented by this step is related to the more complex hardware used for the decoding from RNS to binary step, as can be observed in fig. 3. In this fig., it can be observed the large critical path composed by multiplier and adders circuits. One interesting aspect to be observed in tab.3 is that our modular addition presents less delay value than the pure RCA and CSA circuits, as can be compared with tab. 2. In some applications, such as digital filters, where a large amount of adders is used, it can become interesting the use of RNS, since the conversion step will be done only one time. As can also be observed in tab.3, the stages of the RNS adder proposed in this work are more efficient than that one proposed by [3]. This explains the higher performance presented by our RNS adder, when compared against the circuit of [3].

5. Conclusions

In this work we have presented the implementation of RNS adder circuit. Based on the obtained results we could conclude that the use of RNS for adder circuit can be not a good way to improve the performance of this circuit. However, the fact that the modular additions in RNS presented better performance than the others adders, indicates that for some applications that do not need to make the conversions every time, such as digital filters, this representation can presents some benefits. We were also able to improve the performance of the RNS adder, when compared against the circuit proposed in literature. As future work, we intend to use more efficient adders and multipliers than the ones that were used in this work in order to improve the performance of the forward and reverse conversion steps.

6. References

[1] Y-C Kuo, S-H Lin, M-H Sheu, J-Y Wu and P-S Wang. Efficient VLSI Design of a Reverse RNS Converter for New Flexible 4-Moduli Set (2^{p+k}, 2^{p+1}, 2^{p-1}, 2^{2p+1}). Graduate School of Engineering Science and Technology, National Yunlin University of Science & Technology, Touliu, Yunlin, Taiwan, 2009.

[2] A. D. Re, A. Nannarelli, M. Re, Implementation of Digital Filters in Carry-Save Residue Number System. University of Rome Tor Vergata, Italy, 2001.

[3] M. Handel, Circuitos Aritméticos e Representação Numérica por Resíduos. Dissertação de Mestrado, Porto Alegre, 2007.

[4] A. Omondi, B. Premkumar. Residue Number Systems – Theory and Implementation. 1st ed. Singapore, 2007.

[5] A. Hiasat, A. Sweidan, Residue number system to binary converter for the moduli set $(2^{n-1}, 2^n - 1, 2^n + 1)$, 2003.

[6] G. Bi, E. V. Jones, Fast conversion between binary and residue numbers. Electronics Letters, [S.I.], v.24, n.19, p.1195-1197, 1988.

[7] R. Chaves, Processamento de Sinal e Criptografia Baseados em Sistemas de Numeração por Resíduos. Trabalho de Conclusão, Lisboa, 2001.

[8] B. Parhami, Computer Arithmetic – Algorithms and Hardware Desingns. 1º ed. New York, 2000.

Optimal Arrangement of Parallel Prefix Adder(PPA) Trees According to Area and Performance Criteria

Kim Aragon Escobar, Luca Couto Manique Barreto, Renato Perez Ribas
{kaescobar,lcmbarreto,rpribas}@inf.ufrgs.br

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

Abstract

With the advance of technology nowadays, digital circuits are more and more present in the daily life, through the use of microprocessors, embedded systems, etc. These circuits have to provide arithmetic operators of bits (addition, subtraction, multiplication, and so on), and for that adder circuits represents the basic block. There are many implementations of adders, each one with a different purpose and features, focusing in complexity (size or area) or in performance (speed and power dissipation). Parallel Prefix Adder – PPA approach, in particular, can be adapted to focus on these two targets (area and speed), or even present an hybrid solutions. The problem of PPA is to find its optimal arrangement of the propagation (P) and generation (G) tree for carries calculation. This paper proposes an algorithm to find the optimal PPA architecture according to these costs, automating the process to any number of bits through geometric patterns found in the PG network.

1. Introduction

Microprocessors (microcontrollers) and embedded systems are largely present in costumer products, like cell phones, notebooks, medical equipments, automobile commends, and so on. All these circuits require internally arithmetic operators, and the basic block to do that are the digital adders. In the design of electronic circuits there are different possibilities to implement this kind of functional block, depending on the focus of design optimization: performance or area.

Adders can be implemented in different architectures, each of them with particular benefits and features [1][5]. The *ripple carry adder* - RCA is probably the simplest among all known architectures, presenting good performance and area for few bits (something like less than 8 bits). It represents the optimum approach for circuit size. The *carry select adder* – CSelA, in turn, is improved in terms of speed at expense of area. The *carry skip adder* - CSkipA, on the other hand, presents a better compromise between these two parameters. In fact, the most popular approach in terms of performance is *parallel prefix adder* – PPA, being the *carry lookahead adder* - CLA a sub-class of that [1][5].

PPAs (and CLAs) are based on the *propagate* (P) and *generate* (G) principles, being the PG tree the critical part of the adder algorithm and circuit. As discussed in Section 2, this PG tree provides the carries calculation, and a large number of different arrangements can be adopted with direct impact in the circuit area and performance. Some solutions are well-known, like Brent-Kung (B&K) and Kogge-Stone (K&S) approaches [4][5]. However, hybrid solutions that focus not only in minimum circuit area or maximum speed are preferable and not easy to obtain.

This paper presents an algorithm for optimizing the carries calculation for a given design constrains in the circuit size and worst-case paths of signal propagation, considering any number of bits. The work has been validated by comparing the results with the extreme cases of optimization, i.e. the B&K and K&S approaches.

In Section 2, it is briefly presented the algorithm of PPA for better understanding of the paper proposition. In Section 3, the proposed algorithm is described and discussed in its peculiarities. In Section 4, some experimental results are provided for work validation, and in Section 5 the conclusions are outlined.

2. Parallel Prefix Adder

Parallel prefix adder (PPA) algorithm is based in the principle of *generate* (G) and *propagate* (P) signals. The principle of P and G is predict if in an sum of 2 vectors, A and B, will occur an Cout. P signal predicts if the sum between two bits a_i and b_i propagates the carry arriving in this index (cin_i) to the next one. It can be done by checking if a_i or b_i presents the logic value '1' (OR or Exclusive-OR operations). For exemple, consider $cin_i = 1$, $a_i = 1$ and $b_i = 0$, the sum of $cin_i + a_i + b_i$ generates a carry signal ($cout_i = 1$) that corresponds, in fact, to the carry in the next index ($cout_{i+1} = 1$). G signal, in turn, predicts if the sum between two bits a_i and b_i generates a carry signal ($cout_i = 1$), without considering the the carry arriving in this index (cin_i). It is possible by just verifying if both a_i and b_i presents the logic value '1' (AND operation).

PPA uses initially the *propagate* and the *generate* signals individually for each bit index. The circuit that calculates these two signals or operations, P_i and G_i, together are called here as *cell operator*, and this operator

is considered as the unit of area of the tree of PPA. The algorithm of PPA present 3 steps, taking into account the two input vectors A and B:

1 – Calculate the G_i and P_i to each bit with the cell operator.

2 – Calculate the *group generate* signals related to the first index ($G_{i,0}$) that corresponds to the carry os each index (cin_i) used for the sum operation. To calculate the $G_{i,0}$ signals, the algorithm use the cell operator of each bit to construct the cell operator of an array of bits (group). If it is taken the P_i and G_i of bit 'n+1' and bit 'n', the evaluation of these two adjacent indexes represents the group signals $P_{n+1,n}$ and $G_{n+1,n}$. This grouping operation is done until all *group generates* $G_{i,0}$.are obtained. The equations to calculate the group signals are:

- *group generate* $G_{n+1,n} = G_{n+1} + P_{n+1} \bullet G_n$;
- *group propagate* $P_{n+1,n} = P_{n+1} \bullet P_n$.

being,

- *individual generate* $G_n = a_n \bullet b_n$
- *individual propagate* $P_n = a_n \oplus b_n$.

3 – Calculate the sums using the equation $S_n = P_n \oplus G_{(n-1),0}$.

Some well-known solutions, as mentioned in Introduction, are the Brent-Kung (B&K) and Kogge-Stone (K&S) approaches. There are also two others, Ladner-Fischer (L&F) and Han-Carlson (H&C). B&K solution optimizes the circuit for the lowest area to level $2 \bullet \log_2 n - 2$. K&S version, on the other hand, illustrated in Fig. 1a, optimizes the tree arrangement to the shortest path (operators in chain or series configuration) and the lowest number of connections (fanout), but the area tends to be large. L&F solution, in turn, depicted in Fig. 1b, optimize the circuit to the lowest area and to the shortest path, without taking into account the number of connections. H&C approach is a hybrid solution that tries to combine the B&K and K&S structures.

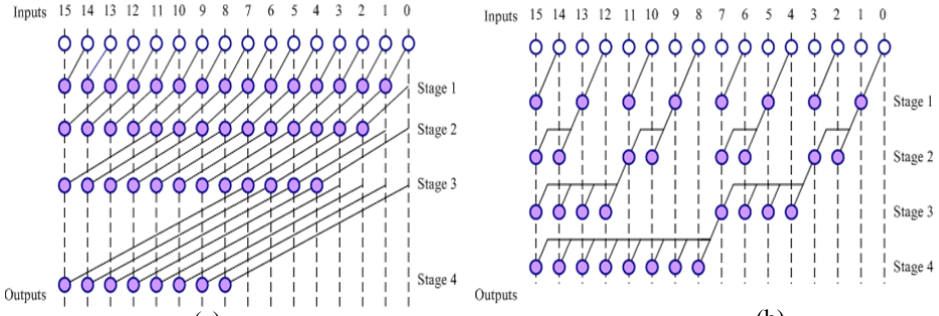


Fig. 1 – PPA structures: (a) Kogge-Stone and (b) Ladner-Fischer[3].

3. Proposed Algorithm

Having the preview knowledge about PPA, we begin with the basic definitions of the algorithm. In an adder, a group is a set of bits - obligatorily adjacent - represented as A[i,j], such that $0 \le i \le j \le n$, this group must have all the cells (j,i);(j-1,i);...;(i+1,1). Upon the groups are defined unary operations: *depth*(A), *area*(A)(units of cells) and *length*(A)(of bits), and a binary operation: *concatenate*(A,B). An interesting property we need to know is that the minimum depth of an adder with *n* bits is log2 *n*, while the maximum depth is *n*-1.

Given an adder with *n*-bits, and the groups A[i,j] and B[j+1,k], such that $0 \le i \le j \le k \le n$, the operation concatenate(A,B) is to connect the cell (i,j) with each (x,j+1), such that x=j+2,...k, and with j+1, returning the group AB[i,k]. The fig. 2 exemplify the explanation above. Tab. 1 shows us some interesting proprieties of concatenate:

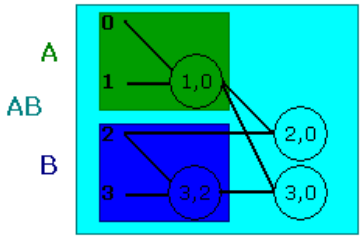


Fig. 2 – Concatenate

Tab.1 – Proprieties of concatenated group

$$depth(AB) = MAX(depth(A), depth(B)) + 1$$

$$area(AB) = area(A) + area(B) + length(B)$$

$$length(AB) = length(A) + length(B)$$

The algorithm has bottom-up design, beginning with smaller adders and making, with these, more complex ones. This happens because each adder's group is an adder by itself.

We start calculating the area of an adder with minimum length (1) and iterate increasing the length until it reaches *n* (the desired length). For each length, the algorithm iterates from its minimum to its maximum depth, calculating the area of the corresponding adder.

To calculate the area, the algorithm needs do know the quantity of groups an adder has. Basically, it's made as follows:

- At its minimum depth an adder A has 2 groups (obviously with *length*(A/2) bits each), as illustrated in fig. 3.
- We reach each subsequent depth level by dividing a group in two. This way, we increase in one the number of group and, because of *depth*(AB) property, increase in one the depth level too, as illustrated in fig. 3.
- Nevertheless, when we divide the first group of an adder, by having a smaller group, we lose (while we gain) a depth level. To solve this problem the algorithm automatically "ignores" this division and "jumps" to next, as illustrated in fig. 3.

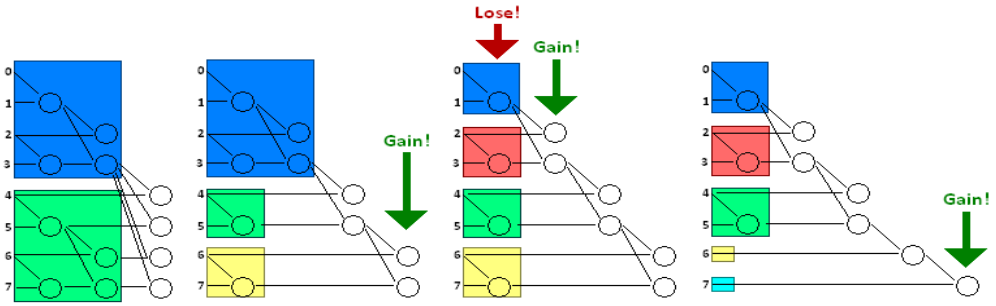


Fig. 3 – groups splits

- An interesting behavior that we have from that phenomena above are that, except for the initial case (2 groups), we'll have always two lengths of group, that we called *high groups* and *low groups*.
- The algorithm calculates the length of high and low groups and the quantity of each one in the corresponding adder. Then, calculating the necessary heigth, it accesses the previous results (the smaller adders) to get the area of each group and uses the *area*(AB) property to reach the total area.

4. Experimental Results

The implementation of the algorithm was made with C++ programming language. In Fig. 4, it is shown the results to different inputs and the comparison to the K&S and L&F approaches. With 16 bits K&S has 4 levels (for cell operators in the longest path) and 49 units of cells, while L&F version has 4 levels and 32 units of cells. In the results obtained can be seen that for level 4 the area is 32, equal to L&F solution, and less than K&S, that is, the result of the algorithm is better than K&S approach if only in the area and longest path are considered, ignoring the number of connections at each internal node.

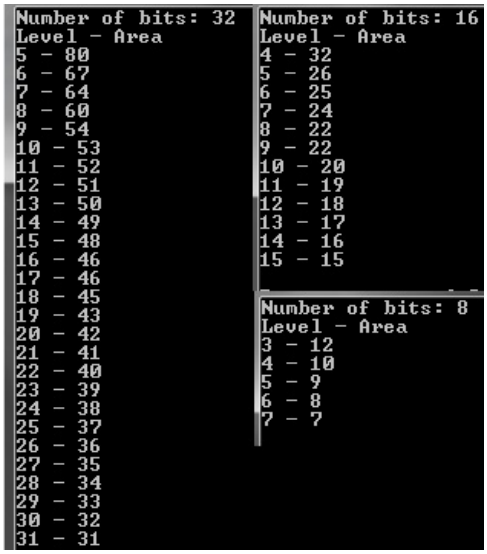


Fig. 4 – Results of the program

5. Conclusion and Future works

This algorithm is only the "core" of all idea and this only generate the outputs which will be used to do some statistic to encounter the circuit corresponding with the weight that the user put to area and performance. For future works will be implemented the statistic function and a parser to translate the information obtained by the program to spice, VHDL and verilog language.

References

[1] I. Koren, "Computer Arithmetic Algorithms", pages 93-132, A. K. Paters, 2ª edition, 2002.

[2] P. M. Kogge, H. S. Stone, "A parallel Algorithm for the Efficient Solution of a General Class of recurrence Equations" pages 786 - 793 , IEEE Transaction on Computer, Vol. C-22, No.8, 1973.

[3] "Hardware algorithms for arithmetic modules" ;http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html#ppa_wlc

Implementation of a Floating Point Unit in the Technology X-FAB 0.35

Raphael Neves, Jeferson Marques, Iuri Castro, Edson Schlosser, Sidinei Ghissoni, Alessandro Girardi
Raphael.cn.gama@gmail.com
{sidinei.ghissoni, alessandro.girardi}@unipampa.edu.br

Federal University of Pampa – UNIPAMPA
Campus Alegrete
Av. Tiaraju, 810 – Alegrete – RS – Brasil

Abstract

This paper presents an architecture and organization of a floating point (FPU) that follows the IEEE 754 standard for the representation of binary numbers. The development of this architecture was done with emphasis on applications requiring low power consumption and area constraints, such as embedded systems. The operations performed by FPU are: addition, subtraction, multiplication and division, with emphasis on the reusability of blocks that perform basic functions. The FPU has been described in language hardware SystemVerilog and logic synthesis tool was perform at the Design Compile.

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]. The representation of a number in floating-point precision 32-bit, according to IEEE 754, is composed of 1 bit for sign, 8 bits for exponent and 23 bits for `fraction, as shown in Figure 1. We call mantissa M the binary number composed by an implicit 1 and the fraction:

$$M = 1.f$$

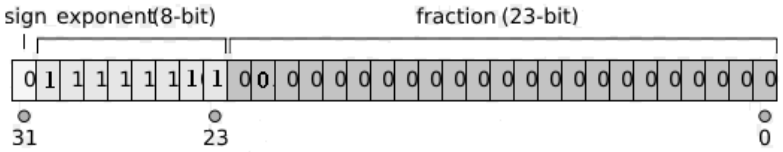


Fig. 1-IEEE Standard for floating point numbers in single precision.

This paper presents the architecture and organization of a floating point unit single-precision 32-bit. The FPU has developed applications in embedded system [7], with low power dissipation and area restrictions, including the 4 basic arithmetic operations: addition, subtraction, multiplication and division. The rounding mode was adopted as specified by IEEE 754. The unit was described in System Verilog and was performed logic synthesis using Design Compile which is a tool of Synopsys Galaxy Package. This paper is organized as follows: In Section II presents the operations point floating, Section III describes the proposed data path, section IV presents the results of the implementation, and, finally section V conclusion on paper.

2. Floating Point Operations

This section is present the operations supported by the floating point unit, as well as their flowcharts and algorithms used. At the beginning of each flow are conducted the analysis of cases of exceptions such as overflow / underflow, division by zero and Not-a-Number (NaN), as mentioned in the IEEE 754. The

operations of addition and subtraction are complicated by the need to align the exponents [3]. According to the block diagram shown in Figure 2, this alignment is required only if the exponents are different. If this statement is true, the smallest exponent is incremented by the difference calculated by subtracting the exponents. The outcome as the subtraction of this signal will determine the alignment of the exponents and hence to the alignment of the mantissa. Thus, according to Figure 2(a), is made first verification of the exponents. The alignment of the exponents and mantissas is needed when the exponents are different.
In the block diagram of the multiplication, shown in Figure (3), the sign of the operation is defined in parallel with the sum of the exponents. In this operation, you must subtract the bias (127), since for the sum of the exponents it is added twice. For the multiplication of the mantissas, we used a multiplier sequence of integers.

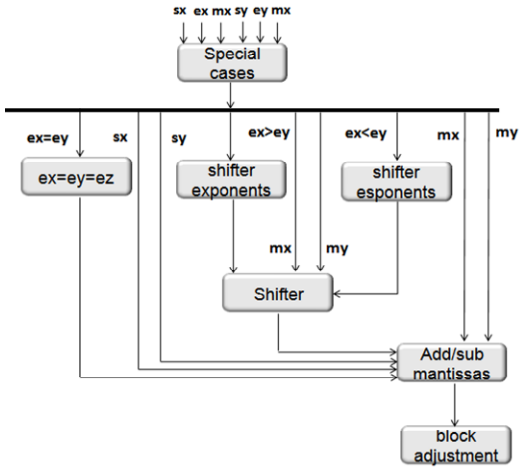


Fig. 2(a) – The flowchart of addition and subtraction.

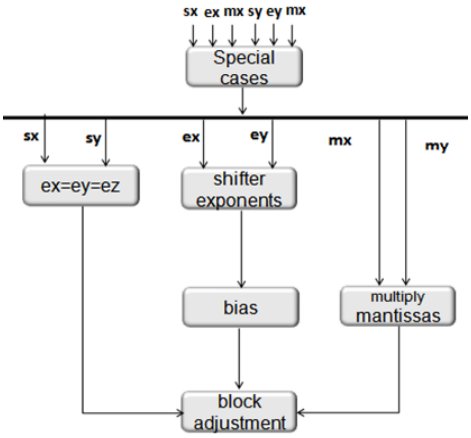


Fig. 2(b) – The flowchart of multiplication.

The operation of floating-point division, follows the same data stream by multiplying, with the difference that the mantissas are divided and the exponents subtracted. In the division of the mantissas used a sequential divider for real numbers, because even the division of two integers can result in a real number. In Figure 3 is shown the restoration algorithm [3] used to divide the mantissas. This algorithm is characterized by the recursive procedure represented by Eq (1):

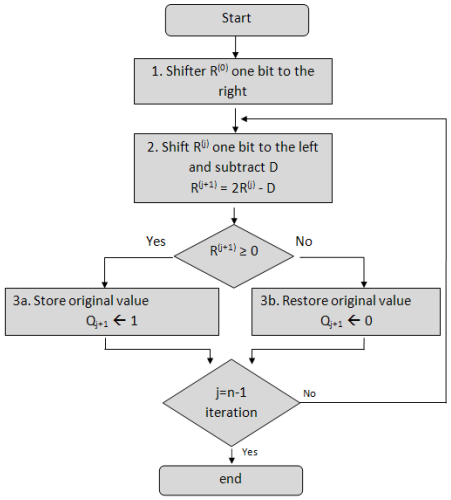


Figure 3: Algorithm for restoration division of real numbers.

$$R^{(j+1)}=2\bullet R^{(j)}-q_{j+1}\bullet D \qquad (1)$$

where j = 0,1, ... , N-1 is the index of recursion, and $R^{(j)}$ is the rest part of the j-th iteration. The initial partial remainder $R^{(0)}$ is equivalent to the dividend and $R^{(n)}$ is the final rest, and $q_{j+1} \in (0,1)$, and $0 \leq R^{(j+1)} \leq D$. In the

first step of this algorithm, shifts $R^{(0)}$ one bit to the right, this shift is necessary to ensure that the dividend will always be less than the divisor.

Thus, shifts $R^{(j)}$ a bit to the left and subtract D. Thus, it is observed the magnitude of the signal $R^{(j+1)}$, ie, the possibility of restoration. In Eq (3) have the multiplying factor q_{j+1} D. This factor is an estimate of the signal $R_{(j+1)}$. That is, if $R^{(j+1)} \geq 0$, $q_{j+1} = 1$. Therefore, $R^{(j+1)} = 2R^{(j)} - 1 \bullet D$. On the other hand, if $R^{(j+1)} < 0$, $q_{j+1} = 0$, then $R^{(j+1)} = 2R^{(j)} - 0 \bullet D$, which would be the same as a restoration. As the machine does not make a prediction signal of the operation, it needs to perform a number of deletions and restore them if necessary. Note that, in the operations of multiplication and division, both the divider and the multiplier are based on sequential circuits [5]. The choice of sequential architectures due to the fact that they offer better performance in area and power consumption for the combinational architectures [6]. However, we have a decrease in speed. As the design requirements of the FPU is to minimize area and power, the reuse of hardware becomes an important and possible with the choice of these architectures, for both operations have the data path similar, differing only in control.

Upon completion of the data flow of each floating point operation, the result is sent to the stage of adjustment. This stage is responsible for the detection of overflow / underflow in an intermediate stage of the operation, besides the normalization and rounding.

