Designing Programmable Platforms: From ASIC to ASIP

MPSoC 2005

Heinrich Meyr

CoWare Inc., San Jose
and
Integrated Signal Processing Systems (ISS),
Aachen University of Technology,
Germany
Agenda

- Facts & Conclusions
- Heterogeneous MPSoC
  - Energy Efficiency vs. Flexibility
  - How to explore the Design Space?
- ASIP Design
- Economics of SoC Development
- Conclusions
Facts & Conclusion
Core Proposition

- ASIP based Platforms
  (heterogenous MPSoC)

Results (Design Productivity)

- 1974 Transistor-level design – Calma, Computervision
- 1984 Gate-level design – Daisy, Mentor, Valid
- 1994 Register-transfer level Cadence, Synopsys
- 2004 ???

Effort (EDA tools investment)
Agenda

- Facts & Conclusions
- Heterogeneous MPSoC
  - Energy Efficiency vs. Flexibility
  - How to explore the Design Space?
- ASIP Design
- Economics of SoC Development
- Conclusions
Trade-off between Flexibility and Energy Efficiency
Architectural Objectives

Need more MOPS/Watt and MOPS/mm² to minimize the global performance measure for battery driven devices

Energy / decoded Bit = (Joule/Bit)
Computational Efficiency vs. Flexibility

Source: T.Noll, RWTH Aachen
Enabling MP-SoC Design
System Level Tools I: Application & IP Creation

algorithmic exploration

system application design

algorithm domain

Language

• Matlab
• SPW
• System Studio

Architecture Description Language

block specification

• LISATek Processor Synthesis
• ConvergenSC Buscompiler

High-level IP block design

• RTL Synthesis

micro architecture domain

block implementation
System Level Tools I: Application & IP Creation

- Matlab
- SPW
- System Studio application design

- MP-SoC Intermediate Representation
  - ConvergenSC Platform Creator

- High-level IP block design
  - RTL Synthesis

- SystemC Transaction Level Modeling
  - virtual prototype

- System Level Tools II: MP-SoC Platform Design
  - Matlab
  - SPW
  - System Studio application design

- Algorithmic exploration

- Architecture Description Language
  - block specification

- micro architecture domain
  - block implementation

- SystemC
  - abstract architecture
Agenda

- Facts & Conclusions

- Heterogeneous MPSoC
  - Energy Efficiency vs. Flexibility
  - How to explore the Design Space?

- ASIP Design

- Economics of SoC Development

- Conclusions
Processor Design Space

- Instruction-Set Design
  - Compiler Design

- Micro Architecture Design

- RTL Design
  - RTL ISS Co-verification

Optimal design requires powerful tools and automation!

MESCAL 2: Inclusively identify the architectural space

- System Integration
  - Embedded Software Simulation

- Instruction-Set Design
  - Compiler Design

Optimal design requires powerful tools and automation!
The purpose of an architecture description language (e.g., LISA) is:

- To allow for an iterative design to efficiently explore architecture alternatives.
- To jointly design “Architecture – Compiler” and on-chip communication.
- To automatically generate hardware (path to implementation).
- To automatically generate tools, such as Assembler, Linker, Compiler, Simulator, co-simulation interfaces.

From a single model at various level of temporal and spatial abstraction.

MESCAL 3: Efficiently describe the ASIP
LISA 2.0 - Abstraction Levels

- architecture
  - + IRQ, etc.
  - + Pipelines
  - Functional units, Registers, Memories
  - Pseudo Resources (e.g. c-variables)

- details
  - Pseudo Instructions
  - Processor Instructions
  - Cycles
  - Phases

- accuracy
  - instruction accurate model
  - cycle accurate model
  - phase accurate model

- time
  - very detailed
Rapid modeling and re-targetable simulation allows for:

- **MESCAL 3:** Efficiently describe and evaluate the ASIP
- **MESCAL 5:** Successfully deploy the ASIP

Function and instruction level profiling reveals hot-spots:

- Special purpose instructions

**Describe/Adopt Processor Model**

**Generate Tools**

- Describe/Adopt Processor Model
- Generate Tools

**LaTeX**

<table>
<thead>
<tr>
<th>Empty Model</th>
<th>RISC Sample</th>
<th>VLIW Sample</th>
<th>DSP Sample</th>
<th>FFT Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISATek IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Custom Processor Model**
(LISA 2.0 language)
Current Work

Target Architecture
LISA Description
LISA Compiler

C-Compiler
Assembler
Linker
Simulator
Model Verification & Evaluation
Evaluation Results
Profile Information, Application Performance