3. PROPOSED ARCHITECTURE

This section presents the data path and control the FPU. The data path of FPU is shown in Figure 4. This architecture should be able to withstand the flow of operations presented in Section III. We used the following strategy: the functional units (adder / Subtractor, shifter, registers) are shared by different operations. For example, the multiplication operation are due to add the exponents, and the addition operation must be adding the mantissas. Thus the same hardware should be able to perform the sum of both exponents and mantissas. The great advantage of this technique is the economy of the functional units, resulting in a smaller silicon area and thus minimizes the costs for synthesis physical the project.

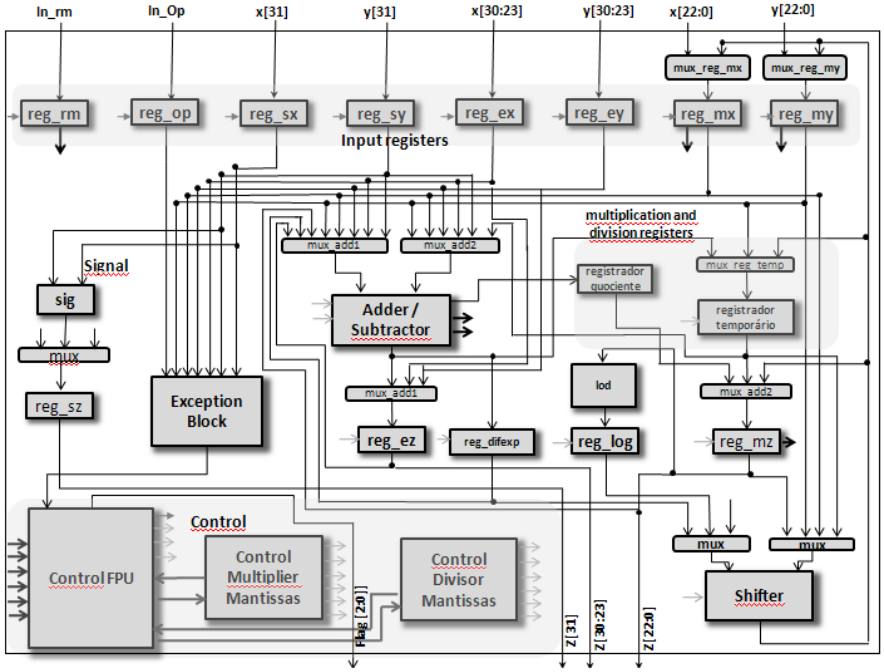


Figure 4: Part of the operational architecture of FPU in the IEEE 754 single precision.

The data path of the blocks are highlighted in Figure 4. Block Adder / Subtractor deserves this architecture, it is used in all operations. The FPU operations begin with the recording of input registers, which are stored the rounding mode, operation and values of the operands according to the IEEE 754. The exception block checker analyzes the existence of a special case. Confirming the exception, a signal is sent to the control, which returns a three-bit flag indicating the special case occurred. If no special case is reported, the next step is to perform the operation desired. For the addition operation, as the flow presented in section IV, it is necessary to determine the smallest exponent, shifting the mantissa to the right, add the exponents, and finally, add or subtract the mantissa. The difference of the exponents is performed in block adder / Subtractor through subtraction. From the smallest exponent is determined which mantissa should be shifted. In this project, we used 7 Carry Look Ahead adder (CLA) of 4 bits, as the larger fleet to be added has 27 bits [7]. The exponents and mantissas are represented in sign magnitude and CLA performs operations in addition to 2. You must then check the sign of the result of the operation of the CLA and reverse it if necessary. The control used in this

project was based on finite state machine. The control is responsible for sending signals to the operational side in order to control the functional units of the FPU. The control of the operative part was divided into three parts: the FPU control, control of multiplication and division control.

4. Results

7.2. We used the tool for Design Compile perform the logic synthesis architecture. The technology used was X-FAB 0.35. The Table 1 is presented the results of area and frequency. The functional requirements of frequency of operation of the circuit is 50MHz.

Tab.1 – Results area for the FPU architecture.

Circuito	Combinational área(µm):	Noncombinational area(µm:	Net Interconnect area (µm):	Total cell area(µm):	Total area(µm:
FPU	1514736.815285	53056.00062	417839.46217	1567792.8159	1985632.27

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. 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 will perform the physical synthesis 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 0.18um 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.

Floating Point Unit Implementation for a Reconfigurable Architecture

[1]Bruno Heckttheuer, [1]Mateus Grellert, [1]Júlio Mattos, [2]Antonio Beck, [3]Mateus Rutzig, [3]Luigi Carro

{brunob.ifm,mateusg.ifm,julius}@ufpel.edu.br, caco@inf.ufsm.br, {mbrutizg, carro}@inf.ufrgs.br

[1]Federal University of Pelotas
[2]Federal University of Santa Maria
[3]Federal University of Rio Grande do Sul

Abstract

With the complexity of embedded systems growing day by day, there are several electronic devices with different applications in a single device. To cope with this heterogeneous behavior of these applications it is necessary embedded architectures providing high performance. Reconfigurable architectures present a solution based on high performance maintaining low power consumption. Thus, this paper presents the implementation of a combinational Floating Point Unit (FPU) for a reconfigurable architecture. The designed architecture supports four basic arithmetic operations (addition, subtraction, multiplication and division), and is compatible with the IEEE 754 standard. In this paper, we present the Floating Point Unit target to a reconfigurable architecture and the results in terms of area and delay.

1. Introduction

Nowadays, the majority of electronic devices available on the market have computational resources embedded in their project. Researches show that the average annual growth of embedded systems in many sectors of industry is always above 5% [1]. Since the growth of the embedded market, the complexity of these systems has been increasing due to aggregation of different functionalities in a single device, e.g., cell phones. These devices run applications with heterogeneous behaviors, i.e., distinct hardware resources are needed in order to support the efficient execution of these applications. The use of general purpose processors to handle this problem does not guarantee an efficient implementation and results in terms of energy consumption.

Reconfigurable architectures [2] may be a good solution to this problem. These platforms have the ability to adapt (reconfigure) according to the type of application. The major advantage of this type of architecture is the flexibility, i.e., efficiency in execution does not depend on the behavior of the application being executed. These architectures are located between the von Neumann and Dataflow models.

The target architecture used in this work consists in a Reconfigurable Functional Unit (reconfigurable array), a unit responsible for reconfiguration and a general-purpose processor. The basic idea of this approach is to find pieces of code that can be executed more efficiently in the Reconfigurable Functional Unit [3] during the program's execution. This unit is implemented in a combinational logic. It is composed by functional units that perform logic and integer arithmetic operations, multipliers and memory access units.

Multimedia and communication algorithms applied in the field of embedded systems make an intensive use of floating point arithmetic. Due to the complexity and cost of implementations of floating point units in hardware, algorithms often use emulation in software or conversion (manually or automatically) of floating point to fixed point operations.

The target architecture of this work did not use floating point operations in hardware, which produced no suitable solutions in terms of performance and introduced inaccuracies in software implementations (floating point emulation software is not able to provide the required accuracy).

Therefore, this work presents the Floating Point Unit (PFU) development target to Reconfigurable Functional Unit. Thus, the reconfigurable architecture will provide more efficiency regarding floating point operations, making their use more feasible for embedded systems. The designed FPU implements the addition, subtraction, multiplication and division operations, based on combinatorial logic and the IEEE 754 standard [4]. The architectures modules were described using VHDL and synthesized in FPGA.

The paper is organized as follows: Section 2 presents the related work. Section 3 introduces the reconfigurable system where the FPU is attached. Section 4 shows the implemented Floating Point Modules. Section 5 presents the obtained results in terms of FPGA resources and frequency and, finally, Section 6 presents the conclusions and future work.

2. Related Works

There are several works related to floating-point architectures in the literature. These works present technical enhancements of floating point operations in terms of performance, area and power. Due to the large area occupied by floating-point units, many studies proposed the area optimization. Chong [5] proposes a multi-mode unit that can be reconfigured in several operations, providing the economy in area. Several other studies propose optimizations in FPGA implementation [6][7][8].

Other works present the floating-point units design and implementation for ASIPs. Karuri [9] shows an architecture target to embedded applications. Chong [10] also shows the generation of a FP unit for ASIPs.

Implementations of asynchronous floating-point units (in a fully combinatorial) are not common, but some works implement the FPU in this way, usually for dividers. In [11] and [12] are presented an asynchronous IEEE 754 divider design. Finally, Noche [13] shows a floating point unit fully implemented in a combinational level of transistors.

3. The Reconfigurable System

The reconfigurable system used as case study in this work is composed by a Reconfigurable Unit (reconfigurable array), a unit responsible for reconfiguration and a general-purpose processor [3]. The reconfigurable array is composed by functional units, multipliers, memory access units, multiplexers and demultiplexers.

The execution is dataflow using combinational logic. Thus, this architecture provides a higher consumption of area, but allows a simple control unit and energy savings. The binary translation engine is a complex hardware responsible not only for detecting binary code parts that can run on the array, but also for generating the configuration bits to configure the array execution flow. These reconfiguration bits are stored in a memory called reconfiguration cache (index by PC – Program Counter).

This reconfigurable system is coupled to a MIPS processor. Basically, if there is a section of code that can run on reconfigurable array, it is identified by the binary translator, and in the first time the code is executed in the processor, the configuration bits are generated and stored in the reconfiguration cache. On other times this piece of code is detected analyzing the program counter (PC) and when it is detected the execution flow changes, and it is executed by the array instead of by the processor.

Figure 1 shows the reconfigurable system and a pipeline of five stages representing the processor, where blocks highlighted in black represents the execution flow when the reconfigurable array is used. The memory can be accessed by the array, because the main memory is shared by the array and the processor.

It is important to note that the array implementation (using combinational logic) implies a large consumption of area and power. However, the instruction execution in dataflow is faster and the energy consumption is reduced significantly.

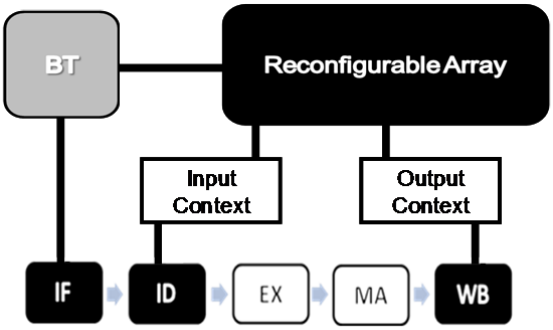


Fig. 1 – Reconfigurable System coupled to a five stage MIPS processor.

4. Floating Point Architectures Implementation

Four architectures are developed: addition, subtraction, multiplication and division. These architectures are single precision, e.g. IEEE 754 32-bit representation [4]. This standard defines how the floating point numbers are stored in memory, the algorithms of rounding, exception handling and so on. The four modules were developed in a combinational way, using the standard algorithms and the format specified by IEEE 754. The rounding technique chosen is called truncation. This technique consists of simply removing the values that cause overflow.

A. Addition / Subtraction Module

The algorithm implementation of addition / subtraction in sequential logic is simple. However, its hardware implementation in combinational way may be a challenge. The steps of the algorithm are (considering two data operands, A and B):

1. Make equal the exponents of A and B (the common exponent must be the highest value);
2. Normalize the mantissa of the operand that the exponent was changed;

3. Add / subtract the normalized mantissas;
4. Check overflow / underflow;
5. Normalize the result (if necessary).

The Step 1 was done by a subtraction between the exponents, where the result is the number of shifts required in the mantissa so that both assume the same exponent. The step 2 is done by right shifts of the mantissa using the result of the subtraction as selector. The steps 3 and 4 are arithmetic operations and checking the overflow and underflow. The result normalization is similar to step 2, but the selection is done by the number of left zeros present in the mantissa result. Figure 2 shows the module addition / subtraction. The circuit is simple, however the normalization circuit is complex.

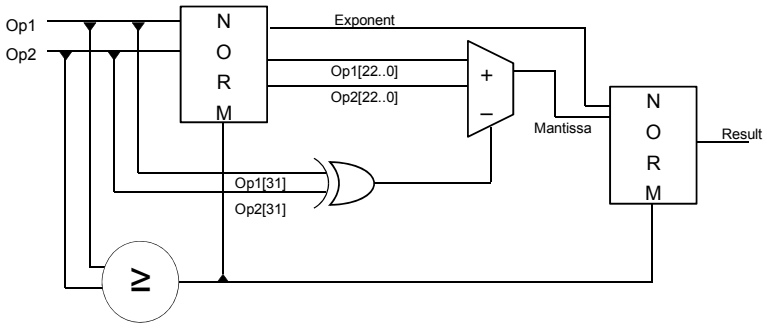


Fig. 2 - Addition / Subtraction Module.

B. Multiplication Module

The multiplication algorithm has the following steps:

1. Add the exponents of the operands;
2. Multiply the mantissas;
3. Check overflow / underflow;
4. Normalize the result if necessary.

The implementation of the four steps are very similar to the steps of module addition / subtraction. The multiplication module diagram can be seen in Figure 3.

C. Division Module

The division module was simplified reusing the same architecture of multiplication module. The equation 1 shows the architecture implements the division using multiplication. In other words, divide X by Y is equivalent to multiply X by the inverse of Y.

$$x/y = x * (1/y) \quad (1)$$

The values for 1/y are stored in a ROM memory. The ROM size is from 1Kb to 32MB providing different solutions in terms of area and accuracy. Large memories provide better degree of accuracy, however it is necessary more area. Despite occupying a large area, this procedure introduces a low complexity architecture. Figure 3 shows the division module. Besides this, another algorithm (based on subtractions and shifts) was tested. This is an iterative algorithm making it impossible to implement in combinational way.

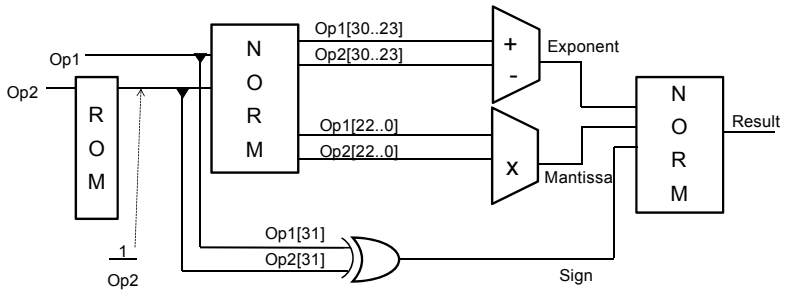


Fig. 3 - Divison Module.

5. Results

This section presents the results of synthesis for the developed architectures. The modules were implemented using the VHDL language and synthesized using Xilinx ISE 10.1 tools and target to device XC2VP30 FPGA family Virtex-II Pro. The validation was performed by ModelSim tool from Mentor Graphics comparing the outputs with reference software implemented in C.

The Table 1 shows the results in terms of area (FPGA resources) and frequency for the FPGA. The implementation is fully combinational, however for the synthesis, registers are used in the input and output of the circuits. The multiplication module has better results in area. This happens because the result normalization is simpler when compared to the other modules normalization. The worst results in terms of area and performance is also expected: the division module. These results are explained by area and access time spent with the ROM (a memory containing 1024 possible values for 1 / y are used).

Tab. 1 - Synthesis Results.

Architectures	ALUT	Slices	Mult. 18x18	Frequency
Sum/Subtraction	892	498	-	83,8 Mhz
Multiplication	119	64	4	78,1 Mhz
Division	1409	741	4	53,1 Mhz

6. Conclusion and Future Works

This paper presented the implementation of a Floating Point Unit based on combinational logic. The algorithms for floating point operations area studied and after four architectures for different operations are implemented in VHDL and prototyped in FPGA.

The Floating Point Unit developed is divided into three modules: addition / subtraction, multiplication and division. Based on the asynchronous floating point unit, the binary translator should be changed making the floating point operations compatible with the array. Therefore, the reconfigurable array will be able to run floating point applications natively.

7. References

[1] S. Vassiliadis. The Hipeac Embedded Systems. Apresentação oral no Hipeac Compilation Architecture, May 2006.

[2] Compton, K. and Hauck, S. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.* 34, 2 (Jun. 2002), 171-210.

[3] Beck, A. C., Rutzig, M. B., Gaydadjiev, G., and Carro, L. Transparent reconfigurable acceleration for heterogeneous embedded applications. In Proceedings of DATE 2008. ACM, New York, NY, 1208-1213.

[4] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985. IEEE, New York, 1985.

[5] Chong, Y. and Parameswaran, S. 2009. Flexible multi-mode embedded floating-point unit for field programmable gate arrays. In *Proceeding of the ACM/SIGDA international Symposium on Field Programmable Gate* Array*s* (Monterey, California, USA, February 22 - 24, 2009). FPGA '09. ACM, New York, NY, 171-180.

[6] Beauchamp, M. J., Hauck, S., Underwood, K. D., and Hemmert, K. S. 2006. Embedded floating-point units in FPGAs. In Proceedings of the 2006 ACM/SIGDA 14th international Symposium on Field Programmable Gate Arrays (Monterey, California, USA, February 22 - 24, 2006). FPGA '06. ACM, New York, NY, 12-20.

[7] deLorimier, M. and DeHon, A. 2005. Floating-point sparse matrix-vector multiply for FPGAs. In Proceedings of the 2005 ACM/SIGDA 13th international Symposium on Field-Programmable Gate Arrays (Monterey, California, USA, February 20 - 22, 2005). FPGA '05. ACM, New York, NY, 75-85

[8] Gaffar, A. A., Luk, W., Cheung, P. Y., and Shirazi, N. 2002. Customising Floating-Point Designs. In Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (September 22 - 24, 2002). FCCM. IEEE Computer Society, Washington, DC, 315.

[9] K. Karuri, R. Leupers, G. Ascheid, H. Meyr, M. Kedia, "Design and Implementation of a Modular and Portable IEEE 754 Compliant Floating-Point Unit," date, vol. 2, pp.43, Proceedings of the Design Automation & Test in Europe Conference Vol. 2, 2006

[10] Chong, Y. J. and Parameswaran, S. 2007. Automatic application specific floating-point unit generation. In Proceedings of the Conference on Design, Automation and Test in Europe (Nice, France, April 16 - 20, 2007). Design, Automation, and Test in Europe. EDA Consortium, San Jose, CA, 461-466.

[11] M. Hiromoto, H. Ochi, Y. Nakamura. An Asynchronous IEEE754-standard Single-precision Floating-point Divider for FPGA. IPSJ Transactions on System LSI Design Metodology, vol.2, p.103-113, Feb. 2009.

[12] J. Kolouch. Combinational Divider in FPGA. In Proceedings of 17th International Conference Radioelektronika, p.1-4, 2007.

[13] J. Noche and J. Araneta. An Asynchronous IEEE Floating-Point Arithmetic Unit, Science Diliman, vol. 19, n.2, p.12-22, July-December 2007.

Devices and Analog Design

1V Self-Biased Current Sources with 10nW of Maximum Power Consumption

[1]Roddy A. Romero, [1]Henrique M. Hayasaka,
roddy.romero@ieee.org, hmhayasaka@gmail.com

[1]Universidade Federal de Santa Catarina

Abstract

This work presents the design of two ultra-low-power self-biased current sources. We propose the use of a very simple topology along with a design methodology based on the concept of the inversion level in order to bias any NMOS or PMOS transistor without dependence on the variation of technologic parameters during fabrication process. An efficient design methodology has resulted in cells with area around 0.0147 mm^2 in the AMS 0.35um CMOS technology and power consumption around 10nW for a 1V supply. Simulated results validate the design and show that the current source can operate at supply voltages down to 1V with an acceptable Power Supply Regulation.

1. Introduction

CMOS Analog design based on the concept of inversion level have been shown to provide a robust alternative for high performance in very-low power and low-voltage circuits. In addition, it is crucial to correctly set the operating point of a MOS transistor in order to precisely control its small-signal characteristics like its transconductance. This fact can be achieved if a bias current dependent on technological parameters, namely specific current, is available and, thus, any transistor can be operated at a given inversion level using trivial scaling rules. This work presents the design of a Self-Biased Current Source (SBCS) which was developed in [1]. This circuit can operate at low supply voltage and provides an output reference current proportional to the specific current of a MOSFET.

This document is organized as follows: In section 2 the principles of working of each block that compound the SBCS is briefly explained. The choices employed in the design are exposed in Section 3. The simulations results of the extracted view of the NMOS and PMOS versions are presented in Section 4. Finally, in Section 5, conclusions and comments are presented.

2. Circuit description

The core of the SBCS is the Self-Cascode MOSFET (SCM) structure shown in fig. 1.

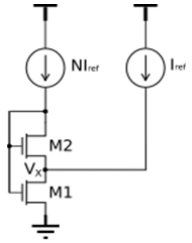


Fig.1 - Self-Cascode MOSFET

This structure can be used as a specific current generator or as a PTAT voltage reference. This behavior can be understood from the following equation and its graphical representation is shown in fig. 2.

$$V_X = \varphi_t \cdot \left[\sqrt{1+M.i_{f2}} - \sqrt{1+i_{f2}} + \ln\left(\frac{\sqrt{1+M.i_{f2}}-1}{\sqrt{1+i_{f2}}-1} \right) \right] \quad (1)$$

Where M is constant variable expressed as:

$$M = \frac{if_1}{if_2} = \left[1 + \frac{S_2}{S_1} \cdot \left(1 + \frac{1}{N} \right) \right] \quad (2)$$

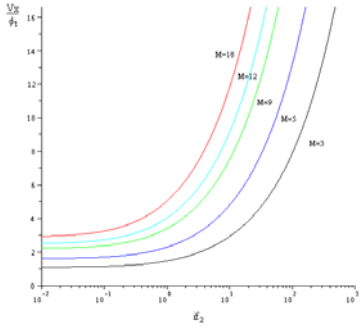


Fig.2 – V-I plot of the SCM

The transistor M1 works in the triode region and M2 in the saturation region. Expressions from the ACM model [2] were used to obtain this equation. It is clear that if the inversion level of M2 is fixed and is independent on the temperature, a PTAT voltage Vx will be generated. On the other hand, if a PTAT voltage Vx is fixed, a certain level of current will be generated. This last behavior is the main principle of operation of the current source because this current will be proportional to the specific current because and the inversion level of any transistors is well defined, as can be seen from the following relation:

$$if_i = \frac{I_i}{S_i I_{SH}} \tag{3}$$

As can be seen from fig. 2, if the SCM is designed to work as a current generator it is necessary to bias the transistors out of the weak inversion because in that region a small variation of the fixed voltage Vx could cause a large variation of current. This can be thought in the opposite way for the behavior of the SCM as a voltage generator.

To build the SBCS, two SCM working in the two different modes can be used with a block that joins them. This block is presented in fig. 3 and it is called the voltage follower current mirror (VFCM) because it can copy a certain voltage V_B to another node Vx and also maintain a fixed relation of currents. For this to be done, M8 and M9 must work in weak inversion [1].

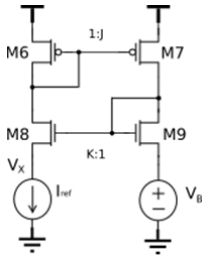


Fig.3 - Voltage-following current mirror

The final circuit of the SBCS is presented in fig. 4.

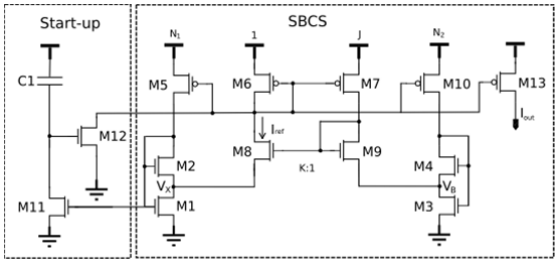


Fig.4 – Self-Biased Current Source

A start-up circuit is important for helping the SBCS achieve a stable output current in a short time.

3. Design procedure

The inversion level of transistor M2 is chosen to be in the moderate inversion in order to the output reference current be less sensitivity to Vx variations. The parameter N is chosen considering area and power trades-off. The current level is defined by taking into account the maximum limit of power consumption and also trying to keep it not too low because the values could be comparable to the leakage currents.

The transistor M1 inversion level is restricted by the sensitivity, low supply voltage and area issues. The sensitivity with respect to Vx is given by [1]:

$$\frac{\delta i_{ref}}{i_{ref}} = 2.\frac{\delta V_x}{\varphi_t}.\left(\sqrt{1+i_{f1}} - \sqrt{1+\frac{i_{f1}}{1+M}}\right)^{-1} \quad (4)$$

Its graphical representation for M=5 is shown in fig. 5:

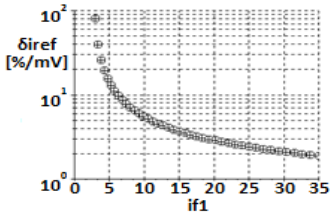


Fig.5 – Current sensitivity with respect to Vx with M = 5.

It can be noted that for if1 higher than 10 the sensitivity of the output reference current with respect to Vx is lower than 6%/mV. The minimum supply voltage is defined by the branch containing M1, M2 and M5, since Vx is lower than 100mV, and depends mainly on the M1 gate voltage which is given by:

$$V_{G1} = n.\varphi_t.\left(\sqrt{1+i_{f1}} - 2 + \ln\left(\sqrt{1+i_{f1}} - 1\right)\right) + V_{th0} \quad (5)$$

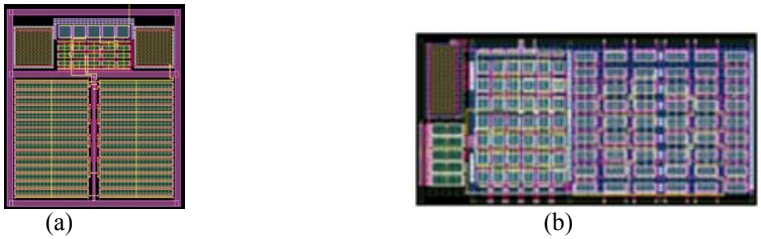
Thus, we have a range from 10 to 115.64 for if1 considering 1V as minimum supply voltage and a Vds_{sat} of 100mV for the PMOS current mirror transistors assuming they are in weak inversion. To choose if1 it must be consider the occupied area because, for low current level, as if1 increases the M1 channel length also increases.

To design the other SCM we sized transistors M3 and M4 to provide the Vx voltage and work at W.I. For a certain Vx, if4 and M are taken using equation (1) and fig. 2. M3 is sized using equation (2). All other transistors were sized to work at weak inversion due to the low supply voltage requirements.

For the start-up circuit, C1 has to be at least 10 times higher than the parasitic capacitance associated to its connected node. The control of the startup time is done by charging the most capacitive node in the circuit.

4. Results

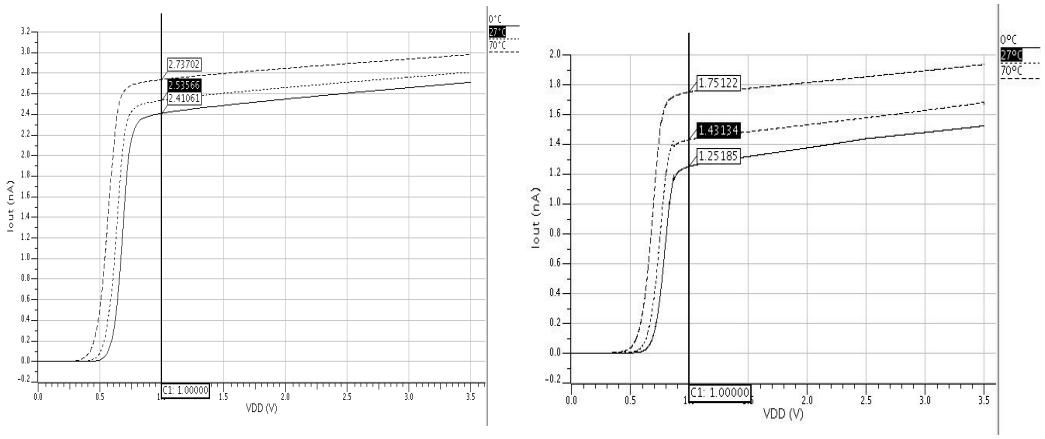
Two SBCS were designed for biasing NMOS and PMOS transistors. The technology used was AMS 0.35µm and the BSIM3v3 model was used for simulations in the ELDO Spice simulator. The layouts of the circuits are presented in fig. 6. The occupied area by each circuit was of 0.0147 mm^2 and 0.0146 mm^2 for the NMOS and PMOS version respectively.



(a) (b)

Fig.6 – SBCS layout for biasing (a) NMOS and (b) PMOS transistors

Fig. 7 shows the extracted simulation of the output reference current of both SBCS as a function of the voltage supply at different temperatures. From these graphics the value of the Power Supply Regulation (PSR) can be calculated which gives the idea of the sensitivity of the circuits from DC variations on the voltage supply.



(a) (b)

Fig.7 – Output reference current of the (a) NMOS and (b) PMOS SBCS

Finally, tab. 1 summarizes the main characteristics of both circuits.

Tab.1 – Results of the designed SBCS

	NMOS			PMOS			
Characteristic	**Min**	**Typ**	**Max**	**Min**	**Typ**	**Max**	**Unit**
Iout @1V	2.4	2.5	2.7	1.25	1.4	1.75	nA
Power Supply Reg.		3.8		5.5	6	9	%/V
Total Power Consumption	9.6	10	10.8	5	5.6	7	nW
Start Time	0.2		11	0.4	0.5	11	ms
Area		0.0147			0.0146		mm²

5. Conclusions

Two SBCS proportional to NMOS and PMOS transistors specific current in the AMS 0.35µm have been presented. With these circuits, any transistor can be biased in a well defined inversion level without dependence on the variation of technological parameters during fabrication process. Both SBCS can be operated at low supply voltage and exhibit ultra-low power consumption. The area occupied by each version is almost the same.

6. Acknowledgment

The authors of this work want to thank CNPq, the Brazilian Agency of Science and Technology, for financial support of this work.

7. Bibliography

[1] CAMACHO-GALEANO, M. E. “Referência de Corrente CMOS para Aplicações de Ultrabaixo Consumo de Potência”. Florianópolis, 2004. Dissertação (Mestrado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina

[2] A. I. A Cunha et al., “An MOS transistor model for analog circuit design,” *IEEE J. Solid-State Circuits,* vol. 33, no. 10, pp. 1510-1519, Oct. 1998.

Automatic Sizing of Analog Integrated Circuits Including Analysis of Parameter Variation

Lucas Compassi Severo, Alessandro Girardi
Lucascs.eletrica@gmail.com, alessandro.girardi@unipampa.edu.br

Federal University of Pampa – UNIPAMPA – Campus Alegrete

Abstract

This paper presents a tool for analog design automation (ADA) of integrated circuits using an external electrical simulator for evaluating the electrical specifications and genetic algorithms as optimization meta-heuristic. The purpose of the tool is to automatically size MOSFET transistors of a basic analog block, searching for optimized power consumption and gate area, and to evaluate the sensitivity of the electrical characteristics with respect to the variation of circuit parameters. As example, we designed a differential amplifier, which has 7 free variables. The results showed that the circuit scale reaches the design specifications, even considering the worst case of parameter variation.

1. Introduction

The design automation of analog integrated circuits is mandatory on state-of-the-art applications. The goal of automation is to provide an efficient search in the design space in order to make the circuit as efficient as possible within a set of arbitrary constraints and specifications. Several works have been done on this theme, with the main objective of providing low power optimized circuits in an acceptable computational time [1, 2].

In analog integrated circuits, it is important to automate some design stages such as transistor sizing and layout generation. The large number of free variables - and hence the large design space - makes it extremely difficult to resolve [3]. Thus, it is essential the use of computational optimization techniques in order to solve these problems.

In the transistor sizing stage, the automatic optimization-based design can be divided in two types: based on analytical equation model and based on electrical simulation. Using analytical equations, design specifications are evaluated by first order models, providing fast but inaccurate results. In the electrical simulation approach, circuit performance of an analog block is evaluated by electrical simulation provided by an spice-like electrical simulator with advanced device models. This approach is accurate but slow.

Another advantage of analog design automation is the possibility to analyze the sensitivity of the solution in relation to process parameters variation. Therefore, due to various manufacture stages of integrated circuits, the size and some physical parameters of the transistors experience some changes. These changes may cause some units of the designed circuit to fail to meet the nominal specifications of the design.

In this context, we propose a methodology for fully automatic sizing of analog basic blocks, evaluating the sensitivity to process variation by circuit electrical simulation. This methodology is based on the global search strategy performed by genetic algorithms [4, 5] and on the evaluation of the characteristics of the circuit by a commercial electrical simulator.

2. Automatic synthesis

The proposed methodology is based on optimizing a cost function determined from functional specifications of an analog basic block. The cost function is dependent on the electrical characteristics and variables of the circuit, such as power consumption, silicon area, and voltage gain, among others.

We also take into account the sensitivity to parameter variation as design constraints, beyond the specifications of the circuit. This ensures that the values of the specifications are maintained within a desired range even when small changes in the nominal values of the parameters are present [6].

The parameters of a MOSFET transistor can be divided into two categories: process parameters (physical characteristics of the semiconductor, such as oxide thickness or number of dopants in the substrate) and geometrical parameters (transistor sizes, such as gate width and length). Each parameter variation is related to a different cause and affects electrical characteristics in different aspects. Corner and Monte Carlo models provided by the foundries are important tools for simulating the electrical behavior of the designed block. Statistical simulations are mandatory, mainly in breakthrough designs, whose bias point is located, in general, on the limits of design space.

The optimization problem is modeled by a meta-heuristic optimization based on genetic algorithms using an external electrical simulator to provide values for the electrical characteristics of the circuit.

The genetic algorithm is a heuristic for nonlinear optimization inspired on an analogy with the theory of biological evolution [7]. It is a non-deterministic algorithm and it works with a variety of solutions (population), simultaneously. The size of this population is defined so as to maintain diversity in the population and, at the same time, to consider an efficient computational time. Each individual of the population is called a

chromosome and is composed by a vector with the variables of the problem. The quality of a chromosome can be evaluated by the cost function of the problem.

Figure 1 shows the design flow of the methodology using genetic algorithms. The algorithm takes as input a randomly generated population of solutions, the parameters for crossover and mutation and the parameters of the technology model. With this evaluation and selection method, the parent chromosomes are selected to generate new chromosomes. The generation of new chromosomes is given by a function of recombination and mutation [8]. The next step is to evaluate these new chromosomes and insert them in the population. Older and weaker member are excluded from the population. After each iteration, the stopping criterion is tested and, if true, the optimization is finished. If false, new parents are selected and the process continues.

The synthesis tool developed in this work used the GAOT implementation of genetic algorithms in Matlab® [10] and Dolphin Smash® as electrical simulator. As transistor model, we used the ACM model with parameters of AMS 0.35μm technology. Using this model, we guarantee an efficient search in strong, moderate and weak inversion regions, because it has continuous equations in all regions of operation of the transistors [9].

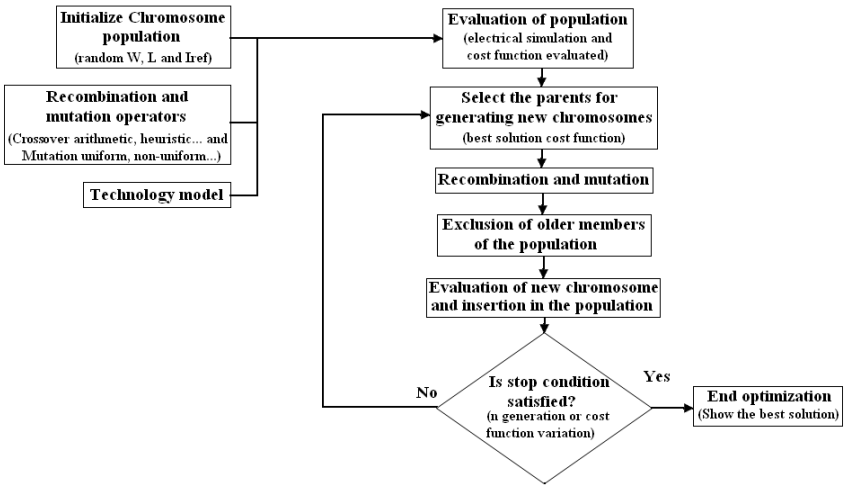


Fig. 1 - Analog design methology using Genetic Algorithms.

3. Design Example

As an example of the use of the proposed automatic analog synthesis methodology, we designed a differential amplifier with active load. This circuit is very versatile and usually serves as input stage for operational amplifiers [11]. The basic function here is to amplify the difference voltage between its inputs. The circuit is composed by a current mirror load (M2a and M2b), a differential pair (M1a and M1b) and a current mirror reference (M3a and M3b), shown in Figure 2. The design free variables are the width and length of each MOSFET transistor (W and L), and the value of the reference current source (Iref). As M1a and M1b are identical transistors, and the same occurs for M2a and M2b and for M3a and M3b, we have 7 free variables in this optimization problem: WM1a, LM1a, WM2a, LM2a , WM3a, LM3a and Iref. The population is evaluated using an external electrical simulator. Given arbitrary values for the design variables, the simulator is executed and the results (electrical characteristics) are read and evaluated in the cost function.

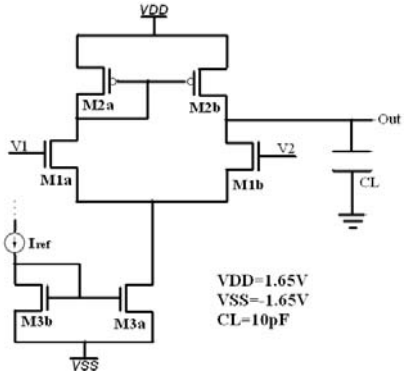


Fig. 2 - Schematics of a differential amplifier circuit with active load.

The main specifications in this circuit are: low frequency voltage gain (Av_0), gain-bandwidth product (GBW), slew-rate (SR), maximum and minimum common mode input voltage range (ICMR), dissipated power

(Pdiss) and consumed area. The analytical equations that describe the behavior of this circuit are well-known [11].

As the values of the specifications of the amplifier are provided by an external electric simulator, it is unnecessary the use of mathematical equations that model the differential amplifier. Thus, the values of the characteristics of the circuit are obtained by making AC, DC, OP and transient spice simulation analysis, as shown in Table 1.

Tab. 1 - Types of spice simulation analysis for the evaluation of circuit specifications.

Specification	Analysis
Av_0	AC
GBW	AC
SR	Transient
ICMR-	DC
ICMR+	DC
P_{diss}	OP

Tab. 2 - Worst result of voltage gain with respect to parameter variation.

Parameter	Worst case
W	Lower values
L	Lower values
Vt	Higher values
Tox	Higher values
U_0	Lower values

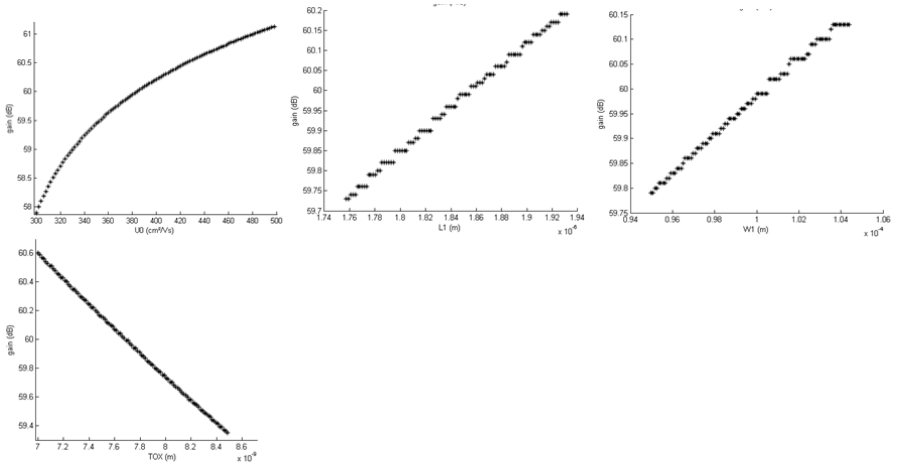
To model the sensitivity to variation of circuit parameters, we adopted a percentage change in geometric and process parameters in the circuit in order to express the variation which provides the worst possible outcome. For example, in an amplifier circuit it is always desired a high voltage gain Av0. Thus, the worst possible result is obtained by variations that cause the greatest reduction in voltage gain. For the analysis of variance, we consider channel width (W) and length (L) as geometric parameters, and threshold voltage (VT), silicon oxide thickness (Tox) and mobility of carriers in the channel (U0), as process related parameters. By means of electrical simulation, we analyzed which parameter variations cause the greatest degradation in the voltage gain of the circuit. Figure 3 shows the curves for the variation of some parameters and sensitivity to process parameters. The results of this analysis are shown in Table 2.

So, we adopted this behavior of parameter variations in order to emulate the sensitivity of voltage gain. The same study was realized for the other specifications.

As cost function for this analysis we assumed the sum of dissipated power, gate area and the constraints of the problem, defined by:

$$f = \frac{2.(v_{DD} + |v_{SS}|).I_{ref}}{P_0} + \frac{\sum_{i=1}^{3} W_i.L_i}{A_0} + R_R + R_S \quad (6)$$

where RR is a constraint function for design specifications considering nominal circuit values and RS is a constraint function including parameter variation. If all constraints are met, the values of RR and RS are equal to zero. P0 and A0 are the value of power and area references, respectively, used to normalize the terms of the cost function.



(a) (b) (c) (d)

Fig. 3 - Voltage gain variation with respect to variation of individual parameters. (a)Tox, (b)U0, (c)L and (d)W.

The optimization was executed using a population of 1000 individuals and a sensitivity analysis with a variation of 5% for the worst case of all parameters. Figure 4 shows the cost function variation along the iterations. In this figure it is possible to notice that the value of cost function is slowly reduced along the optimization process.

Table 3 shows the results of the optimization. In this table we can see that the values satisfy the initial specifications, achieving at a power consumption of 341.08µW and gate area equal to 726.75µm². The spent optimization time was approximately 51 minutes. As the variation of the parameters is part of the cost function, it guarantees that constraints are met even with a variability of 5% on geometric and processes parameters.

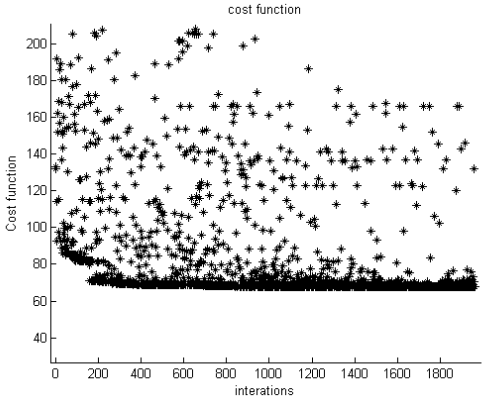


Fig. 4 - Cost function variation.

Tab. 3 - Optimization results for the differential amplifier.

Specification	Required	Achieved	Worst case variation
Av0	≥ 60.00 dB	60.45 dB	60.00 dB
GBW	≥ 1.00 MHz	5.45 MHz	5.46 MHz
SR	≥ 5.00 V/µs	5.00 V/µs	5.01 V/µs
ICRM-	≤ -0.70 V	-0.70 V	-0.70 V
ICMR+	≥ 0.70 V	1.18 V	1.18 V
Gate Area	Minimize	726.75 µm²	-
Power	Minimize	341.08µW	-
W(M1a e M1b)	-	84.52 µm	-
W(M2a e M2b)	-	49.56 µm	-
W(M3a e M3b)	-	34.04 µm	-
L(M1a e M1b)	-	2.25 µm	-
L(M2a e M2b)	-	2.25 µm	-
L(M3a e M3b)	-	12.15 µm	-
Iref	-	51.67 µA	-
Time	-	≈ 51 minutes	

4. Conclusion

This paper presented a tool for automatic design of a differential amplifier with 7 free variables, with optimization in area and dissipated power. Sensitivity analysis guarantees that the designed block can meet specifications even at worst cases, in which process parameters contributes for performance degradation. The employment of genetic algorithms provides the search in the entire design space, allied to the use of a transistor model continuous in all operation regions, which permits a good optimization in terms of dissipated power. The proposed automatic design methodology is precise and has good possibilities to be expanded for sizing large analog blocks with dozens of free variables. Another advantage of the methodology is the short time spent in modeling the circuit, because all evaluations are performed by electrical simulations, without the need of experienced human interference.

5. Acknowledgment

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. Goffart, 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] Taherzadeh-Sani, M. Lotfi, R. Zare-Hoseini, H. Shoaei, O., “Design optimization of analog integrated circuits using simulation-based genetic algorithm”, Signals, Circuits and Systems, 2003, International Symposium on, Iasi, Romania.

[6] T. Massier, S. Little, C. Myers, N. Seegmiller, T. Yoneda, “The Sizing Rules Method for CMOS and Bipolar Analog Integrated Circuit Synthesis”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 27, December 2008.

[7] P. Venkataraman, “Applied Optimization with Matlab Programming”, John Wiley & Sons, New York, 2002.

[8] R. Linden, “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.ise.ncsu.edu/mirage/GAToolBox/gaot/.

[11] P. E. Allen and D. R. Holberg, “CMOS Analog Circuit Design”, Oxford University Press, Oxford, Second Edition, 2002.

Photoluminescence Behavior of Si Nanocrystals Produced by Hot Implantation into Silicon Nitride

[1]Fellipe C. Pereira, [1]Pietro S. Konzgen, [2]Felipe L. Bregolin, [1]Uilson S. Sias
pereira.cti@gmail.com, pietroserpa@yahoo.com.br, felipe.bregolin@ufrgs.br, uilson.sias@gmail.com.

[1]Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense - Campus Pelotas, RS - Brazil
[2]Instituto de Física, Universidade Federal Rio Grande do Sul, Porto Alegre, RS - Brazil

Abstract

Silicon nanoparticles have being known for their intense light emission in the visible and near-infrared spectral range, emerging as a quite attractive alternative for development of optoelectronic devices. A recent improvement on emission quality was achieved by performing the Si implantation on heated substrates (hot implantation). In the present contribution we report the photoluminescence (PL) from Si nanocrystals (Si NCs) embedded into a 380 nm thick Si_3N_4 matrix, produced by Si hot implantation. The sample excitation was performed by an Ar ion laser at 488nm, obtaining a PL emission in the 400 - 900 nm range. The excess of Si was obtained by a 170 keV implantation with the substrate at different temperatures with the fluence of $1x10^{17}$ Si/cm^2. Samples were annealed since the room temperature (RT) up to 600˚C in a N_2 atmosphere. We have observed that samples annealed at 475˚C and implanted at 200˚C present a 20% PL increase as compared to RT implanted one. The PL emission was attributed to radiative states at the matrix and silicon nanocrystals interface.