HDL Generator
SystemC, VHDL, Verilog Output
Gate Level Synthesis
Evaluation Results

MESCAL 3:
.....evaluate the ASIP

• Instruction Set Synthesis
• Memory architecture
• Verification

EXPLORATION

IMPLEMENTATION
A Novel Approach for Flexible and Consistent ADL-driven ASIP Design

Gunnar Braun
Achim Nohl
CoWare, Inc
DAC Booth #1844
www.CoWare.com

Weihua Sheng, Jianjiang Ceng, Manuel Hohenauer, Hanno Scharwächter, Rainer Leupers, Heinrich Meyr
Integrated Signal Processing Systems (ISS)
Aachen University of Technology
Germany
Architecture Description Languages (ADL)

- Automatic generation of Software Toolkit (Compiler, Assembler, Linker, IS-Simulator)
- Architecture Exploration
- SystemC models, RTL code, verification tools, ... 

Challenges:

- Different tools need different information
- Unambiguous, redundancy-free **architecture** model (rather than **tools description**)
- Multiple abstraction levels (instruction-accurate and/or cycle-accurate)
Tool Requirements: Compiler

C Compiler

add c = a, b

a = b + c;

Assembly

mul rd = rs, rt

definitions:
rs
rt
rd

ld rd = @
definitions:
@
LD
rd

add rd = rs, rt

definitions:
rs
rt
rd

st @ = rs

definitions:
rs
ST
@

Tool Requirements: Simulator

**Machine Code**

```
add r5 = r2, r1
```

**Simulation Code (C)**

```
add rd = rs, rt
MUL_read (rs, rt);
MUL_add ();
Update_flags ();
writeback (rd);
```

```
lr rd = @
LSU_addrgen();
data_bus.req();
data_bus.read();
writeback (rd);
```

```
mul rd = rs, rt
MUL_read (rs, rt);
MUL_add ();
Update_flags ();
writeback (rd);
```

```
ld rd = @
LSU_addrgen();
data_bus.req();
data_bus.read();
writeback (rd);
```

```
add r5 = r2, r1
ALU_read (r2, r1);  
ALU_add ();  
Update_flags ();  
writeback (r5);
```

```
st @ = rs
LSU_addrgen();
LSU_read(rs);
data_bus.req();
data_bus.write(rs);
```

```
add rd = rs, rt
ALU_read (rs, rt);
ALU_add ();
Update_flags ();
writeback (rd);
```

```
st @ = rs
LSU_addrgen();
LSU_read(rs);
data_bus.req();
data_bus.write(rs);
```
ADL Model

C Compiler

a = b + c;

Simulator

add c = a, b

SYNTAX {
    "ADD" dst, src1, src2
}
CODING {
    0b0010 dst src1 src2
}
BEHAVIOR {
    ALU_read (src1, src2);
    ALU_add ();
    Update_flags ();
    writeback (dst);
}
SEMANTICS {
    src1 + src2 \rightarrow dst;
}

ALU_read (r2, r1);
ALU_add ();
Update_flags ();
writeback (r5);

add r5 = r2, r1
Problem Statement

- **Compiler and Simulator need different information:**
  - **Compiler:** C operation to instruction(s)
    - WHAT is the instruction good for? Purpose?
  - **Simulator:** instructions to sequence of operations
    - HOW is the instruction executed? What actions to perform?
  - **Architecture Designer’s Perspective:**

```c
src1 + src2 \rightarrow dst;
```
```
ALU_read (src1, src2);
ALU_add ();
Update_flags ();
write
```
ASDSP FPGA Implementation

Myjung Sunwoo, Ajiou University,

ASDSP Core Design

✓ SEC 0.18um Synthesis
  • Gate : 77,000
  • Program Memory : 4 Kbyte, Data Memory : 8 Kbyte
  • Frequency : 290MHz
  • Power consumption : 0.87W (3mW/MHz)

FPGA Implementation

✓ iProve Xilinx xc2v6000

Support the Special Instruction Set for FFT Operation and the BMU Instruction
Improve the Performance for OFDM Communication
A low-power ASIP for Infineon DVB-T 2nd generation Single-Chip Receiver:

- ASIP for DVB-T acquisition and tracking algorithms (sampling-clock-synchronization, interpolation / decimation, carrier frequency offset estimation)
- Harvard Architecture
- 60 mostly RISC-like Instructions & Special Instructions for CORDIC-Algorithm
- 8x32-Bit General Purpose Registers, 4x9-Bit Address Registers
- 2048x20-Bit Instruction ROM, 512x32-Bit Data Memory
- I2C Registers and dedicated interfaces for external communication
The Motorola M68HC11 Architecture
Architecture Overview