1. Introduction

Since the discovery of strong visible light emission in porous Si and in Si nanocrystallites an intense research activity has been devoted in studying Si nanostructures due to their promising applications in optoelectronic and photonic devices [1]-[3]. Quantum confinement has led to a dramatic improvement of light generation efficiency in Si nanostructures [4], [5]. High emission efficiencies have been achieved in oxidized porous Si [4-6] and sizeable optical gain have been obtained from Si nanocrystals (Si NCs) embedded in a SiO_2 matrix, as recently demonstrated by several groups [1], [7]-[9]. In addition to the quantum confinement, the presence of interface states located at the nanocrystals surface has also shown a strong influence on the light emission from the Si NCs. In fact, for some systems the mechanism responsible for the light emission is still an open question.

On the other hand, in the last years an increasing research activity has been observed concerning the light emission from Si NCs embedded in Si nitride films [10]-[12]. Photoluminescence (PL) bands have been observed in the range of 400-900 nm, being their origin still unclear. The number of PL bands with quite different characteristics already reported in the literature should be attributed to the origin of the Si NCs, obtained either from near stoichiometric or from nitrogen/silicon–rich amorphous SiN_x films.

In the present article we report the PL results obtained by Si ion implantation into a stoichiometric Si_3N_4 film followed by a high temperature anneal. To our knowledge this is the first time that ion implantation technique has been employed in this system in order to obtain the Si NCs. In order to maximize the PL yield we have changed the annealing temperature, the implantation fluence and even the temperature of implantation. Finally, we have compared the present with the previous results already reported in the literature.

2. Experimental details

A 380 nm thick Si_3N_4 film was grown on a Si <100> wafer by the plasma enhanced chemical vapor deposition technique. The stoichiometry was verified using the Rutherford backscattering technique. Samples were implanted keeping constant the substrate temperature from room temperature (RT) up to 600 ^oC with 170 keV Si ions at a fluence of $1x10^{17}$ Si/cm^2 providing a peak concentration profile at around 100 nm-depth and an initial Si excess of about 10%. The as-implanted samples were further annealed for one hour at temperatures that range from 350 up to 900 ^oC in a N_2 atmosphere using a conventional quartz-tube furnace. This is a standard procedure in order to nucleate and precipitate the Si NCs and, eventually, remove the implantation damage [13].

Subsequently we have changed the implantation temperature in order to observe if this procedure induce any change in the PL yield and /or in its position. In previous experiments dealing with Si NCs embedded in a SiO_2 matrix we have shown that the implantation temperature plays an important role not only in the induced

PL yield but also in the position of the PL band [7]. As a final step, with the aim to verify if the Si excess on the Si_3N_4 film has any influence on the PL emission behavior, we have changed the Si implantation fluence.

The PL measurements were performed at RT using an Ar ion laser (488 nm) as an excitation source. The emission was dispersed by a 0.3 m single spectrometer and detected with a visible near-infrared silicon detector (wavelength range from 400 to 1100 nm).

3. Experimental results

3.1. PL intensity as a function of the annealing temperature:

In this sequence of experiments we have implanted the samples at RT and then submitted those to one hour anneal in N_2 atmosphere in a range of temperatures from 350 up to 900 ^oC.

In Fig.1 are shown typical PL spectra of samples annealed at different temperatures. It can be observed that the spectra are broad ranging from 600 up to around 1100 nm with a small structure at around 750 nm which should be attributed to reflection at the Si/SiO_2 interface. In all cases the maximum of the PL band remains around 900 nm without showing any dependence with the annealing temperature.

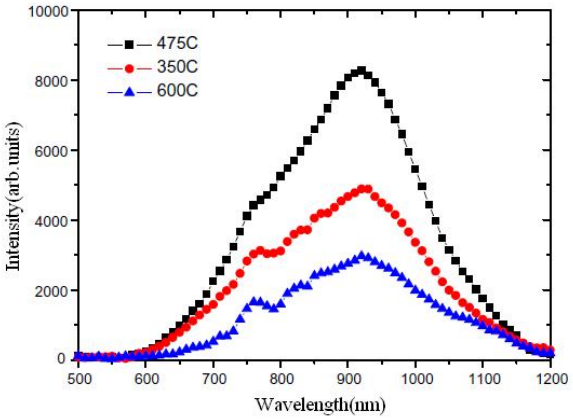


Fig. 1 – Typical PL spectra of samples implanted at RT and annealed at different temperatures.

In Fig. 2 are summarized the results of the present experiment where the maximum PL yields is shown as a function of the annealing temperature. As can be observed the higher yield was obtained for T_a =475 ^oC. For lower or higher temperatures the PL yield decreases without substantial change in their shape. We have also performed the annealing procedure in vacuum, Ar or even in a forming gas atmosphere without obtaining any substantial change in the PL emission.

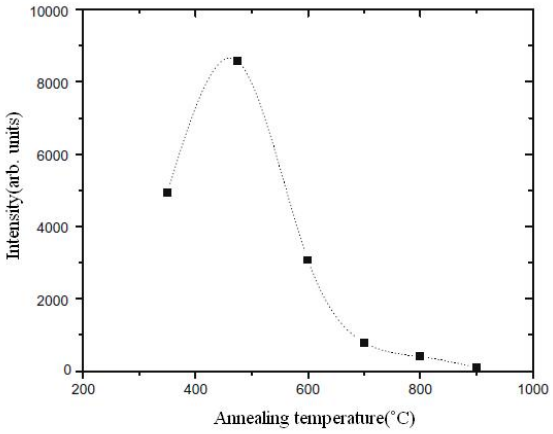


Fig. 2 – PL intensity as a function of annealing temperature for samples implanted at RT with a fluence of $1x10^{17}$ Si/cm^2.

3.2. PL intensity as a function of the implantation temperature:

In this set of experiments we have changed the implantation temperature (Ti) from 200 up to 600 ^oC and subsequently annealed the samples at 475 ^oC. The best result was obtained at Ti= 200 ^oC. For this implantation

temperature the PL yield is 20 % higher as compared with the one obtained with RT implantation. For higher Ti the PL even became lower when compared with the RT results see Fig. 3. Finally, it is interesting to point out that a direct implantation at Ti = 475 °C without further annealing was able to induce the same PL band but with a yield that was three times lower than the one obtained with Ti = 200 °C and Ta = 475 °C – see Fig. 3. As observed in Fig. 3 for the implantation at 475˚C, even without annealing, a PL is already induced.

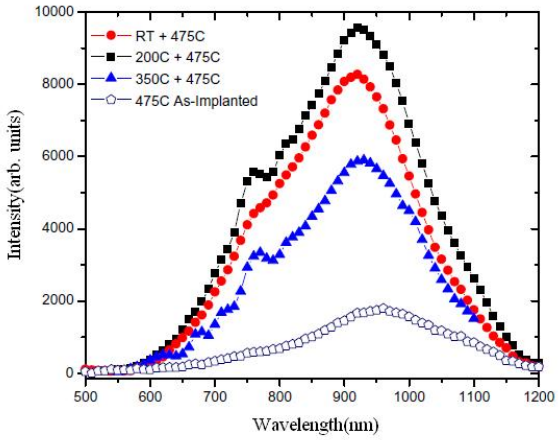


Fig. 3 – PL intensity as a function of implantation temperature for samples annealed at 475˚C and implanted at a fluence of $1x10^{17}$ Si/cm^2.

3.3. PL intensity as a function of the implantation fluence:

As a last step we have changed the Si excess at the Si_3N_4 film by varying the implantation fluence between 0.5 and $2.0x10^{17}$ Si/cm^2. This was done at T_i = 200 °C and T_a= 475 °C. As revealed by analysis of Transmission Electron Microscopy (TEM), the Si NCs size have changed with the implantation fluence, however, no variation have been observed nor in the yield neither on the position of the PL band. This is a clear indication that the origin of the present band is due to Si Ncs/SiO_2 interface radiative states.

4. Discussion and Conclusions

Although systems formed by Si nanocrystals embedded in SiO_2 matrix have an excellent structural stability, which is crucial for reliable fabrication of light emitting devices (LEDs), they have some problems. For example, the high annealing temperature over 1100 °C to form the luminescent nanocrystals is very high when integrating the LEDs with electronic components. Another problem is related to carrier injection for electroluminescence, which takes place by a tunneling mechanism. Being the SiO_2 matrix band gap of order 8.5 eV, the operating voltage of the LEDs should be inappropriately high. Then, would be suitable to have Si NCs produced in a low temperature and in a matrix with smaller band gap, like Si_3N_4.

In the last decade it has been observed an increasing activity related to the search for enhancing and explaining the PL emission from Si NCs embedded into a Si nitride film.

Up to the moment all the experiments were done either using Si or N into amorphous SiNx or non-stoichiometric nitride films. In this way PL bands ranging between 400 up to 900 nm were obtained. Moreover, in [11] it was shown that, by varying the conditions used to form the non-stoichiometric films the authors were able to change the characteristic wavelength of the emitted PL band.

To our knowledge the present work is the first attempt to obtain Si NCs into a stoichiometric nitride film by using the ion implantation technique followed by further annealing. The 170 keV, RT Si implanted into the Si_3N_4 film, followed by a 475 °C anneal has produced Si NCs which, when excited, have given place to a broad PL distribution centered at around 900 nm. High temperature implantation (at 200 °C) has increased the PL yield (20 %) when compared to the RT implantation. This behavior is qualitatively similar with what was observed for the case for Si NCs [7] and Ge NCs [14] produced in a SiO_2 matrix, where a strong increase in the PL yield was obtained as a result of a high temperature implantation.

It was also shown that the annealing atmosphere does not have any influence on the characteristics of the PL band. Even anneals performed in a forming gas environment did not produce any change in the PL band.

When the present results are compared to the ones obtained in previous works, which have used different techniques to provide the Si excess, it can be observed that in most of the cases the obtained PL bands were centered in the 400-600 nm region. One exception can be observed in the reference [12]. By using the plasma enhanced chemical vapor deposition technique to obtain a non-stoichiometric film they have obtained a PL band similar to the one obtained in the present work. However, one major difference that is found between both methods in order to obtain the maximum PL yield is the annealing temperature used in each experiment. While

in the present case it was 475 ^{o}C, in the cited work it was 700 ^{o}C. This feature could be attributed to the fact that for Si NCs growth kinetics in nitrides is faster than in other matrix like for example SiO_2 which require higher annealing temperatures (Ta =1100 ^{o}C). Consequently, since the implantation process already produce an excess of Si in a localized region (Rp= 100 nm and ΔRp= 60 nm), then it requires a lower temperature to form small size Si NCs. Moreover, as was described above, a high temperature implantation is already enough to induce the formation of Si NCs, but with less efficiency than the normal procedure of initially performing the implantation and then, the high temperature anneal.

By changing the Si implantation fluence we have modified the size of the nanocrystals, as revealed by TEM observations. However, no modification was observed nor in the yield, neither in the position of the PL band. This last feature clearly indicates that the origin of the present band is due to radiative defects at the interface between the nanocrystals and the matrix. This feature is in agreement with what was observed in [12] where the authors attributed the existence of the PL band to radiative defects of the type Si-N.

5. References

[1] L. Pavesi, L. dal Negro, C. Mazzoleni, G. Franzo and F. Priolo. “Optical gain in silicon nanocrystals”. London: Nature, 408, 2000, pp., 440 - 444.

[2] A. T. Fiori and N. M. Ravindra. “Light emission from silicon: Some perspectives and applications”. Boston: J. Electron. Mater., 32, 2003, pp 1043-1051.

[3] L. Brus in Semiconductors and Semimetals edited by D. Lockwood. New York: Academic, 1998, 49, p. 303.

[4] Silicon Photonics, edited by L. Pavesi and D.J. Lockwood (Springer, Berlin, 2004).

[5] B. Gelloz. “Possible explanation of the contradictory results on the porous silicon photoluminescence evolution after low temperature treatments”. Amsterdam: Appl. Surf. Sci., 108, 1997, pp. 449-454.

[6] B. Gelloz, T. Nakagawa and N. Koshida. “Enhancement of the quantum efficiency and stability of electroluminescence from porous silicon by anodic passivation”. Woodbury: Appl. Phys. Lett., 73, 1998, pp. 2021-2023.

[7] U.S. Sias, L. Amaral, M. Behar, H Boudinov, E.C. Moreira and E. Ribeiro. “Photoluminescence behavior of Si nanocrystals as a function of the implantation temperature and excitation power density”. Woodbury: J. Appl. Phys.,98, 2005, pp. 34312-34317.

[8] L. dal Negro, M. Cazannelli, L. Pavesi, S. Ossicini, D. Pavesi, G. Franzo, F. Priolo and F. Iacona. “Dynamics of stimulated emission in silicon nanocrystals”. Woodbury: Appl. Phys. Lett., 82, 2003, pp. 4636-4638.

[9] L. Kriachtchev, M. Rasanen, S. Novikov and J. Sinkkonen. “Optical gain in Si/SiO2 lattice: Experimental evidence with nanosecond pulses “. Woodbury: J. Appl. Phys., 79, 2001, pp.1249- 1251.

[10] Kwang Son Seol, Tsuyoshi Futami, Takashi Watanabe, Yoshimichi Ohki. “Effects of ion implantation and thermal annealing on the photoluminescence in amorphous silicon nitride”. Woodbury: J Appl. Phys., 85, 1999, pp. 6746-6750.

[11] Y.Q.Wang, Y.G.Wang, L. Cao and Z. X. Cao. “High-efficiency visible photoluminescence from amorphous silicon nanoparticles embedded in silicon nitride”. Woodbury: Appl. Phys. Lett., 83, 2003, pp. 3474 - 3476.

[12] L. dal Negro, J. H. Yi, J. Michel, L. C. Kimerling, T. W. F. Chang, V. Sukhovatkin and E.H. Sargent. “Light emission efficiency and dynamics in silicon-rich silicon nitride films”. Woodbury: Appl. Phys. Lett., 88, 2006, pp. 233109 -233111.

[13] M. Y. Valahnk, V. A. Yukhimchuk, V. Y. Bratus, A. A. Konchits, P.L.F. Hemment, T. Komoda, J. Appl. Phys. 85, 68 (1999).A. G. Cullis, and L. T. Canham. “Visible light emission due to quantum size effects in highly porous crystalline silicon”. London: Nature, 1991, pp. 335-338.

[14] F.L. Bregolin, and M. Behar, and U.S. Sias, and E.C. Moreira. “Photoluminescence induced from hot Ge implantation into SiO_2”. Holanda: Nucl. Instr. and Meth. B, 2009, pp. 1321- 1323.

Study of the Electroluminescence from Ge Nanocrystals Obtained by Hot Ion Implantation into SiO_2

[1]Pietro S. Konzgen, [1]Fellipe C. Pereira, [1]Uilson S. Sias, [2]Felipe L. Bregolin
pietroserpa@yahoo.com.br, pereira.cti@gmail.com, uilson.sias@gmail.com, felipe.bregolin@ufrgs.br

[1]Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense Campus Pelotas, RS-Brazil
[2] Instituto de Física, Universidade Federal Rio Grande do Sul, Porto Alegre, RS, Brazil

Abstract

Usually, photoluminescence (PL) and electroluminescence (EL) from Ge nanocrystals (Ge NCs) have been obtained by room temperature (RT) Ge implantation into a SiO_2 matrix followed by a high temperature annealing. In the present work, we have used a novel experimental approach: we have performed the Ge implantation at high temperature (Ti) and subsequently a high temperature annealing at 900 °C in order to grow the Ge NCs. By performing the implantation with the matrix kept at Ti = 350 °C, the electrical stability of the MOSLED devices were enhanced, as compared to the ones obtained from RT implantation. Moreover, by changing the implantation fluence with $\Phi = 0.5 \times 10^{16}$ and 1.0×10^{16} Ge/cm^2 we have observed a blueshift in the EL emission peak.

1. Introduction

Since the discovery of photoluminescence (PL) in porous Si [1], a large number of studies concerning the properties of Si or Ge nanoclusters (NCs) have been reported. Among the several techniques used to produce the NCs embedded in a matrix, the ion implantation technique has shown to be a very reliable tool because it offers several advantages [2]-[4]: in addition to the compatibility with the microelectronic technology, it is very precise in controlling the amount and depth of the excess ions introduced in the matrix, thus presenting great reproducibility.

First experiments using ion implantation as a technique to produce Si or Ge NCs were already reported in the early 1990's [2]-[4] and their promising results were followed by an intense research activity, as illustrated by the review of Rebohle et al. [5]. However, in all the cases, the Ge implantation was performed at room temperature (RT), followed by a high temperature anneal.

Recently we have used a different experimental approach. Instead of performing Ge implantation into the SiO2 layer at RT, we have done it keeping the substrate at 350 °C and then annealed the samples at 900 °C. As consequence, the 390 nm band increased its PL yield by a factor of almost four as compared with the RT implantation. Moreover by finding the proper Ge implanted concentration we were able to further increase the PL yield of the 390 nm band by another factor of 3 [6].

The main goal of the present paper is to study the electroluminescence (EL) emitted by metal–oxide–semiconductor (MOS) devices made with the Ge NCs obtained by hot implantation and compared the results with those produced by RT implantation results of EL obtained using implantation at hot and at RT in the various fluencies of Ge used.

2. Experimental procedure

A 195 nm-thick SiO_2 layer, thermally grown onto a n-type Si <100> wafer by dry oxidation at 1050 ºC, was implanted with 120 keV Ge ions keeping constant the substrate temperature at RT and 350 ºC, respectively. The implantations were done at fluences of 0.5 and 1.0 x 10^{16} Ge/cm², corresponding to a Gaussian-like depth profile with a peak concentration of about 1.5 at. % and 3 at. %, respectively, 90 nm far from the SiO_2 surface. Subsequently, the as-implanted samples were submitted to a furnace anneal at 900 ºC for 30 min in flowing N_2. Then, a SiON layer with a thickness of 100 nm was deposited onto the SiO_2 layer by plasma-enhanced chemical vapor deposition (PECVD) in order to enhance the electrical stability of the device [7], followed by the same annealing process. MOS dot structures for EL studies were prepared using sputtered layers of indium tin oxide (ITO) and Al as front and rear electrodes, with a thickness of 100 nm and 150 nm, respectively. Photolithography was used to make a dot matrix pattern with a dot diameter of 200 μm. Finally, an annealing procedure of 400 °C for 30 min was performed to improve the ohmic behavior of the contacts. A sketch of the device is shown in Fig. 1.

The EL measurements were performed at room temperature utilizing a Triax 320 spectrometer with a R928 Hamamatsu photomultiplier. Current injection was done by a Keithley 2410 sourcemeter, with a positive voltage applied to the gate, corresponding to a electron injection from the Si substrate into the SiO_2 layer.

Structural characterization of the samples were performed by transmission electron microscopy (TEM), using a 200 keV JEOL microscope with the samples prepared in a cross sectional mode by mechanical polishing and ion milling techniques.

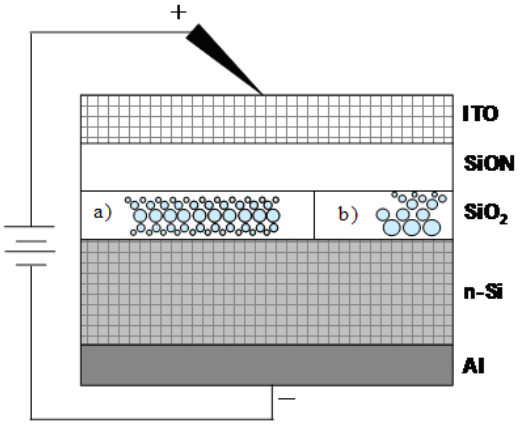


Fig. 1 – Schematic diagram of the MOSLED (not drawn to scale). In Detail: Representation of the Ge NCs for a) RT implantation b) High temperature implantation

3. Experimental results

3.1. TEM Results

The TEM measurements reveal the formation of crystalline Ge nanoclusters in both RT and hot implanted samples after the 900 ºC annealing, as shown in Fig. 2 a) and b), respectively. For the RT implanted sample (Fig. 2a) we have found a Gaussian–like NCs size distribution with quite regular sizes varying from 2.5 up to 5 nm diameter, resulting in a mean diameter of 4.2 nm (σ=1.2 nm).

Concerning the hot implantation (Fig. 2b), the mean size and size distribution differ significantly from the one observed when the Ge implantation is done at RT. In fact, the Ge NCs distribution presents a positive gradient profile of crystal sizes along depth. The shallow region shows quite small nanocrystals having about 2 to 3 nm in diameter. The intermediate one contains medium size Ge NCs of around 3 to 5 nm and in the deepest region it is possible to observe larger NCs ranging from 5 to even 9 nm in diameter.

3.2. Electroluminescence measurements

In this set of experiments, the electrical properties of the MOS light emitting devices (MOSLED) were analyzed. Fig. 3 shows the EL spectra of the devices under a constant current density of 320 μA/cm² for the hot and RT samples implanted at the two different fluences. The first observed feature is that the intensities of the EL peaks of the hot implanted samples are around 30 % lower than the ones corresponding to the RT implanted ones. Another interesting feature present in this spectra is the blueshift in the EL peak position, of 8 nm, in the devices implanted with the higher fluence (1.0×10^{16} Ge/cm²), as compared to the ones implanted with the lower fluence (0.5×10^{16} Ge/cm²). It should be mentioned that the 310 nm band seen in [6] could not be detected in the present experiment because non UV-transparent optics were utilized.

Figure 4 displays the EL intensity as a function of the injected carriers under constant current density of 320 μA/cm^2 with the spectrometer centered on the wavelength of the EL peak of each device. The experiments were done for the hot and the RT implanted samples, and for both implanted fluences. It was verified that the hot implanted samples show an electrical stability that is around three times larger than the one obtained by the RT implants.

4. Discussion and conclusions

In previous works [4]-[6] the Ge NCs embedded in SiO_2 matrix were obtained by RT implantation followed by a high temperature anneal. When excited at 5.1 eV, two PL bands were obtained, one at 310 nm (4.0 eV) and the other, with much higher yield at 390 nm (3.1 eV). The origin of the PL bands was attributed to radiative defects present at the Ge NCs/matrix interface, specifically, neutral oxygen vacancies (NOV) like ≡Ge–Si≡ and/or ≡Ge–Ge≡ defects generated the local deficiency of oxygen and the incorporation of Ge into the SiO2 network surrounding the NCs [5], [8],[9].

Related to the EL measurements, the lower EL intensity of the hot implanted samples–see Fig. 3–can be explained based on the results of the TEM observations. The hot implanted samples have significant larger NCs at the deepest region of the implantation profile, as compared to the RT implanted ones. These larger NCs act as scattering centers for the electrons during the injection process as illustrated by Fig. 1 causing a kinetic energy loss and thus decreasing the corresponding EL cross section, producing a less intense emission. The blueshift in the EL spectra observed for the highest implantation fluence (see Fig. 3) can qualitatively be explained as follows. The EL induced by the Ge NCs is due to NOV-type radiative defects, such as ≡Ge–Si≡ and/or ≡Ge–Ge≡, with emission energies of 2.92 and 3.1 eV, respectively [7]. Both of them are among the ones that contribute the most to the observed EL band. A higher Ge implantation fluence produces larger NCs, after the thermal annealing, due to the higher Ge concentration in the matrix. Consequently, the or ≡Ge–Ge≡ to ≡Ge–Si≡ ratio increases, producing a more intense emission of the 3.1 eV component and a corresponding reduction of the 2.92 eV component of the EL band, resulting in a slight blueshift, as observed in Fig. 3.

Fig. 4 indicates that the hot implanted samples can sustain an approximately three times larger number of injected charges before the breakdown device (Q_{BD}) occurs, as compared to the RT implanted ones. Since the Q_{BD} is depending, among other factors, on the injected current density and operation time, this means that a MOSLED made utilizing hot implantation can sustain a current density three times higher, giving a higher EL intensity, or, for the same current density, results in a three times higher operation time. As the breakdown is a statistical event, the improvement factor may vary, however, the general tendency is clear.

The above feature in principle can be attributed to the fact that the hot implantation produces less damage in the SiO_2 layer during the implantation process and a higher quality of the SiO_2/Si interface, thus resulting in lower number of nonradiative defects present in the oxide and in the interface.

In order to compare, under the same conditions the PL and EL emission yields, all the implantation and annealing parameters were the ones reported in [6], which gave place to the maximum PL emission. It is possible that the best conditions for the PL emission are not the same as the EL ones.

In summary, in the present communication we have found that devices based on Ge NCs produced by hot implantation have greater electrical stability, as compared with the ones produced by RT implantation. Concerning the EL yield, as mentioned above, it is possible that the best conditions for the EL emission are not the same as compared with the PL ones. It is necessary to perform further optimizations in order to increase the EL emission of the MOS devices. In this sense, work is on the way.

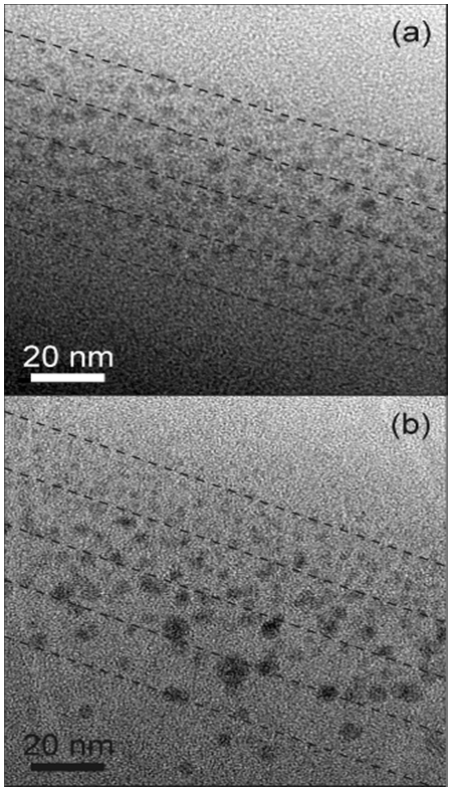


Fig 2- Detailed TEM view showing the NCs size distribution for a) RT implantation b) high temperature implantation (samples implanted to $\Phi = 1.0 \times 10^{16}$ Ge/cm^2 and annealed at 900 ^{o}C for 1 h.

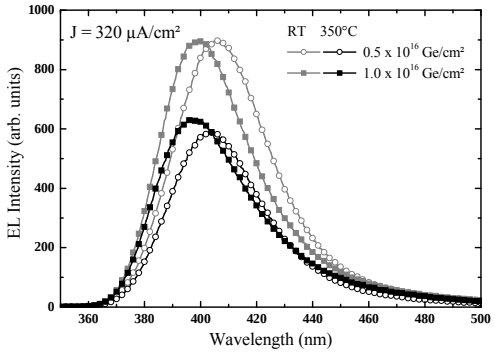


Fig. 3 – EL spectra from the MOS devices, implanted at different temperatures(gray lines:RT, black lines:350˚C) RT, Black lines: 350 ºC), and fluences (Open circles: Φ = 0.5 x 10^{16} Ge/cm^2, Solid Squares:Φ = 1.0 x 10^{16} Ge/cm^2).

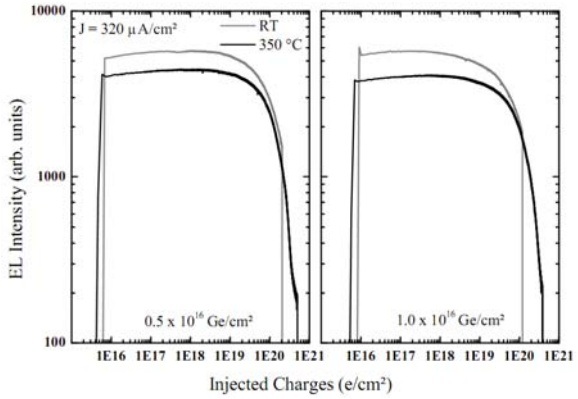


Fig 4 - EL peak intensity over injected charges, implanted at different temperatures (gray lines: RT and black lines: 350˚C) and fluences (Left: Φ = 0.5 x 10^{16} Ge/cm^2, Right: Φ = 1.0 x 10^{16} Ge/cm^2)

5. References

[1] L. T Canham, "Silicon quantum wire array fabrication by electrochemical and chemical dissolution of wafers." Woodbury: Appl. Phys. Lett., 57, 1990, pp. 1046-1048.

[2] T. Shimizu-Iwayama, K. Fujita, S. Nakao, K. Saitoh, T. Fujita, and N.Itoh. "Visible photoluminescence in Si^+-implanted silica glass". Woodbury: J. Appl. Phys. 75, 1994, pp.7779-7784.

[3] L. Rebohle, J. von Borany, R. A. Yankov, W. Skorupa, I. E. Tyschenco, H.Fröb, and K. Leo. "Strong blue and violet photoluminescence and electroluminescence from germanium-implanted and silicon-implanted silicon-dioxide layers". Melville: Appl. Phys. Lett., 71, 1997, pp. 2809-2811.

[4] K. V. Shcheglov, C. M Yang, K. J. Vahala, and H. A. Atwater, "Electroluminescence and photoluminescence of Ge-implanted Si/SiO_2/Si structures". Melville: Appl. Phys. Lett., 66, 1995, pp. 745-747.

[5] L.Rebohle, J. von Borany, H. Fröb, and W.Skorupa. "Blue photo- and electroluminescence of silicon dioxide layers ion-implanted with group IV elements". Berlin/Heidelberg: App. Phys. B: Laser and Optics, 71, 2000, pp. 131-151.

[6] F. L. Bregolin, M. Behar, U. S. Sias, and E. C. Moreira. "Photoluminescence induced from hot Ge implantation into SiO_2". Amsterdam: Nucl. Instrum. Methods Phys. Res. B, 267, 2009, pp.1321-1323.

[7] J. M. Sun, L. Rebohle, S. Prucnal, M. Helm, and W. Skorupa. "Giant stability enhancement of rare-earth implanted SiO2 light emitting devices by an additional SiON protection layer". Melville: Appl. Phys. Lett., 92, 2008, pp. 071103-071105.

[8] Y.H Ye, J.Y. Zhang, X.M. Bao,X.L Tan,and L.F Chen, "Visible photoluminescence from Ge+-implanted SiO2 films thermally grown on crystalline Si". Berlin/Heidelberg: Appl. Phys. A, 67, 1998, p. 213-217.

[9] J. M.J. Lopez, F. C. Zawislak, M. Behar, P. F. P.Fichtner, L. Rebohle, and W.Skorupa, "Cluster coarsening and luminescence emission intensity of Ge nanoclusters in SiO_2 layers". Melville: J. Appl. Phys., 94, 2003, pp. 6059-6064.

Digital Design and Embedded Systems

Designing NBTI Robust Gates

[1]Paulo F. Butzen, [1]Vinícius Dal Bem, [2]André I. Reis, [2]Renato P. Ribas
{pbutzen, vdbem, andreis, rpribas}@inf.ufrgs.br

[1]PGMICRO – Universidade Federal do Rio Grande do Sul
[2] Instituto de Informática – Universidade Federal do Rio Grande do Sul

Abstract

Negative Bias Temperature Instability (NBTI) has become a critical reliability for nanometer PMOS transistors. In this paper, the pull-up transistor arrangement of CMOS gates is restructured to reduce the number of PMOS transistors under severe NBTI biasing and consequently mitigates the performance degradation due to NBTI phenomenon. The proposed technique introduces no gate area penalty. Simulation results show that on average 10% of performance degradation due to NBTI can be recovered.

1. Introduction

CMOS technology has been permanently scaling down during last decades. Several aspects ignored in earlier technology nodes, such as leakage currents, variability and aging effects, are becoming critical concerns in nanoscaled design. Aging effects, particularly Negative Bias Temperature Instability (NBTI), are factors that limit circuit lifetime by modifying their characteristics over time. NBTI refers to the generation of positive oxide charge and interface traps in metal-oxide-silicon structure under negative gate voltage bias ($V_{gs} = -V_{dd}$), in particular at elevated temperature. It mostly affects PMOS transistors, since these devices are negatively biased when they are conducting. This effect is measured as an increase in the transistor threshold voltage (V_{th}) magnitude, impacting consequently the device drive current and circuit speed.

NBTI degradation needs to be taken into account during the design to ensure product reliability over time. Several logic functions can be designed using different transistor arrangement structure, as the OAI21 gates depicted in Fig. 1. It is well know that these solutions present different delay and power consumption behaviors. They also present different levels of degradation due to NBTI. In this paper, the structural PMOS transistor arrangement is explored considering its influence in terms of the gate delay and NBTI degradation.

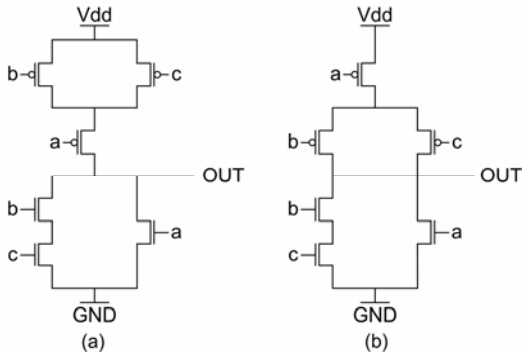


Fig. 1: Logically equivalent AOI21 CMOS gates.

2. Background

The NBTI degradation is proportional to the temperature, power supply, threshold voltage and probability of the transistor is negative gate biasing, i.e. the gate signal be '0' and the source potential be '1'. This probability defines the time that a PMOS transistor stays in the stress or recovery phase. A lower signal probability means a longer recovery time and a smaller V_{th} degradation due to NBTI effect.

When a transistor stack is considered, as in AOI gates depicted in Fig. 1, the steady state of the intermediate nodes determine whether each individual device is under negative bias stress [1]. It means that the signal probability of each PMOS transistor in a stack can not be considered individually. Instead of, it has to be associated to the device position in the transistor stack and to the other inputs probability. A basic rule to reduce the NBTI degradation can be simplified by "PMOS transistors that are connected to the power supply (V_{dd}) suffer more V_{th} degradation due to NBTI than the ones far from that". Fig.2 shows the degradation time (%) of a four transistor stack when all inputs have signal probability of 0.5.

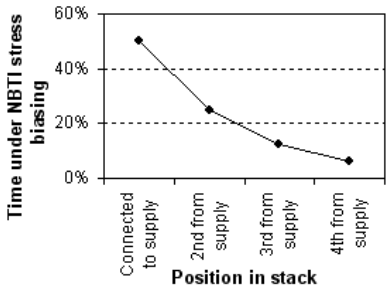


Fig. 2: Time under stress biasing versus position in a four transistor stack.

Several design techniques to mitigate NBTI effects are reported in the literature [1-4]. The tuning of V_{dd} and V_{th} are effective techniques to compensate the performance degradation due to NBTI, since it is exponentially dependent of these parameters [2]. Gate sizing has been another proposed technique to deal with performance uncertainties. Sufficient design margin to compensate NBTI degradation can be set before fabrication by oversizing the gate properly [3]. The signal probability and the position of the device in a transistor stack are explored in several ways. Pin reordering is used to reduce NBTI degradation when the circuit is in active mode [1]. Input vector is explored in order to minimize static degradation when the circuit is in standby state [4]. Functional symmetries are used to manipulate the stress probability by logic restructuring to reduce the NBTI effects in circuit critical paths [5].

3. Transistor Network Restructuring

Assuming all gate inputs with equal signal probability, PMOS transistors that are connected to the power supply (V_{dd}) suffer more V_{th} degradation due to NBTI than the ones far from that, as shown in Fig. 2. The proposed methodology explores the transistor arrangement restructuring in the PMOS pull-up plane to evaluate the robustness of CMOS gates in terms of the NBTI effect considering both signal probability and position of the transistor in a stack.

In simple PMOS stack topology, such as in NOR gates, the proposed restructuring approach is not applied since there is no structural difference in the position of the transistors in the stack. In this case, previously referenced techniques can be used to mitigate this aging effect.

However, in logic cells where more complex transistor arrangements are observed, such as AOI gates depicted in Fig. 1, the transistor network presents significant influence in gate delay, power consumption and also in delay degradation due to NBTI effect. It is expected a smaller degradation in the arrangements that have a smaller number of transistors closer to the power supply, i.e., a more robust CMOS gate design with respect to NBTI degradation may connect as few as possible transistors at V_{dd}.

4. Simulation Results

To confirm such behavior, the AOI gates depicted in Fig. 1, 3, and 4 were designed in a predictive 45nm CMOS technology process [5]. Two versions of each gate ("new" and "after 10 years of degradation") have been characterized using the Encounter Library Characterized considering an input slope of 50ps and a FANOUT4 load.

For a given time *'t'* and a probability of each device is under negative bias stress *'α'*, the V_{th} degradation can be predicted by the long term prediction model, as expressed in equation (1):

$$\Delta V_{th} = b.(\alpha.t)^n \tag{1}$$

where $b = 3.9 \times 10^{-3}\ V \times s^{-1/6}$ and '*n*' is the time exponential constant, equals to 0.16 [4].

To evaluate the PMOS transistor V_{th} degradation due to NBTI, the equation (1) has been considered for a time equal to 10 years and all input signal probability equal to 0.5. As mentioned before, the probability of each device is under negative bias stress in a transistor stack is not the same as the respective input signal probability. Both, the position of the device in the stack and the input signal probability have to be considered together to compute the probability of each device is under negative bias stress.

Tab. 1, Tab. 2 and Tab. 3 presents absolute the rise propagation delays and the delay degradation of each input signal and the average delay for the gates depicted in Fig. 1, Fig. 3, and Fig. 4 respectively. The results for the gates depicted in Fig. 3b and 4b are not presented since they represent an intermediate solution between the extreme cases illustrated in Fig. 3a – Fig. 3c, and Fig. 4a – Fig. 4c, respectively.

In the pull-up arrangement of the logic cell illustrated in Fig. 1a, two transistors suffer high NBTI degradation since they are connected directly to V_{dd}. In Fig. 1b, just one transistor is connected to V_{dd} and consequently suffers high degradation. Simulation results show that the delay degradation of minimum and maximum rise propagation delays do not present high differences between the two distinct arrangements, since they are the best case (transistor connected to cell output – small NBTI degradation) and the worst case (transistor connected at V_{dd} – high NBTI degradation). Differences are verified when the average value is

computed. In this case, the gate shown in Fig. 1b presents higher robustness than the one depicted in Fig. 1a in terms of delay degradation due to NBTI effect. This robustness is due to the fact that only one transistor in arrangement (b) suffers high NBTI degradation instead of two transistors in arrangement (a). The same analysis can be performed for the data presented in Tab. 2 and Tab. 3. On average, 10% of performance degradation due to NBTI effect can be recovered by restructuring the transistor arrangement.

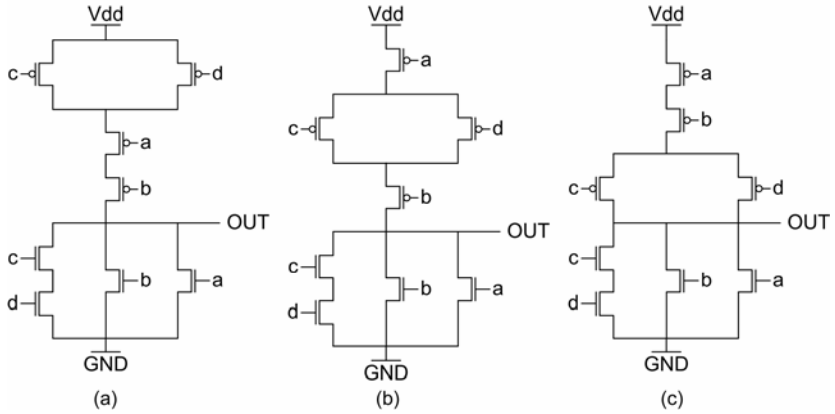


Fig. 3: Logically equivalent AOI211 CMOS gates.

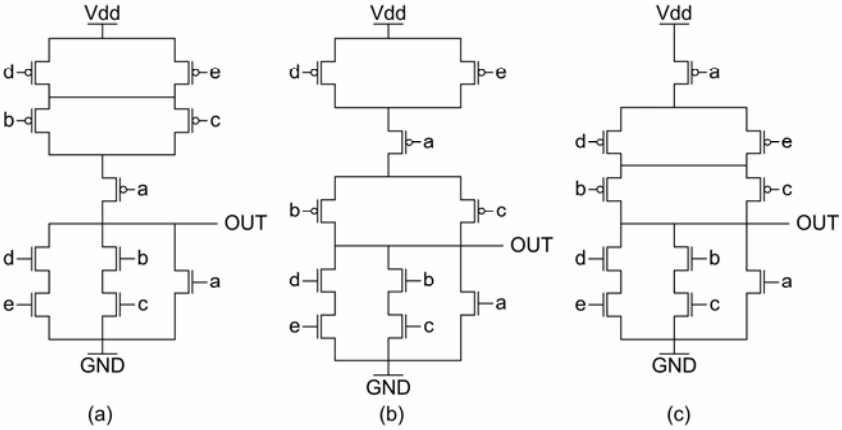


Fig. 4: Logically equivalent AOI221 CMOS gates.

Tab.1 – Gate Delay and degradation of AOI21 gates depicted in Fig.1

	Gate Delay (ps)		Difference		Delay Degradation (ps)		Difference	
	Fig. 1b	Fig. 1a	(ps)	(%)	Fig. 1b	Fig. 1a	(ps)	(%)
A to OUT	91,9	73,0	18,9	26%	8,3	5,7	2,5	44%
B to OUT	74,7	81,8	-7,0	-9%	5,2	7,3	-2,1	-28%
C to OUT	79,1	86,1	-7,0	-8%	5,6	7,9	-2,3	-30%
Average Delay	81,9	80,3	1,6	2%	6,4	7,0	-0,6	**-9%**

Tab.2 – Gate Delay and degradation of AOI211 gates depicted in Fig.3

	Gate Delay (ps)		Difference		Delay Degradation (ps)		Difference	
	Fig. 3c	Fig. 3a	(ps)	(%)	Fig. 3c	Fig. 3a	(ps)	(%)
A to OUT	139,2	116,2	23,1	20%	10,9	10,1	0,8	8%
B to OUT	131,9	94,4	37,5	40%	10,0	7,9	2,1	26%
C to OUT	95,5	122,6	-27,1	-22%	7,1	11,7	-4,6	-39%
D to OUT	100,7	127,5	-26,8	-21%	7,6	11,6	-4,0	-34%
Average Delay	116,8	115,2	1,7	1%	8,9	10,3	-1,4	**-14%**

Tab.3 – Gate Delay and degradation of AOI221 gates depicted in Fig.4

	Gate Delay (ps)		Difference		Delay Degradation (ps)		Difference	
	Fig. 4c	Fig. 4a	(ps)	(%)	Fig. 4c	Fig. 4a	(ps)	(%)
A to OUT	160,0	106,2	53,9	51%	13,9	8,6	5,3	62%
B to OUT	142,3	145,1	-2,8	-2%	11,2	13,1	-1,9	-15%
C to OUT	147,3	150,0	-2,7	-2%	12,5	14,1	-1,6	-11%
D to OUT	104,7	129,7	-25,0	-19%	8,1	10,5	-2,4	-23%
E to OUT	109,8	134,7	-24,9	-18%	8,6	11,4	-2,8	-24%
Average Delay	132,8	133,1	-0,3	0%	10,9	11,5	-0,7	**-6%**

5. Conclusions

An analysis that explores the transistor arrangement restructuring in the PMOS pull-up plane to evaluate the robustness of CMOS gates in terms of the NBTI effect considering both signal probability and position of the transistor in a stack is presented. Three different AOI gates have been evaluated and the results shows that on average, 10% of performance degradation due to NBTI effect can be recovered by restructuring the transistor arrangement.

6. Acknowledgments

Research partially funded by Nangate Inc under a Nangate/UFRGS research agreement, by CNPq Brazilian funding agency, and by the European Community's Seventh Framework Programme under grant 248538 – Synaptic

7. References

[1] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. “NBTI-Aware Synthesis of Digital Circuits”. DAC 2007.

[2] L. Zhang and R. P. Dick. “ Scheduled Voltage Scaling for Increasing Lifetime in the Presence of NBTI”. ASPDAC 2009.

[3] B. C. Paul et al. “Temporal Performance Degradation under NBTI: Estimation and Design for Improved Reliability of Nanoscale Circuits”. DATE 2006.

[4] Y. Wang et al. “Temperature-Aware NBTI Modeling and the Impact of Input Vector Control on Performance Degradation”. DATE 2007.

[5] K.-C. Wu and D. Marculescu. “Joint Logic Restructuring and Pin Reordering against NBTI-Induced Performance Degradation”, DATE 2009.

[6] N. H. E. Weste. “CMOS VLSI design: a circuits and systems perspective”. Boston: Pearson/Addison Wesley, 2005.

[5] FreePDK 45nm Predictive Technology. Avaliable at: http://www.eda.ncsu.edu/wiki/FreePDK. 2009.

A Study About Transient Vulnerabilities in Combinational Circuits

[1]Mayler G. A. Martins, [1]Fernanda L. Kastensmidt, [1]Renato P. Ribas, [1]André I. Reis
{mgamartins,fglima,rpribas,andreis}@inf.ufrgs.br

[1]Universidade Federal do Rio Grande do Sul

Abstract

The new technology ICs have become more vulnerable to radiation single-event transients (SET) which are temporary perturbation in logic gates and more critical with increased clock speeds. They are generated by alpha particle or neutron strikes, which can cause a transient voltage, probably generating a fault. In bulk CMOS the most sensitive areas are depletion regions at transistor drains. This paper proposes a study of combinational blocks and gates vulnerability to single event transients, using different descriptions for the same equation. This will be achieved by simulating the CMOS circuits and analyzing the effects induced by SETs. A tool was developed to achieve the objective, with SPICE parsing ability and with using event-driven simulation. The analysis is made exhaustively, checking all values in primary inputs and in the end of analysis, informing the degree of vulnerability of a combinational block or circuit. The result will be used for comparing different implementations for the same logic function.