- **M68HC11 CPU Architecture:** Hot spots
  - 8-bit micro-controller.
    - Harvard Architecture
  - 7 CPU Registers.
  - 6 different Addressing Modes.
  - Shared data and program bus: stalled data access
  - Instruction width: 8, 16, 24, 32, 40: multi-cycle fetch
  - 8-bit opcode: 181 instructions
  - Clock speed: ~200 MHz
  - Performance: non-pipelined
  - Area: 15K to 30K (DesignWare® Library)
Architecture Development with LISA

+ pipelined architecture
+ separate program and data bus

FE -> DC -> EX

512Bytes int. RAM
64Bytes Conf. Reg.
3.5K ext. RAM
61K ext. RAM

0x0000
0x10000

ACCU
Accu A
Accu B
Index X
Index Y
Stack Pointer
Condition
Results

• Area
  < 23k gates

• Clock speed
  ~ 200 MHz

• Execution time speed up
  62% for spanning tree application

• Mapped onto Xilinx FPGA
## Architecture Development with LISA

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Studying the architecture</td>
<td>4 days</td>
</tr>
<tr>
<td>Basic architecture modifications</td>
<td>2 days</td>
</tr>
<tr>
<td>Grouping and coding of the instructions</td>
<td>1 day</td>
</tr>
<tr>
<td>Writing the LISA model</td>
<td></td>
</tr>
<tr>
<td>- Basic syntax and coding</td>
<td>4 days</td>
</tr>
<tr>
<td>- Behavior section</td>
<td>6 days</td>
</tr>
<tr>
<td>Validation</td>
<td>4 days</td>
</tr>
<tr>
<td>HDL Generation</td>
<td>2 days</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>23 days</strong></td>
</tr>
</tbody>
</table>
Design of Application Specific Processor Architectures

Rainer Leupers
RWTH Aachen University
Software for Systems on Silicon
leupers@iss.rwth-aachen.de
Overview

1. Introduction
2. ASIP design methodologies
3. Software tools
4. ASIP architecture design
5. Case study
6. Advanced research topics
1. Introduction
Embedded system design automation

- Embedded systems
  - Special-purpose electronic devices
  - Very different from desktop computers

- Strength of European IT market
  - Telecom, consumer, automotive, medical, ...
  - Siemens, Nokia, Bosch, Infineon, ...

- New design requirements
  - Low NRE cost, high efficiency requirements
  - Real-time operation, dependability
  - Keep pace with Moore’s Law
What to do with chip area?

Additional functionality
Example: wireless multimedia terminals

- Multistandard radio
  - UMTS
  - GSM/GPRS/EDGE
  - WLAN
  - Bluetooth
  - UWB
  - …

- Multimedia standards
  - MPEG-4
  - MP3
  - AAC
  - GPS
  - DVB-H
  - …

Key issues:
- Time to market (≤ 12 months)
- Flexibility (ongoing standard updates)
- Efficiency (battery operation)
As the performance of conventional microprocessors improves, they first meet and then exceed the requirements of most computing applications. Initially, performance is key. But eventually, other factors, like customization, become more important to the customer...“


\[
\text{design budget} = (\text{semiconductor revenue}) \times (\% \text{ for R&D})
\]

\[\text{growth} \approx 15\% \quad \approx 10\%\]

\[
\# \text{IC designs} = (\text{design budget}) / (\text{design cost per IC})
\]

\[\text{growth} \approx 15\% \quad \text{growth} \approx 50-100\%\]

[Keutzer05]

→ Customizable application specific processors as reusable, programmable platforms
**Efficiency and flexibility**

Why use ASIPs?
- Higher efficiency for given range of applications
- IP protection
- Cost reduction (no royalties)
- Product differentiation

Source: T. Noll, RWTH Aachen
2. ASIP design methodologies
ASIP architecture exploration

initial processor architecture

optimized processor architecture
From an ANSI C/C++ application the XRES Compiler generates an optimized set of processor extensions ...

... that is reusable over a range of similar application software code.

Source: Tensilica Inc.
MIPS CorXtend/CoWare CorXpert

1. Profile and identify custom instructions

2. Replace critical code with special instruction

3. Synthesize HW and profile with MIPSsim and extensions

User Defined Instruction

CorXpert Module

CorXpert Module

CorXpert Module

CorXpert Module

CorXpert Module

CorXpert Module

CorXpert Module
- Integrated embedded processor development environment
- Unified processor model in LISA 2.0 architecture description language (ADL)
- Automatic generation of:
  - SW tools
  - HW models
Reflects hierarchical organization of ISAs
LISA operations structure

LISA operation

- DECLARE
- CODING
- SYNTAX
- EXPRESSION
- ACTIVATION
- BEHAVIOR
- SEMANTICS

References to other operations
Binary coding
Assembly syntax
Resource access, e.g. registers
Initiate “downstream” operations in pipe
Computation and processor state update
C compiler generation
LISA operation example

OPERATION ADD
{
    DECLARE
    {
        GROUP src1, src2, dest = { Register }
    }
    CODING { 0b1011 src1 src2 dest }
    SYNTAX { “ADD” dest “,” src1 “,” src2 }
    BEHAVIOR { dest = src1 + src2; }
}

OPERATION Register
{
    DECLARE
    {
        LABEL index;
    }
    CODING { index }
    SYNTAX { “R” index }
    EXPRESSION{ R[index] }
}
Exploration/debugger GUI

- Application simulation
- Debugging
- Profiling
- Resource utilization analysis
- Pipeline analysis
- Processor model debugging
- Memory hierarchy exploration
- Code coverage analysis
- ...

ISS

2005 © R. Leupers
Some available LISA 2.0 models

- **DSP:**
  - Texas Instruments TMS320C54x
  - Analog Devices ADSP21xx
  - Motorola 56000

- **RISC:**
  - MIPS32 4K
  - ESA LEON SPARC 8
  - ARM7100
  - ARM926

- **VLIW:**
  - Texas Instruments TMS320C6x
  - STMicroelectronics ST220

- **μC:**
  - MHS80C51

- **ASIP:**
  - Infineon PP32 NPU
  - Infineon ICore
  - MorphICs DSP
3. Software tools
Tools generated from processor ADL model

- Application
- Compiler
- Assembler
- Linker
- Simulator
- Profiler

Diagram showing the flow from Application through Profiler, Compiler, Assembler, Linker, Simulator, and back to Application.
Instruction set simulation

**Interpretive:**
- flexible
- slow (~ 100 KIPS)

**Compiled:**
- fast (> 10 MIPS)
- inflexible
- high memory consumption

**JIT-CCS™:**
- „just-in-time“ compiled
- SW simulation cache
- fast and flexible
JIT-CC simulation performance

- Dependent on simulation cache size
- 95% of compiled simulation performance @ 4096 cache blocks (10% memory consumption of compiled sim.)
- Example: ST200 VLIW DSP
Why care about C compilers?

- Embedded SW design becoming predominant manpower factor in system design
- Cannot develop/maintain millions of code lines in assembly language
- Move to high-level programming languages
Why care about compilers?

- Trend towards heterogeneous multiprocessor systems-on-chip (MPSoC)
- Customized application specific instruction set processors (ASIPs) are key MPSoC components
- How to achieve efficient compiler support for ASIPs?
Efficient C-compilers cannot be designed for ARBITRARY architectures!

Compiler and processor form a UNIT that needs to be optimized!

“Compiler-friendliness“ needs to be taken into account during the architecture exploration!
Retargetable compilers

Classical compiler

Retargetable compiler

source code

Compiler

processor model

asm code

source code

Compiler

processor model

asm code
• Probably the most widespread retargetable compiler

• Mostly used as a native Unix/Linux compiler, but may operate as a cross-compiler, too

• Support for C/C++, Java, and other languages

• Comes with comprehensive support software, e.g. runtime and standard libraries, debug support

• Portable to new architectures by means of machine description file and C support routines

"The main goal of GCC was to make a good, fast compiler for machines in the class that the GNU system aims to run on: 32-bit machines that address 8-bit bytes and have several general registers. Elegance, theoretical power and simplicity are only secondary."
CoSy compiler system (ACE)

- Universal retargetable C/C++ compiler
- Extensible intermediate representation (IR)
- Modular compiler organization
- Generator (BEG) for code selector, register allocator, scheduler
LISATek C compiler generation

LISA processor model

SYNTAX
{ “ADD" dst, src1, src2 }
CODING
{ 0b0010 dst src1 src2 }
BEHAVIOR
{ ALU_read (src1, src2); ALU_add (); Update_flags (); writeback (dst); }
SEMANTICS
{ src1 + src2 \rightarrow dst; }
...