1. Introduction

The advances in microelectronics field have pushed the silicon process limits in terms of scaling, making deep-sub micron (DSM) effects appear with more intensity, involving crosstalk, noise, tunnelling and second order effects, including short channel effect and drain current. Device shrinking, power supply reduction and increasing operating speeds that follow the technological evolution towards nanometric technologies, reduce significantly the noise margins and this the reliability of DSM ICs. This process is now approaching a point where it becomes unfeasible to produce ICs that are free from these effects.

The new technology ICs have become more vulnerable to radiation single-event transients (SET) which are temporary perturbation in logic gates and are more critical with increased clock speeds. They are generated by alpha particle or neutron strikes, which can cause a transient voltage in a part of the circuit and propagates through it and can latch to an erroneous data, probably generating a circuit fault [1]. In bulk CMOS the most sensitive areas are depletion regions at transistor drains. Hole-electrons produced here will be swept apart by the electric field that the injected charge tends to change the state of the struck node with a brief voltage pulse.

Fault injection is usually used to simulate single event upsets (SEUs), since it allows perturbing the systems with faults, generating effects of soft errors during normal circuit operation. There are many techniques used and the simulation-based fault injection is very interesting because it is flexible since faults can be injected in any component of the system. But it needs high CPU times for simulating complex circuits. [2]

In this context, it will become mandatory to design the future ICs that are used normally in aerospace equipment to be less vulnerable for SEUs. Sometimes, traditional fault tolerant design such as triple modular redundancy (TMR) or shielding cannot be used due its high cost. In safety critical systems, fault masking techniques are often used to obtain the necessary level of dependability [3].

This paper proposes a study of combinational blocks and gates vulnerability to SET, using different descriptions for the same equation. This will be achieved by simulating the CMOS circuits and analyzing the effects induced by SETs in nodes that are vulnerable to that fault. The analysis is made exhaustively, checking all values in primary inputs and in the end, informing the degree of vulnerability of a combinational block. The result will be used for comparing different implementations for the same logic function.

2. Background

Cosmic particle strikes are simulated using lasers and heavy ion beams to simulate them with a range of Linear Energy Transfer (LET) levels. The LET of a particle is a measure of how much energy is deposited as it passes through some material. The SET response of a component is dependent on LET for characterization. At high LET levels the particles have sufficient energy to cause an upset when they strike any sensitive node and this total sensitive area is known as the "saturated cross section". For radiation tolerant design it is desirable to raise the LET upset threshold as high as possible and to minimize the saturation cross section.

In the last years, several approaches to simulation-based transient analysis have been proposed. Some works have focused on modeling effects of transients occurring in registers. But a combinational logic can own a multiple fanout in the logic and generate multiple bit flips. Other works, proposed the use of switch-level simulators [4] for analyzing transient propagation through a circuit. The switch level can model important parameters of MOS circuits such as charge sharing and variant driving strengths.

3. Simulation and Transient Detection Algorithm

The tool concept used in analysis was adapted with a simplified form for execution of fault injection targeting soft errors in combinational logic and saturated cross-section for the analysis of this work.

The tool algorithm uses an event-driven simulator to load the inputs and evaluate the outputs of the circuits. An event-driven simulator evaluates only changes, e.g. The propagation of the value in a transistor is made automatically when there is a defined value for the gate and also a defined value for either drain or source. In this situation, the defined value in the drain (source) propagates to the source (drain).

The primary inputs are controlled, and all values are generated in the appropriate time, generating an exhaustive test. After simulation and analysis, all internal values are reset to default state, to assure the correct operation.

There is an initial transient count before the optimizations run. This adjust will calculate how many transistors in a logic gate will have a vulnerability. The tool considers the drain sharing (parallel MOS) and only counts one drain when this happens.

The first transient counting reduction is a global circuit optimization that can be considered a logical masking. It will analyze each gate and test other gates connected in the output of the former gate (the output fanout) to verify if they are affected with the transient propagation until some primary circuit output. If the output of the latter gates connected doesn't change, the former has no vulnerability. In this work it is assumed that a transient pulse will have enough strength to go to the primary output of circuit.

The second counting reduction is a local component optimization that can be considered a electrical masking. It will analyze each gate internally and test the drain nodes that are still detected as transient vulnerable and will check a way to the gate output. If not detected in any way, the node will not be considered a vulnerable drain and will be discarded in transient counting. The algorithm is in Figure 1.

```
for each input vector
   simulate
   for each gate
      do (Initial Count)
   end for each
   optimize (Logical Masking)
   for each gate
      optimize (Electrical Masking)
   end for each
   reset
end for each
```

Fig. 1: Algorithm used for the tool.

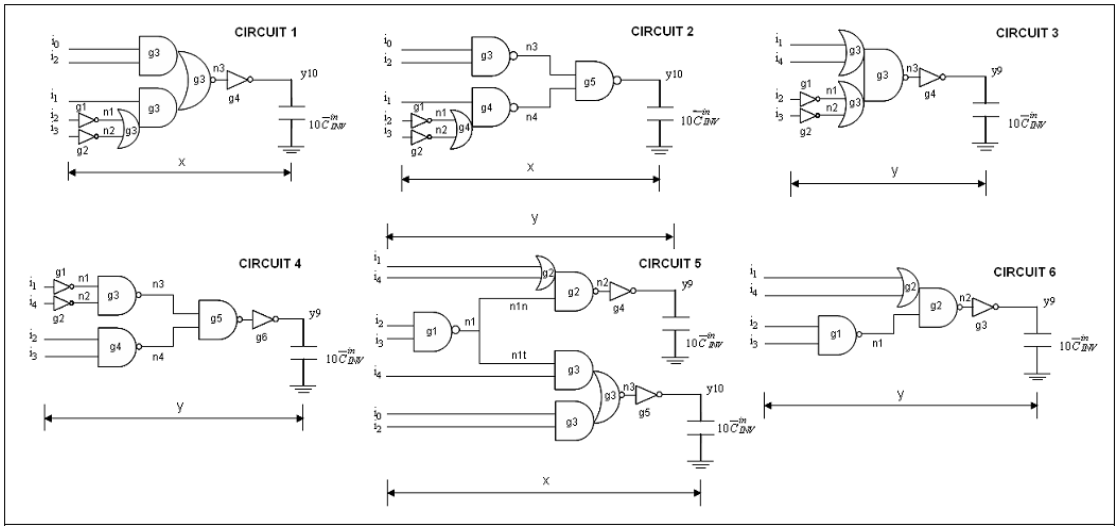


Fig. 2: Circuits for outputs y9 and y10 of circuit C17.

4. Results and Analysis

In this section, it will be shown the analysis of various implementations of ISCAS85 benchmark C17 which implements two logic functions: y9 and y10. The following functions are represented below (see Eq. 1 and 2) :

$$y9 = (i1 + i4) \cdot (i2' + i3') \quad (1)$$

$$y10 = i0 \cdot i2 + i1 \cdot (i2' + i3') \quad (2)$$

Fig. 2 depicts some possible implementations for these functions, in static CMOS. Circuits 1 and 2 implement function y10, circuits 3, 4, and 6 implement function y9, and circuit 5 implements both functions.

The purpose of this experiment is to evaluate the quantity of transient drains in circuit and analyze the impact of complexity of circuit in the number of vulnerable drains. In Table 1 we have the number of the vulnerable drains in the circuit and a metric of robustness of the circuit that is given by the sum of transient vulnerable drains.

Tab.1 - Results of transient drains in C17 implementations

Circuit (out)	Vulnerable drains in circuit	Sum of transient vulnerable drains (STVP)
1 (y10)	12	61
2 (y10)	16	69
3 (y9)	11	63
4 (y9)	18	99
5 (y9+y10)	17	238
6 (y9)	10	67

The purpose of this experiment is to evaluate the quantity of transient drains in circuit and analyze the impact of complexity of circuit in the number of vulnerable drains. In table 1 we have the number of the vulnerable drains in the circuit and a metric of robustness of the circuit that is given by the sum of transient vulnerable drains.

The groups of circuits to be compared are (1,2) and (3,4,6), characterizing y9 and y10 outputs. The level of complexity informed is about the more complex gate implemented in the circuit. The SVTP is an index that measures the overall vulnerability (less is better).

In circuit 1, it has a large gate (aoio12) and in circuit 2, it has a simpler gate (oai21). The metrics appoints the circuit 1 more robust with less vulnerable drains and lesser STVP.

In the circuit 3, it has a medium gate (oai22), as like circuit 6 (oai21), which is a little simpler and in circuit 4, only simple gates (nand2). The SVTP appoint circuits 3 and 6 similar, with a little advantage to circuit 3 that has a little more complex gate than circuit 6. The circuit 4, which has only simple gates has far more (almost 30% more) SVTP than the circuit 3 and 6, that uses a complex gate.

The circuit 5 is a different circuit that has 2 outputs, but has a SVTP almost the double of the circuit 1 and 3 that implements the y10 and y9 separately. Despite the number of vulnerable drains of circuit 5 is lesser than circuit 1 + circuit 3 (17 – 23), the logic sharing has a large impact on propagating at least one output the SEU, increasing the SVTP.

With this test, it can be observed that a circuit that shares logic has much more vulnerability than a circuit that has independent logic. Using complex ports to represent the logic functions minimizes vulnerability drains, which is desirable for a more robust circuit.

5. Conclusions

In this paper, a circuit analysis tool was shown that targets combinational circuits and makes a logic simulation and evaluates it´s robustness to SETs. The model used a switch level algorithm that can model important parameters and can process large circuits fast and with a high level of accuracy. The simulator makes exhaustive input evaluation to make a counting of vulnerable drains, considering shared drains and making optimizations. The logical masking analyses the logic ahead of each gate and the electrical masking analyses each node inside the gate, searching for a valid way to the gate output. Both optimizations improve the counting, lowering the drains affected in each vector.

6. Acknowledgements

Research partially funded by Nangate Inc. under a Nangate/UFRGS research agreement, by CNPq Brazilian funding agency, and by the European Community's Seventh Framework Programme under grant 248538-Synaptic.

7. References

[1] K. J. Hass, J. W. Gambles, B. Walker and M. Zampaglione. Mitigating single event upsets from combinational logic. In Proc of the 7th NASA Symposium on VLSI Design, 1998.

[2] M. S. Reorda and M. Violante, "Efficient Analysis of Single Event Transients," Journal of Systems Architecture, vol.50 no 5, pp.239–246,2004.

[3] L. Anghel and M. Nicolaidis, "Cost Reduction and Evaluation of a Temporary Faults Detecting Technique," Proc. Design, Automation and Test in Europe (DATE 00), IEEE CS Press, 2000, pp. 591-598.

[4] P. Dahlgren and P. Liden, "A Switch-Level Algorithm for Simulation of Transients in Combinational Logic, " Proc. IEEE 25th Int'l Symp. Fault-Tolerant Computing, pp.207-216, June 1995.

Design and Verification of a Layer-2 Ethernet MAC Search Engine and Frame Marker for a Gigabit Ethernet Switch

Jorge Tonfat, Gustavo Neuberger, Ricardo Reis
jltseclen@inf.ufrgs.br, neuberg@inf.ufrgs.br, reis@inf.ufrgs.br

Grupo de Microeletrônica (GME) – PGMICRO – UFRGS, Porto Alegre, RS, Brasil

Abstract

This work presents the design and verification of two main blocks of a gigabit Ethernet switch for a FPGA-based SoC implementation. The main function of the Layer-2 search engine is to forward Ethernet frames to their corresponding output ports. To accomplish this task the block stores the source MAC address from frames in a SRAM memory and associates it to one of the ports. This search engine uses a hashing scheme that has been proven to be effective in terms of performance and implementation cost. It can search an average of 8 million frames per second, which is enough to work at wire-speed rate in a four-port gigabit switch. The main challenge was to achieve wire-speed rate during the "learning" process using external SRAM memory. The frame marker is the mechanism to classify frames by means of data rate configured for each input port. It works in a credit-based system that uses buckets to track the rate of each port. This block will be part of a QoS mechanism. These two blocks are verified using System Verilog. A constrained-random stimulus is used in a layered-testbench environment with self-checking.

1. Introduction

Ethernet is the most popular layer-2 protocol (data link layer according to the OSI model). It is widely used in Local Area Networks or LANs and recently also in Metropolitan Area Networks or MANs. Its popularity is mainly due to the low cost and high performance characteristics and also its fast standardizations: 10 Mbits/s in 1983, 100 Mbits/s in 1995, 1 Gbit/s in 1998, and 10 Gbits/s in 2002.

At the beginning, LANs were designed using one shared communication channel. During late 80s and early 90s, two main factors changed the way LANs were designed [1]: the LAN topology that change to a structured wiring system using central hubs and the improvement of computing systems and applications, which exceed the capacity of shared LANs, limiting the overall performance.

These factors together with the advances in microelectronics technology, allow the development of "LAN switches" that use the wiring structure already installed to create a microsegmented network. These changes have the following advantages: first, the possibility to eliminate collisions, if the full-duplex operation mode is used and the fact that each device has dedicated bandwidth and independent data rate.

In the present work, we propose a layer-2 search engine and a frame marker for a gigabit Ethernet switch. This paper is organized as follow. In section 2 an introduction to the NetFPGA Platform is presented. In section 3 and 4 are presented the design of the L2 search engine and the frame marker respectively. Section 5 presents the verification methodology used for the two blocks. Section 6 shows the results for a FPGA implementation, and in section 7 the conclusions and future work are presented.

2. The NetFPGA Platform

The NetFPGA platform [2] was developed by a research group at Stanford to enable fast prototyping of networking hardware. It basically contains an FPGA, four 1GigE port and buffer memory. The core clock of the board runs at 125 MHz. NetFPGA offers a basic hardware modular structure implemented in the FPGA as shown in Fig. 1a. Frames inside the pipeline have their own header format as shown in Fig. 1b. New modules also can add more headers. This pipelined structure allows us to implemented specific functions on modules and integrate them quickly.

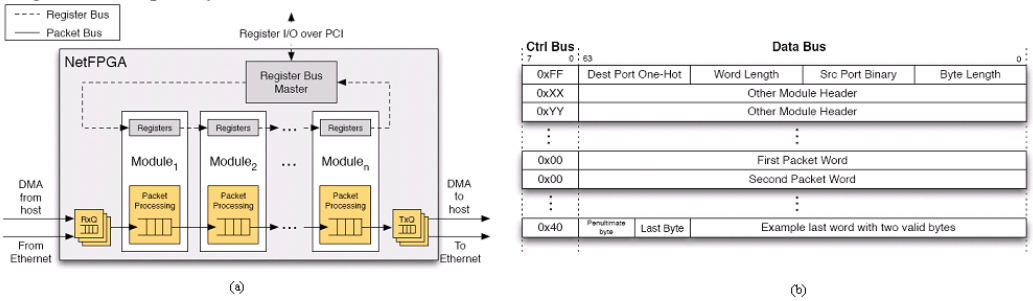


Fig. 1 – (a) The NetFPGA framework. [2] (b) The NetFPGA header format. [2]

3. L2 search engine architecture

The L2 search engine is the block that implements the main function in a gigabit Ethernet switch, the frame forwarding. It uses the MAC destination address (MACDA) and the MAC source address (MACSA) of the frame to forward them to their proper destination port. In order to achieve this task, it will need to create a table entry with the MACSA and the input port in a process called "learning". When the MACDA is not found on the table (miss), the frame will be sent to all ports except the source port. According to the IEEE 802.1D standard [3], it is necessary to age out all the entries that are not accessed for a certain amount of time; this is not a priority task and should not interrupt the main learning/forwarding process. VLAN tags should also be considered during the frame learning/forwarding to be compliant with the IEEE 802.1Q [4] standard.

Every frame needs at least two read accesses from the lookup table, one for the MACDA (forwarding) and another for the MACSA (learning). If the MACSA is not found or the source port associated is different, then a third access (write) should be necessary to update the input port number or create a new entry in the table.

Hashing is used as the method to store the 48-bit MAC addresses. Other methods such as CAMs were discarded due to the elevate cost and large power consumption. Using hashing methods leads to the possibility of collisions, which will reduce the performance of the system. That's why the hash function has to produce a relatively uniform distribution of the output values. The hash function selected is the CRC-CCITT ($x^{16}+x^{12}+x^5+1$) because according to the results in [5] CRC polynomials are excellent hashing functions.

Each entry is composed of a MAC address, the input port, the VLAN id, and three status bits. These status bits are: the valid bit, the static bit and the age bit. When a new MAC address is learned, the valid and the age bit are set. If this MAC address is founded later, the age bit is refreshed. In the aging process, the age bit of each valid entry will be cleared. If the age bit is not set, then the valid bit is cleared and the entry is aged out. The aging process will not modify the entries with the static bit set. The static bit denotes an entry programmed by software and is more commonly used with multicast MAC addresses.

This search engine should be able to support 4 GigE ports at wire-speed or to process 6 million frames per second in the worst case [1]. Considering the minimum frame size (64 bytes) and the inter-frame gap, a frame should be processed at least in 168 ns (672 ns for a single GigE port) or 21 clock cycles for a 125 MHz clock. To achieve this goal, this block needs to be tightly-coupled with the external SRAM memory. Since the SRAM is accessed by three different sources (the forwarding/learning module, the aging module and the external access through the register bus) an arbiter is needed. This arbiter is configured to serve in a WRR (Weighted Round Robin) manner giving the preference to the forwarding/learning module. The block diagram is presented in figure 2.

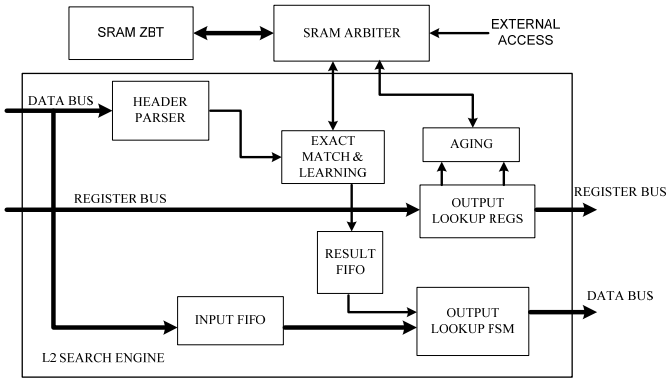


Fig 2 – L2 search engine block diagram.

The header parser module will extract the MACSA, MACDA and the VLAN tag ID from the frame and send this information to the Exact Match & Learning block. The frame will wait for the lookup using the input fifo as a buffer. Considering the worst case (MACSA not found), the module have a latency of 15 clock cycles.

4. Frame marker architecture

The frame marker is part of the QoS system of the switch. The rate control works in a credit-based system that uses buckets to track the rate of each port. The credits are continuously added in a bucket accordingly to the programmed bucket bit rate and they are decremented each time a frame ingresses at the port. If no frames arrive, credits are added until the programmed bucket size. The bucket can absorb bursts of frames limited to its size; when emptied the rate is limited to bucket bit rate. Up to this point, depending upon suppression method used, the frames can be dropped or a flow control mechanisms takes place forcing the port to reduce the rate making use of pause frames.

The frame marker will use a two-bucket credit system that makes possible to implements single rate Three Color Marker (srTCM) and two rate Three Color Marker (trTCM) metering and marking systems as described in RFC 2697 [7], RFC 2698 [8] and RFC 4115 [9]. The bucket system described in RFC2698 is slightly

different from RFC 2697 and RFC 4115. To get maximum flexibility and still meet all desired recommendations we must implement both approaches and make selection available through configuration registers.

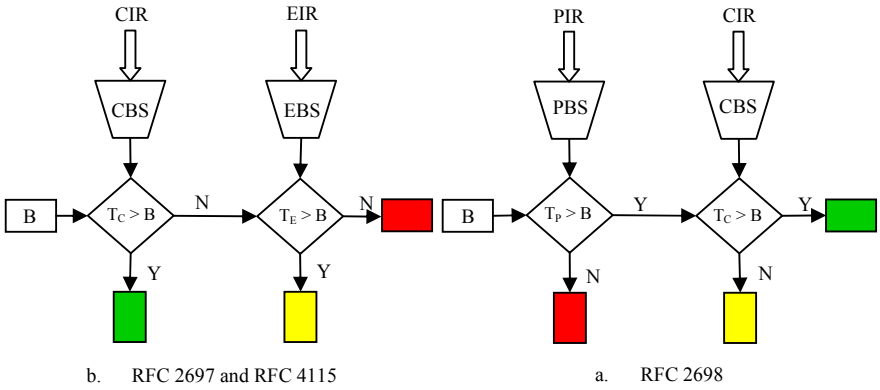


Fig. 3 – Flow diagram of the three marking systems.

Figure 3a. describes a particular RFC 2697 and RFC 4115 two-bucket system configuration where *B* represents the frame size, *CBS* is the Committed burst size, *EBS* is the Excess burst size, *CIR* is the Committed information rate, *EIR* is the exceed information rate, T_C is the current size of CBS, T_E is current size of EBS. Note that to change marker behavior between RFC 4115 and RFC 2697 is just necessary to make EIR = CIR. Figure 3b. describes RFC 2698 system where T_P is current size of PBS. This block will add a new module header (See Table 1). This header will be used to identify colored frames in future QoS classifier/marker.

Tab.1 – Frame marker header format.

Control Bus (8 bits)	**Data Bus (64 bits)**	
0xFE	Reserved [63:2]	Color Frame [1:0]

The bucket rate for each bucket is configured in *n* x 8kB/s steps with *n* = 0 ... 128000. Each bucket has its own 32 bits register to support this configuration. Another 32 bits register is used to program the bucket depth. The configuration interface also support an additional general register used to enable frame marker function per port and additional status registers to track classified/dropped packets per port.

The block diagram of the module is shown in figure 4; includes a fall-through input fifo with a depth of 4 positions. The block latency depends on the configuration of each port. If the color marker is disabled for all ports then the latency is zero. On the other hand, if the color marker is only disabled for some ports, then the latency is 1 cycle for the disabled ports and 3 cycles for the enabled ports.

The finite state machine has 5 states using the one-hot codification style. Depending on the configuration of each port, the state machine calculates the color of each frame. The state machine also detects if the frame already have a color and will change the color only when the calculated color elevate the frame status (towards to green).

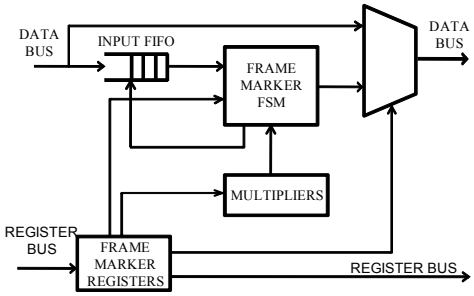


Fig. 4 – Frame Marker block diagram.

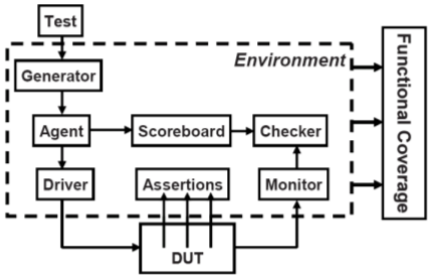


Fig. 5 – Full Testbench environment from [6].

5. Verification Methodology

For the verification stage, we use System Verilog and Modelsim to create the testbench environment. Since the two modules share the same I/O interface, the testbench interface for both blocks is the same. The testbench architecture is better explained in Figure 5.

For each block, a testcase have been done with particular constraints that will limit the random stimulus generation. With these constraints the generator will create a programmable amount of random frames that will be inserted in the DUT (Design under Test). The agent or transactor will take these frames and will transform

them into signals (bytes) and will send them through the interface (driver). The scoreboard will predict the expected result from each block and this result will be used by the checker to compare them with the received data from the DUT. During this process tens of bugs were founded in the design and corrected. It is always preferable that the testbench and the design are designed by different persons; this will add some redundancy to the interpretation process of the specification.

6. FPGA Implementation

The implementation results are shown in table 2. The FPGA used is the Xilinx Virtex-IIPRO XC2VP50-7. This is the device used in the NetFPGA card. For the complexity of the search engine, the maximum frequency obtained is lower. The frame marker is occupying more flip-flops due to the large amount of 32-bit registers needed to configure the marker algorithm for each of the input ports. Both blocks achieve the expected frequency of 125 MHz that will be used as the core clock for the entire switch. Further comparisons are needed but the lack of related works in the literature make this complicated. The L2 search engine architecture used here is shown in [10] but in this case is an ASIC implementation with embedded SRAM.

Tab.2 – Synthesis Results for a Xilinx XC2VP50 FPGA.

Circuit	4-input LUTs	Flip-flop slices	Frequency (MHz)
Frame marker	4774	3895	160.5
L2 search engine	3570	1966	142.9

7. Conclusions and Future work

Two modules for a gigabit Ethernet switch were presented. These two blocks are part of the design of a 4-port gigabit Ethernet port. Due to the modular nature of the NetFPGA structure, the design and verification of these two blocks were possible to be done separately without worrying about timing issues between them for instance. The verification stage is very important to find bugs that will only appear in some special cases. The random-constrained approach is more time-efficient to reach the coverage goal than other simpler methods such as direct-test.

The architecture presented in this work achieves the necessary throughput for a 4-port GigE. The next step is to work in order to improve the throughput of the L2 search engine. The main idea is to reduce the number of clock cycles needed to process a frame.

8. References

[1] Seifert, R.; Edwards, J. "The All-New Switch book: The Complete Guide to LAN Switching Technology", Second Edition. Wiley, 2008.

[2] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. "NetFPGA: reusable router architecture for experimental research". In: Proceedings of the PRESTO, pages 1–7, New York, NY, USA, 2008. ACM.

[3] IEEE std. 802.1D, "IEEE Standard for Local and Metropolitan Area Networks Media Access Control (MAC) Bridges," IEEE Std 802.1D-2004, pp.1-269, 2004

[4] IEEE std. 802.3Q, "IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks," IEEE Std 802.1Q-2005, vol., no., pp.0_1-285, 2006

[5] R. Jain, "A comparison of hashing schemes for address lookup in computer networks," IEEE Trans. On Communications, vol. 40, no. 3, pp. 1570-1573, Oct. 1992.

[6] Spear, C.; "SystemVerilog for Verification: A Guide to Learning the Testbench Language Features"; Springer; New York; 2008.

[7] Heinanen J. and Guerin R., "A single rate three color marker," IETF, RFC 2697, Sept. 1999.

[8] Heinanen J. and Guerin R., "A two rate three color marker," IETF, RFC 2698, Sept. 1999.

[9] Aboul-Magd O. and Rabie S., "A Differentiated Service Two-Rate, Three-Color Marker with Efficient Handling of in-Profile Traffic," IETF, RFC 4115, Jul. 2005.

[10] Lau, M.V, et.al. "Gigabit Ethernet switches using a shared buffer architecture," Communications Magazine, IEEE, vol.41, no.12, pp. 76- 84, Dec. 2003

Evaluating the Efficiency of Software-Only Techniques in Microprocessors

José Rodrigo Azambuja, Fernando Sousa, Lucas Rosa, João Almeida e Fernanda Lima Kastensmidt
{jrfazambuja, faacsousa, llrosa, jpvalmeida, fglima}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul (UFRGS) – Instituto de Informática

Abstract

This paper presents a detailed evaluation of the efficiency of software-only techniques to mitigate SEU and SET in microprocessors. A set of well-known rules is presented and automatically implemented to transform an unprotected program into a hardened one. SEU and SET are injected in all sensitive areas of a MIPS-based microprocessor architecture. The efficiency of each rule and a combination of them are tested. Experimental results show the inefficiency of the control-flow techniques in detecting the majority of SEU and SET faults. Three effects of the non-detected faults are explained. The conclusions can lead designers in developing more efficient techniques to detect these types of faults.

1. Introduction

The last-decade advances in the semiconductor industry have increased microprocessor performance exponentially. Most of this performance gain is due to smaller dimensions and low voltage transistors. However, the same technology that made possible all this progress also lowered the transistor reliability by reducing threshold voltage and tightening the noise margins [1, 2] and thus making them more susceptible to faults caused by energized particles [3]. As a consequence, high reliable applications demand fault-tolerant techniques capable of recovering the system from a fault with minimum implementation and performance overhead.

One of the major concerns is known as *soft error*, which is defined as a transient effect fault provoked by the interaction of energized particles with the PN junction in the silicon. This upset temporally charges or discharges nodes of the circuit, generating transient voltage pulses that can be interpreted as internal signals, thus provoking an erroneous result [4]. The most typical errors concerning soft errors are *single event upsets* (SEU), which are bit-flips in the sequential logic and *single event transients* (SET), which are transient voltage pulses in the combinatorial logic that can be registered by the sequential logic.

In areas where computer-based dependable systems are being introduced, the cost and development time are often major concerns. In such areas, highly efficient systems called systems-on-chip (SoC) are being used. SoC's are often designed using intellectual property (IP) cores and commercial off-the-shelf (COTS) microprocessors, which are only guaranteed to function correctly in normal environmental characteristics, while their behavior in the presence of soft errors is not guaranteed. Therefore, it is up to designers to harden their systems against soft errors. Fault tolerance by means of software techniques has been receiving a lot of attention on those systems, because they do not need any customization of the hardware.

Software implemented hardware fault tolerance (SIHFT) techniques exploit information, instruction and time redundancy to detect and even correct errors during the program flow. All these techniques use additional instructions in the code area to either recompute instructions or store and check suitable information in hardware structures. In the past years, tools have been implemented to automatically inject such instructions into C or assembly code, reducing significantly the costs. Nevertheless, the drawbacks of software only techniques are the impossibility to achieve complete fault coverage [5], usual high overhead in memory and degradation in performance. Memory increases due to the additional instructions and often memory duplication, while the performance degradation comes from the execution of redundant instruction [6, 7, 8].

In this paper, the authors implemented a set of software-only techniques to harden a matrix multiplication algorithm in order to point out the main vulnerable areas that are not mitigated by these techniques, more specifically the ones affecting the control-flow. Results can guide designers to improve efficiency and detection rates of soft errors mitigation techniques based on software.

The paper is organized as follows: Section 2 presents the case-study methodology and describe transformation rules to harden the program. Section 3 presents the fault injection campaign and results. Section 5 concludes the paper and presents future work.

2. The proposed case-study hardened program methodology

A set of transformation rules has been proposed in the literature. In [9], eight rules are proposed, divided in two groups: (1) aiming data-flow errors, such as data instruction replication [9, 10] and (2) aiming control-flow errors, such as Structural Integrity Checking [11], Control-Flow Checking by Software Signatures (CFCSS) [12], Control Flow Checking using Assertions (CCA) [13] and Enhanced Control Flow Checking using

Assertions (ECCA) [14]. The proposed techniques could achieve a full data-flow tolerance, concerning SEU's, being able to detect every fault affecting the data memory, which would lead the system to a wrong result. On the other hand, the control-flow techniques have not yet achieved full fault tolerance.

Most control-flow techniques divide the program into basic blocks by starting them in jump destination addresses and memory positions after branch instructions. The end of a basic block is on every jump instruction address and on the last instruction of the code.

ECCA extends CCA and is capable of detecting all the inter basic block control flow errors, but is neither able to detect intra-BB errors, nor faults that cause incorrect decision on a conditional branch. CFCSS is not able to detect errors if multiple BBs share the same BB destination address. In [15], several code transformation rules are presented, from variable and operation duplication to consistency checks.

Transformation rules have been proposed in the literature aiming to detect both data and control-flow errors. In [9], eight rules are proposed, while [16] used thirteen rules to harden a program. In this paper, we address six rules, divided into faults affecting the datapath and the controlpath.

2.1. Errors in the datapath

This group of rules aims at detecting the faults affecting the data, which comprises the whole path between memory elements, for example, the path between a variable stored in the memory, through the ALU, to the register bank. Every fault affecting these paths, the register bank or the memory should be protected with the following rules:

- Rule #1: every variable used in the program must be duplicated;
- Rule #2: every write operation performed on a variable must be performed on its replica;
- Rule #3: before each read on a variable, its value and its replica's value must be checked for consistency.

Fig. 1 illustrates the application of these rules to a program with 3 instructions. Instructions 1, 3, 7 and 8 are inserted due to rule #3, while instruction 3, 6 and 10 are inserted due to rules #1 and #2.

`ld r1, [r4]`	`1: bne r4, r4', error`
	`2: ld r1, [r4]`
	`3: ld r1, [r4 + offset]`

`add r1, r2, 1`	`4: bne r2, r2', error`
	`5: add r1, r3, 1`
	`6: add r1', r3, 1`

`st [r1], r2`	`7: bne r1, r1', error`
	`8: bne r2, r2', error`
	`9: st [r1], r2`
	`10: st [r1 + offset], r2`

Fig. 1 - Datapath rules

These techniques combined require more data, such as registers and memory addresses and, therefore, the microprocessor must have spare registers and the memory must have spare memory positions. This issue can also be solved by setting the compiler options to restrict the data section and the program to a given number of registers.

2.2. Errors in the controlpath

This second group of rules aims at protecting the program's flow. Faults affecting the controlpath usually cause erroneous jumps, such as an incorrect jump address or a bitflip in a non-jump instruction's opcode which becomes a jump instruction. To detect these errors, three rules are used in this paper:

- Rule #4: every branch instruction is replicated on both destination addresses;
- Rule #5: an unique identifier is associated to each basic block in the code;
- Rule #6: At the beginning of each basic block, a global variable is assigned with its unique identifier. On the end of the basic block, the unique identifier is checked with the global variable.

Branch instructions are more difficult to duplicate than non-branch instructions, since they have two possible paths, when the branch condition is true or false. When the condition is false, the branch can be simply replicated and added right after the original branch, because the injected instruction will be executed after the branch. When the condition is taken, the duplicated branch instruction must be inverted and inserted on the branch taken address.

beq r1, r2, 6	1: beq r1, r2, 5 **2: beq r1,r2, error**
add r2, r3, 1	3: add r2, r3, 1
	4: jmp 6 **5: bne r1,r2, error**
add r2, r3, 9 jmp end	6: add r2, r3, 9 7: jmp end

(a) Rule #4

beq r1, r2, 6	1: beq r1, r2, 6
add r2, r3, 1	**2: mv rX, signature 1** 3: add r2, r3, 1 **4: bne rX, signature 1, error**
add r2, r3, 9 st [r1], r2	**5: mv rX, signature 2** 6: add r2, r3, 9 7: st [r1], r2 **8: bne rX, signature 2, error**
jmp end	9: jmp end

(b) Rules #5 and #6

Fig. 2 - Rules #4, #5 and #6

Fig. 2(a) illustrates rule #4 applied to a simple program. For the branch if equal (BEQ) instruction, instructions 2, 4 and 5 must be inserted to replicate it, where instruction 5 is the inverted branch (branch if not equal). Instruction 4 is necessary to avoid false error alerts.

The role of rules #5 and #6 is to detect every erroneous jump in the code. They achieve this by inserting a unique identifier to the beginning of each basic block and checking its value on its end. Fig. 2(b) illustrates a program divided in two basic blocks (instructions 2-4 and 5-8). Instructions 2 and 5 are inserted to set the signature, while instructions 4 and 8 are inserted to compare the signatures with the global value.

3. Fault Injection Experimental Results

The chosen case-study microprocessor is a five-stage pipeline microprocessor based on the MIPS architecture with a reduced instruction set called miniMIPS [17]. In order to evaluate both the effectiveness and the feasibility of the presented approaches, an application based on 6x6 matrix multiplication algorithm is used.

A tool called PosCompiler was developed to automate the software transformation. The tool receives as input the program's binary code and therefore is compiler and language independent and is capable of implementing the presented rules, divided in 3 groups. The first group, called variables, implements rules #1, #2 and #3; group 2, called inverted branches, implements rule #4 and, finally, group 3, also known as signatures, implements rules #5 and #6. The user is allowed to combine the techniques in a graphical interface.

We generated through PosCompiler four hardened programs, implementing: (1) signatures, (2) variables, (3) inverted branches and (4) signatures, variables and inverted branches combined. Table 1 shows the original and modified program's execution time, code size and data size.

Tab. 1 – Original and hardened program's characteristics

Source	Original	(1)	(2)	(3)	(4)
Exec. Time (ms)	1.24	1.40	2.55	1.30	2.71
Code Size (byte)	2060	3500	4340	2580	6012
Data Size (byte)	524	532	1048	524	1056

First, thousands of faults were injected in the non-protected microprocessor, one by program execution. At the end of each execution, the results stored in memory were compared with the expected correct values. If the results matched, the fault was discarded. The amount of faults masked by the program is application related and it should not interfere with the analysis.

When total signal coverage was achieved and at least 3 faults per signal were detected we normalized the faults, varying from 3 to 5 faults per signal, and those faults build the test case list.

In order to achieve a detailed fault analysis, we sorted the faults by their source and effect on the system. We defined four groups of fault sources to inject SEU and SET types of faults: datapath, controlpath, register bank and ALU. We assumed the program and data memories are protected by Error Detection and Correction (EDAC) and therefore faults in the memories were not injected.

The fault effects were classified into 2 different groups: program data and program flow, according to the fault effect. To sort the faults among these groups, we continuously compared the Program Counter (PC) of a golden microprocessor with the PC of the faulty microprocessor. In case of a mismatch, the injected fault was classified as flow effect. If the PC matched with the golden's, the fault was classified as a data effect.

When transforming the program, new instructions were added and therefore the time in which the faults were injected changed. Since the injection time is not proportional to the total execution time, we mapped each fault locating the instruction where the fault was injected (by locating its new PC) and pipeline stage where the fault was manifested. Around 1% of the total number of faults could not be mapped and were changed by new faults.

Tab. 2 - (1) Signature, (2) Variables, (3) Inverted Branches and (4) All Techniques Combined.

	Source	# of Signals	Data	Hardened program versions (%)				Flow	Hardened program versions (%)			
				(1)	(2)	(3)	(4)		(1)	(2)	(3)	(4)
SET	Reg. Bank	2	9	-	100	-	100	1	-	100	-	100
	ALU	10	22	-	100	-	100	21	-	52.3	28.5	80.8
	Controlpath	29	90	-	100	-	100	46	2.17	23.9	2.17	23.9
	Datapath	8	37	-	100	-	100	3	-	100	-	100
	Total	49	158	-	100	-	100	71	1.4	36.7	9.8	45.1
SEU	Reg. Bank	36	18	-	100	-	100	18	-	100	5.5	100
	ALU	2	2	-	100	-	100	0	-	-	-	-
	Controlpath	126	63	-	100	-	100	56	1.8	17.8	1.7	19.6
	Datapath	20	19	-	100	-	100	1	-	-	100	100
	Total	184	102	-	100	-	100	75	1.3	37.3	4	40

Results show that the technique called variables (2) presented the highest detection rate among the three. It was capable of detecting all the faults injected in the register bank and the faults that caused errors on the data flow. The ALU was not completely protected because it also has control flow signals and some of these signals affected the program's flow. With 110% code size overhead, this technique could detect 77% overall. Technique (3) was able to complement technique (1) by detecting faults on branch instructions, mainly in the ALU, where most of the branch errors were found. By using technique (2) and (3), with 135% in code size overhead, these techniques combined could detect 79% overall.

The signatures (1), on the other hand, were responsible for detecting the faults affecting the program's flow, but it could not reach a high detection rate.

Combining all techniques, fault detection coverage reaches 80% with a code size increase of 192% and execution time increase of 118%. However, 20% of faults remain undetected.

4. Conclusion

In this paper we presented a set of rules based on software-only techniques to detect soft errors in microprocessors. A set of faults was built and a fault injection campaign was realized on the implemented techniques. Results showed that the variables and inverted branches presented a high detection rate, up to 77%, while the signatures showed results below expected.

We are currently working on improving the detection rates and decreasing the impact of the drawback on the signatures technique.

5. References

[1] R. C. Baumann. Soft errors in advanced semiconductor devices-part I: the three radiation sources. IEEE Transactions on Device and Materials Reliability, 1(1):17–22, March 2001.

[2] T. J. O'Gorman, J. M. Ross, A. H. Taber, J. F. Ziegler, H. P. Muhlfeld, I. C. J. Montrose, H. W. Curtis, and J. L. Walsh. Field testing for cosmic ray soft errors in semiconductor memories. In IBM Journal of Research and Development, pages 41–49, January 1996.

[3] International Technology Roadmap for Semiconductors: 2005 Edition, Chapter Design, 2005, pp. 6-7.

[4] P. E. Dodd, L. W. Massengill, "Basic Mechanism and Modeling of Single-Event Upset in Digital Microelectronics", IEEE Transactions on Nuclear Science, vol. 50, 2003, pp. 583-602.

[5] C. Bolchini, A. Miele, F. Salice, and D. Sciuto. A model of soft error effects in generic ip processors. In Proc. 20th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pages 334–342, 2005.

[6] Goloubeva O, Rebaudengo M, Sonza Reorda M, Violante M (2003) Soft-error detection using control flow assertions. In: Proceedings of the 18th IEEE international symposium on defect and fault tolerance in VLSI systems—DFT 2003, November 2003, pp 581–588.

[7] Huang KH, Abraham JA (1984) Algorithm-based fault tolerance for matrix operations. IEEE Trans Comput 33:518–528 (Dec).

[8] Oh N, Shirvani PP, McCluskey EJ (2002) Control flow Checking by Software Signatures. IEEE Trans Reliab 51(2):111–112 (Mar).

[9] M. Rebaudengo, M. Sonza Reorda, M. Torchiano, and M. Violante. Soft-error detection through software fault-tolerance techniques. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pages 210–218, 1999.

[10] C. Bolchini, L. Pomante, F. Salice, and D. Sciuto. Reliable system specification for self-checking datapaths. In Proc. of the Conf. on Design, Automation and Test in Europe, pages 1278–1283, Washington, DC, USA, 2005. IEEE Computer Society.

[11] D.J. Lu, "Watchdog processors and structural integrity checking," IEEE Trans. on Computers, Vol. C-31, Issue 7, July 1982, pp. 681-685.

[12] N. Oh, P.P. Shirvani, and E.J. McCluskey, "Control-flow checking by software signatures," IEEE Trans. on Reliability, Vol. 51, Issue 1, March 2002, pp. 111-122.

[13] L.D. Mcfearin and V.S.S. Nair, "Control-flow checking using assertions," Proc. of IFIP International Working Conference Dependable Computing for Critical Applications (DCCA-05), Urbana-Champaign, IL, USA, September 1995.

[14] Z. Alkhalifa, V.S.S. Nair, N. Krishnamurthy and J.A. Abraham, "Design and evaluation of system-level checks for on-line control flow error detection," IEEE Trans. on Parallel and Distributed Systems, Vol. 10, Issue 6, June 1999, pp. 627 – 641.

[15] Cheynet P, Nicolescu B, Velazco R, Rebaudengo M, Sonza Reorda M, Violante M (2000) Experimentally evaluating na automatic approach for generating safety-critical software with respect to transient errors. IEEE Trans Nucl Sci 47(6 part 3): 2231–2236 (Dec).

[16] B. Nicolescu and R. Velazco, "Detecting soft errors by a purely software approach: method, tools and experimental results", Proceedings of the Design,Automation and Test Europe Conference and Exhibition, 2003.

[17] L. M. O. S. S. Hangout, S. Jan. The minimips project. available online at http://www.opencores.org/projects.cgi/web/minimips/overview 2009.

Exploring Embedded Software Efficiency through Code Refactoring

[1]Wellisson G. P. da Silva, [1]Lisane Brisolara, [2]Ulisses B. Corrêa, [2]Luigi Carro

wguilhermino@gmail.com, lisane.brisolara@ufpel.edu.br, ubcorrea@inf.ufrgs.br, carro@inf.ufrgs.br

[1]Instituto de Física e Matemática – Departamento de Informática – Universidade Federal de Pelotas (UFPEL) - Pelotas – RS – Brazil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS) Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

Abstract

Nowadays, power-efficient and high performance systems are a constant demand on the electronic industry. The power and energy consumption are important issues for the design of portable embedded systems, since users want to do more without recharging the device's battery. Although these issues are more directly related to hardware aspects, the way how the software interacts to the hardware resources has an impact on system power/energy efficiency. The increasing complexity of systems and the use of object-oriented languages can increase power and energy consumption as well as decrease performance. In this scenario, embedded software designers are searching for strategies to produce efficient software. This paper analyzes how the inline method refactoring, a software optimization technique, can affect the performance and energy of embedded software written in Java.

1. Introduction

Usually, in embedded software development, hard constraints should be considered, such as high performance and timing requirements, small memory footprint, or low power and low energy requirements [1]. Low power and low energy consumption is a crucial factor for embedded systems, mainly for portable and handheld devices based on batteries. Nowadays, several techniques are used to manage the power/energy consumption of electronic systems, being applied to the hardware [2]. However, the way the software interacts with system power-consuming resources, and the code efficiency has also impact on power and energy consumption as well as on performance.

To handle the increasing complexity of embedded software and the hard time-to-market requirements, the use of object-oriented languages became more important mainly due its modular and reusable code. However, object-oriented languages can introduce penalties to system power and energy consumption and performance [3]. In this scenario, embedded software designers should handle the software complexity and produce efficient software at the same time.

Refactoring is a technique of software engineering in which the code is modified in order to improve its readability and maintainability without changing its computation [4]. A common refactoring method is "method inline", which is also a known compiler optimization technique. This paper analyzes through experiments the impact on the system performance and power consumption achieved by method inline when it is applied to Java codes. This study can help designers to understand how object-oriented practices affect properties like performance and power consumption.

Section 2 gives the background about refactoring and how it can impact power consumption. Section 3 discusses the methodology and target platform used in the experiment. Experimental results are analyzed in Section 4 and Section 5 concludes this work and presents future work.

2. Background

There are three main sources of power consumption in embedded systems [3], which are: processor power consumption, due to the processor activity, memory power consumption, for accessing data and instructions in memory, and the power consumption to connect the processor and memories. All these operations should be taken in account, when developing embedded software. Moreover, some software engineering practices can directly impact system power consumption and performance, like code refactoring.

Refactoring is a process of modifying the code to make it easier to understand and modify without changing its computation as defined in [4]. These code changes can also affect the system performance and power consumption, since it modifies the instructions that will be used, and thus, introducing or removing some overhead which change the behavior of the system even if it do not change its computation. Examples of

changes that can be done are Extract Method – which consists in the creation of a method to represent duplicate code, or to simplify long methods – or Inline Method – which is the opposite of the Extract Method, once it exchanges a method call for its body. Inline is not recommended by the software engineering best practices, because it can make the code hard to understand and so decrease code maintainability.

Although, the Inline Method is expected to increase performance since it removes the overhead of a method call – in Java Virtual Machine each method call creates a frame with its own local variables, operand stack, and reference to the runtime constant pool of the class [5]. This work has as objective to analyze how method inline affects performance and power consumption for Java codes.

3. Methodology and Target Platform

Through dynamic profiling of application code, information about the number of method calls is obtained. This information is used to choose method candidates for refactoring. To increase gains achieved with the reduction of method invocations, the method inline is applied for the methods more called. The method inlines were incrementally done, i.e., in the code original version, the most frequently called method was inlined (generating a version nominated *Iteration1*), after that the second most frequently called method was inlined in the *Iteration1* version (generating the *Iteration2* version), and so on.

In order to obtain performance and energy consumption for these several Java code versions, an estimation tool called Desejos was used [6]. As embedded platform, we adopted the FemtoJava [7] processor. This processor is a stack based Java Virtual Machine implementation, executing Java bytecodes natively. We adopted FemtoJava and DESEJOS due to their use of Java, a language that is gaining attention on the embedded community.

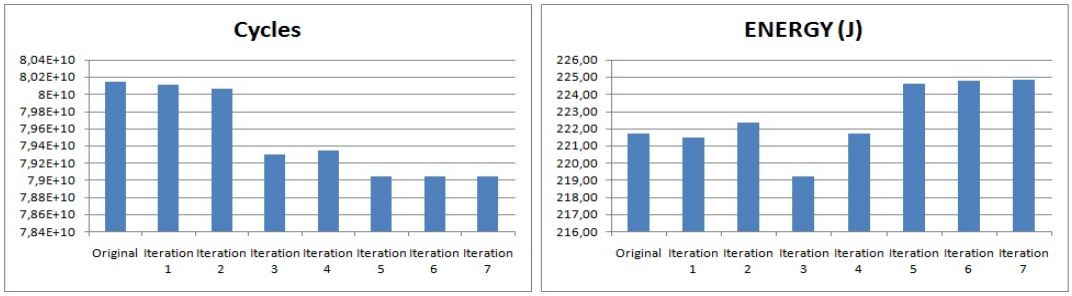
The FemtoJava processor implements a stack machine compatible with Java Virtual Machine (JVM) specification and that is able to execute Java code in hardware. Two different versions of the FemtoJava processor are available: multicycle and pipeline. In this experiment, we adopted the multicycle version that is targeted to low power embedded applications.

4. Experiments

As case study, an Mpeg layer-3 audio decoder (float-point centric) from the Spec jvm2008 benchmark set [8] was used. This benchmark set contains several real life applications and was designed to evaluate the performance of a Java Virtual Machine and underlying hardware.

Execution cycles and energy consumption estimation have been obtained for each version of Java code. Figure 1(a) and 1 (b) illustrates the obtained results for performance (in cycles) and energy consumption (in Joule), respectively.

The experimental results show that a reduction on cycles needed to run the application was achieved, as expected, since cycles from creating frames for each method call were eliminated. However, the energy consumption results were not as expected. After the third iteration, which has a greater gain from its antecessors, all remaining versions have higher consumption and the last ones have a consumption worse than the original code (without any inline applied).



(a) Performance results (b) Energy consumption results

Fig 1. Experimental results: Performance and energy consumption estimates

To better analyze the achieved results for energy consumption, we have analyzed the Java bytecode and have obtained the instruction histogram for the code versions in which the results are not the expected. Summarized results of these histograms can be observed in Table 1 and Table 2. Table 1 shows the instructions that presented different number of calls between the Iterations two and three, whose versions present the lowest energy consumption (see Fig. 1b). A reduction in the number of *iload_1* instructions can be observed in the Iteration3 results in Table 1, as well as an equivalent increase in the number of *iload* instructions. These instructions require one more access to memory than the *iload_<n>* instructions (like *iload_0, ..., iload_3*). This extra memory access is used to load the index of which variable in the pool will be popped on the JVM

operand stack, unlike the simpler *iload_<n>* instructions that has this address implicitly defined. Moreover, these results show reductions in the number of *invokevirtual* and *ireturn* instructions that represents an invocation of a class instance method in Java and the return instruction, respectively. These reductions were expected, since the inline removed method calls. Besides, a great reduction in the number of the *aload_0* instructions can be observed in Table 1.

Tab 1. Main instructions used for the Iteration 2 and 3

Instruction	Iteration 2	Iteration 3	Difference
iload	2256402932	2284795276	**28392344**
iload_1	117059899	88667555	**-28392344**
aload_0	1058245488	916283768	**-141961720**
istore	372594809	400987153	**28392344**
istore_1	28458885	66541	**-28392344**
ireturn	41305600	12913256	**-28392344**
invokevirtual	37981918	9589574	**-28392344**

To observe the increasing on energy provoked by inline, the difference between the third and fourth iterations is analyzed. Table 2 shows the number of used instructions for each code version (Iteration 3 and Iteration 4) and the difference between these numbers. The results show a great increase in the number of *iload* instructions and also a decrease in *aload_1* instructions, and yet an increase in the number of *getfield* instructions. Moreover, Table 2 shows the effect of the inline method, where the number of method calls is decreased, provoking a reduction in the number of *invokevirtual* and *return* instructions. In addition to that, the inline affected the number of method´s local variables causing a changing of position in the variable pool. This changing can be noted in the variation of *istore* and *istore_1* instructions. With the method scope increasing the variable that occupied the second position in the variable pool went to a new position after the fourth position, generating the necessity of an *istore* instruction.

This increase in *iload* instructions explains the increase in power consumption (and consequently increasing energy consumption) since this instruction is more complex and need more accesses to the memory to be executed when compared to the *iload_<n>* instructions. The increase of *iload* is a consequence of the inline method, which inserted more local variables to the method and the instructions *iload_0, iload_1, iload_2* and *iload_3* cannot be used.

Tab 2. Main instructions used for the Iteration 3 and 4

Instruction	Iteration 3	Iteration 4	Difference
iload	2284795276	2383364176	**98568900**
iload_2	289277504	190708604	**-98568900**
aload_0	916283768	939206768	**22923000**
aload_1	203437622	178986422	**-24451200**
istore	400987153	401751253	**764100**
istore_2	16634671	15870571	**-764100**
return	6336716	5572616	**-764100**
getfield	1169270767	1241096167	**71825400**
invokevirtual	9589574	8825474	**-764100**

5. Conclusions

This paper studies the impact on the embedded system performance and energy consumption achieved by the method inline, a well known refactoring method, when applied to Java codes. The Inline Method was expected to increase performance and decrease energy consumption since it would reduce the number of cycles and memory accesses to create a frame in Java Virtual Machine in every method call. However, the experimental results have shown that if the method in which the inline is applied become much complex, with many local variables, to address all its variables in the class stack can require instructions exchanging. Sometimes, simpler instruction like *iload_0*, for instance, which are optimized to access positions in memory, are exchanged by the *iload* instruction, which is more complex and needs one more access to the memory and consequently costs more in terms of energy.

Then when applying inline in a method, the complexity of the method should be taken in account. It is because the reduction of a method call does not result in a gain when the method has so many variables to be addressed in the stack, as the experiments shown.

As future work, we plan to perform experiments using other applications as case studies, and explore other refactoring methods.

6. References

[1] Graaf, B.; Lormans, M.; Toetenel, H. "Embedded Software Engineering: the State of the Practice". IEEE Software, v. 20, n. 6, p. 61- 69, Nov. – Dec. 2003.

[2] Saxe, E. "Power Efficient Software". Communication of the ACM. v. 53, n. 02, Feb. 2010.

[3] Chatzigeorgiou, A. and Stephanides, G. "Evaluating Performance and Power of Object-Oriented vs. Procedural Programming in Embedded Processors", In: 7th International Conference on Reliable Software Technologies, Ada-Europe 2002, Vienna, Austria, June 17-21, 2002.

[4] Fowler, M. "Refactoring: Improving the Design of Existing Code", Addison Wesley, 14th printing, 2004.

[5] Sun Microsystems, Inc. "The JavaTM Virtual Machine Specification", http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html, March, 1999.

[6] Mattos, J.C.B., Carro, L. "Object and Method Exploration for Embedded Systems Applications. In Proc. of the Symposium on Integrated Circuits and Systems Design", In: 20th Symposium on Integrated Circuits and System Design, SBCCI, Rio de Janeiro, Brazil, 2007. New York: ACM Press, 2007.

[7] Ito, S. A., Carro, L., & Jacobi, R. P. "Making java work for microcontroller applications". IEEE Design & Test of Computers, v.18, n. 5, 100–110, 2001.

[8] SPECjvm2008 (Java Virtual Machine Benchmark), http://www.spec.org/jvm2008.

A Front-End Development Environment for the Brazilian Digital Television System

Jônatas Romani Rech, Leonardo Faganello, Altamiro Amadeu Susin
{jrrech, lrfaganello}@inf.ufrgs.br, {altamiro.susin}@ufrgs.br

Universidade Federal do Rio Grande do Sul

Abstract

The objective of this paper is to describe the research and implementation process of a development environment for open-source CPUs, such as the OpenCores Plasma processor. This CPU will integrate the SoC- SBTVD (Access terminal for the Brazilian digital television system) as a front-end module, which primary role in the SoC is to demultiplex the data stream from a signal source (e.g. an antenna).

1. Introduction

The SoC – SBTVD project main objective is to design an implementation of a fully-functional set-top box (or an access terminal) for the Brazilian digital television system on a single silicon wafer, reducing costs and improving the system performance and reliability.

The system consists in an aggregation of several modules that perform different tasks, each one crucial to the system as a whole. Figure 1 pictures a primary diagram of the system architecture:

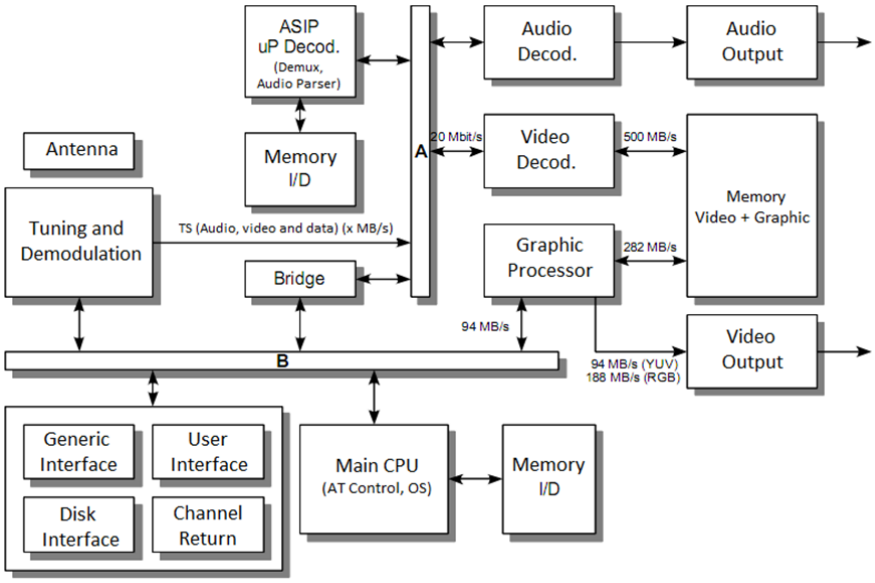


Fig.1 – SoC Architecture.

The main goal of this research is to provide the SoC – SBTVD project a front-end module, capable of demultiplexing data come directly from the MPEG-TS (transport stream) and routing the different streams into the correct processing units. The front-end module must also master a bus that connects the processing units to assure the correct stream routing. These tasks can be accomplished by a small-sized CPU, such as the OpenCores Plasma, which will be approached in this text. Since there's not a well defined and validated development environment for Plasma on Linux platforms, this document will also describe the process of combining open-source tools to implement such environment.

2. Implementation

2.1. Choosing processor and bus

Processors can be implemented in two ways, *hard* or *soft-processors*. Those of the *soft* kind are IP (Intellectual Property) cores, implemented with logic primitives of a reconfigurable technology, as FPGA. They are developed using a hardware description language, which provides them a high flexibility level. This flexibility is vital when the need is for performance, for area minimization or even for something that is a balance between both. Although the final goal of this project is to have the whole access terminal implemented

directly in a silicon piece, our current concern is the FPGA prototyping, what makes us lean towards the soft-processor alternative.

Tab. 1: comparative table of Plasma and ARM processors.

	Plasma	**ARM**
Developer	OpenCores	ARM
Implementation Aim	FPGA	ASIC-otimized
Open-source	Yes	no
Ditribution format	Soft module	Hard and semi-hard modules
Type of distribution	Open-source	Open-source License
Bus Interface	None	AMBA
Compiler	GCC	GCC

So far, the Plasma CPU has met these criteria, and was chosen to integrate the SoC with the role of demultiplexing data and routing streams between processing units. Eventually, the additional function of parsing audio streams may be implemented. Table 1 compares Plasma CPU and ARM processors when it comes to implementation technology, open-source/proprietary descriptions and soft/hard module distribution.

Considering the bus, the best choice available is the AMBA (Advanced Microcontroller Bus Architecture) standard, which compiles to several processor families (ARM and MIPS), and also to many peripherals developed by our research laboratory[1], like the H.264 Video Decoder.

2.2. Plasma Processor

The Plasma Processor, developed by Steve Rhoads, is a small 32-bit RISC processor and it implements almost all of the MIPS I instructions, except for the unaligned load and store, since these operations were patented. This means that, when loading or storing 32-bit values, the memory address must be on a 32-bit aligned address. This limitation, however, is easily avoided by the compiler, because the GCC MIPS compiler seldom generates unaligned accesses to memory. Another limitation of the Plasma processor is that exceptions must not be placed immediately after a branch instruction [2]. Synthesis Data of Plasma is available on Table 2.

Plasma operates in Big Endian mode by default, but it can be easily changed to Little Endian. It also can use a 2 or 3 stage pipeline.

Tab. 2: Synthesis Data of Plasma Processor.

Xilinx Virtex 2 Pro VP30			
Maximum Frequency	72.127MHz		
	Used	**Available**	**%**
Flip-Flops	424	27392	1.55
LUT-4	3219	27392	11.75
Slices	1660	13696	12.12
Brams	4	136	2.94

Further development for Plasma required a C cross-compiler, aside to additional tools like a linker, an assembler and a debugger. MIPS Technologies [3] has made available a ready-to-go toolchain for MIPS processors, which fits very well to our needs. As we intended to compile C source codes using the standard C library, an in-between layer was needed to handle syscalls generated by those functions. The Plasma downloadable pack includes an RTOS (Real Time Operating System) that supplies stubs for such syscalls. The RTOS source code must be compiled and linked together with the programs we intend to run, and so does the source code of a small C library that supplies basic functions (I/O, memory allocation/dump, string handling, etc).

The toolchain currently in use features Binutils 2.19.51 (linker, assembler, dumper), GCC 4.4.1 (compiler) and GDB 7.0.5 (debugger).

2.3. AMBA Bus

As mentioned before, the great advantage in using the AMBA standard is being able to integrate many peripherals developed by our research laboratory. Also, the AMBA protocol is an open standard.

This project will be using AMBA 3 (AHB-Lite) specification because it's a simple implementation of the standard with only a single master. It supports burst transfers from 8 to 1024 bytes, depending on the bus width.

The AMBA protocol uses an Address Phase and a Data Phase. On the first cycle (Address Phase), the Memory Address is defined. On the second cycle (Data Phase) the data is transferred. While the data is being stored or loaded, the next memory address is defined by the CPU. Figure 2 illustrate the two phases of the transfer.

The "HCLK" signal is the clock signal. "HADDR" is the memory address, "HRDATA" is the loaded data and "HWDATA" is the stored data. The "HREADY" signal is used to determine the duration of the transfer, used on burst transfers [4].

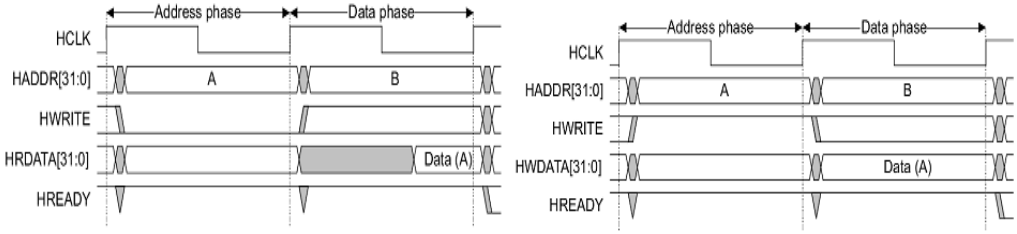


Fig.2 – AMBA transfer. Load and Store operations respectively.

3. Results

3.1. Tools utilized in simulations and tests

The Plasma VHDL project was built on Xilinx ISE 10.1; simulations were run on Modelsim XE III 6.4b and ISE Simulator. In order to test the compiler tools, we must compile the source code, link the object files and insert the binaries into the RAM module of the Plasma ISE project. This is done by a couple of tools provided in the Plasma package, "convert_bin" and "ram_toimage". The UART output is captured into a file named "output.txt". We will watch this file for the text output of our C programs.

3.2. Toolchain Test

This section describes the tests of the compiler tools in a few steps:

- Writing a C source-code (Figure 3);
- Using the tools to compile/link the sources and add the binaries to the RAM VHDL file;
- Run the binaries in a ISim simulation and observe the "output.txt" text file (Figure 4).

```
#include "rtos.h"
#include "plasma.h"

void MainThread(void *Arg)
{
        printf(" Hello world from test program over RTOS!\n");
        return;
}
```

Fig. 3: C language test program.

```
 1000001c 27bd50c8    ADDIU $29 $29 $10 $03 50c8
 10000020 aca00000       SW $05 $00 $00 $00 0000
1=Debug 2=Trace 3=Step 4=BreakPt 5=Go 6=Memory 7=Watch 8=Jump 9=Quit>
Starting RTOS
SimIsr
 Hello world from test program over RTOS!
```

Figure 4: "output.txt" file, containing the text output of the source code running on Plasma.

3.3. Amba and Plasma compatibility test

The Plasma processor is described in VHDL and its Bus Interface is very flexible and adaptable. However, the temporization of the AMBA (Address and Data phases) needed to be compatible with Plasma. In order to verify its compatibility it was made a simple test: a project containing only the Plasma CPU core, memory and a simple GPIO module previously developed and tested with Arm Processor using the AMBA protocol. The test is well illustrated by the Figure 5.

To achieve this compatibility, it was created an area on memory map, on CPU control, that allows switching the access from regular memory to the GPIO module. From 0x00000000 to 0xffffffff it accesses the Internal RAM. From 0x10000000 to 0x1fffffff it accesses External RAM. From 0x30000000 to 0x3fffffff it accesses the GPIO module.

After changing the original Memory Map, the signals between CPU Core and Memory were duplicated to reach the new GPIO module. Those signals were "address", "address_next", "byte_we", and "byte_we_next".

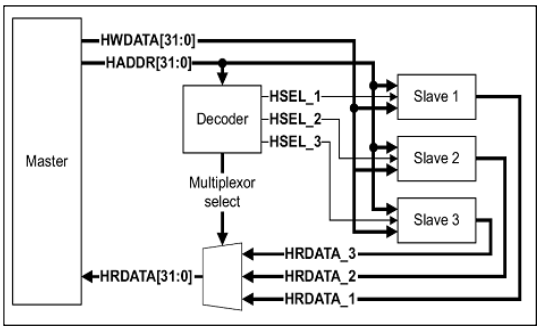


Fig.5 – An example of the test, where the Slaves 1 and 2 are the memory and the GPIO module.

Using the signals "address_next" and "byte_we_next" (byte write enable - which enables storing data in memory) from the CPU core it was possible to anticipate the address and create an Address Phase. In the next cycle, the memory is accessed using the "address" and "byte_we" signals (previously "address_next" and "byte_we_next") while the current "address_next" is used to anticipate the next memory access, creating the second Address Phase. This simple test was enough to verify the compatibility of the Plasma Processor with the AMBA Protocol. Figure 6 shows an access to the GPIO module, using AMBA standard (HCLK, HADDR, HRDATA, HWDATA and HWRITE).

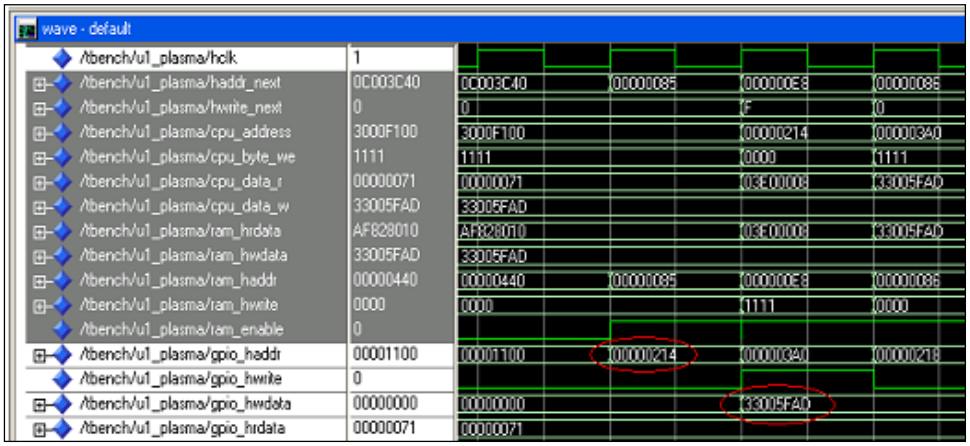


Fig.6 – Plasma accessing the GPIO module, with Address and Data phases.

4. Conclusion

This paper describes the implementation of a development environment for a front-end module in the Brazilian digital television system, so as the integration of a small open-source processor with the AMBA bus standard. Our future works will focus on the FPGA programming and performance measuring having Plasma and AMBA working together, as well as the other peripherals such as the video and audio decoders. A software h.264 demux has been developed in the meantime of this work, being a good test subject for this still under-construction environment. Later optimizations of this software for Plasma are also a goal we expect to achieve.

5. References

[1] LaPSI – Laboratório de Processamento de Sinais. Departamento de Engenharia Elétrica – DELET. Universidade Federal do Rio Grande do Sul – UFRGS.

[2] MIPS TECHNOLOGIES *MIPS Technologies – MIPS Everywhere*, 2006-2010. [Online]. Available: http://www.mips.com. [Accessed: Feb. 20, 2010].

[3] S. Rhoads, "Plasma – most MIPS I(TM) opcodes", OpenCores. Sep. 25, 2001. [Online]. Available: http://www.opencores.org/project,plasma. [Accessed: Dez. 10, 2009].

[4] ARM, "AMBA 3 AHB-Lite Protocol Specification" *ARM – The Architecture For The Digital World*, ARM IHI 0033A, 2006. [PDF]. Available: http://www.arm.com. [Accessed: Feb. 23, 2010].

Author Index