Autom. analyses

Manual refinement

GUI

CoSy system

C Compiler
LISATek compiler generation

C-Code
int a, b, c;
a = b + 1;
c = a << 3;

ASM-Code
LD R1, [R2]
ADD R1, #1
SHL R1, #3

Code-Selector
Register-Allocator
Scheduler
Instruction-Fetch
ALU
Mem
FE
DE
EX
WB
Write-Back

Pipeline Control

Decoder
Jump
ADD
MUL
SUB
JMP
ADD
R[0..31]
Prog-RAM
Data-RAM
Compiled code quality: MIPS example

- LISATek generated C-Compiler
- Out-of-the-box C-Compiler
- No manual optimizations
- Development time of model approx. 2 weeks

- gcc C-Compiler
  - gcc with MIPS32 4kc backend
  - Used by most MIPS users
  - Large group of developers, several man-years of optimization

Overhead of 10% in cycle count and 17% in code density
Compilers for embedded processors have to generate extremely efficient code

- **Code size:**
  - system-on-chip
  - on-chip RAM/ROM

- **Performance:**
  - real-time constraints

- **Power/energy consumption:**
  - heat dissipation
  - battery lifetime
Compiler flexibility/code quality trade-off

- variety of embedded processors
- retargetable compilation
- dedicated optimization techniques
- unification
- specialization

- DSP
- NPU
- VLIW
Adding processor-specific code optimizations

- High-level (compiler IR)
  - Enabled by CoSy’s engine concept
- Low-level (ASM):

  .C ➔ LISA C Compiler ➔ Scheduled .asm

  Optimization 1 ➔ Assembly API

  Binary Code Generation ➔ Assembler ➔ Linker ➔ .out
4. ASIP architecture design
ASIP implementation after exploration
Unified Description Layer

HDL Generation

Gate–Level Synthesis
(e.g. SYNOPSYS design compiler)

Gate–Level Synthesis

Register-Transfer-Level

L I S A
Challenges in Automated ASIP Implementation

ADL:

Instructions

Arithmetic

- Mul
- Mac

Control

- JMP
- BRC

Independent description of instruction behavior:

+ Efficient Design Space Exploration

HDL:

Multiplier (MUL)

Multiplier (MAC)

Independent mapping to hardware blocks:

- Insufficient architectural efficiency by 1:1 mapping
Unified Description Layer

Structure & Mapping
(incl. JTAG/DEBUG)

Optimizations

Backend (VHDL, Verilog, SystemC)

Gate–Level Synthesis
(e.g. SYNOPSYS design compiler)
Optimization strategies

LISA: separate descriptions for separate instructions

Goal: share hardware for separate instructions

Possible Optimizations
• ALU Sharing
Optimization strategies

LISA: separate descriptions for separate instructions
Goal: same hardware for separate instructions

Possible Optimizations
- ALU Sharing
- Path Sharing

Resource Sharing

Mutual Exclusiveness

Instruction A

LISA Operation A

Instruction B

LISA Operation B

AddressA
DataA
Path PA

AddressB
DataB
Path PB

Register Array

DataA, DataB

AddressA

AddressB
5. Case study
Motorola 6811

Project Goals:

• Performance (MIPS) must be increased

• Compatibility on the assembly level for reuse of legacy code (Integration into existing tool flow)

• Royalty free design

➤ compatible architecture developed with LISA using RTL processor synthesis
Motorola 6811

- legacy code
- compiler
- assembly
- assembler

Increase Performance!!! (MIPS)

0100101010011010 1110010110101111 0000110110110100

?
Motorola 6811

- Bluetooth app.
- 6811 compiler
- assembly
- assembler
- Synthesized Architecture

assembly level compatible

LISA

0100101010011010
1110010110101111
0000110110110100

2006 © R. Leupers
Architecture Development

original 6811 Processor

- 8 bit instructions
- 16 bit instructions
- 24 bit instructions
- 32 bit instructions
- 40 bit instructions

Instruction is fetched by 8 bit blocks:

⇒ up to 5 cycles for fetching!

LISA 6811 Processor

- 16 bit instructions
- 32 bit instructions

16 bit are fetched simultaneously:

⇒ max 2 cycles for fetching!

+ pipelined architecture
+ possibility for special instructions

2005 © R. Leupers
Tools Flow and RTL Processor Synthesis

C-Application

6811 compiler

Assembly

LISA assembler

Executable

LISA model

6811 compatible architecture generated completely in VHDL

1) VLSI Implementation:
   Area: <17kGates
   Clock Speed: ~154 MHz

2) Mapped onto XILINX FPGA
References

- C. Rowen, S. Leibson: *Engineering the Complex SoC: Fast, Flexible Design with Configurable Processors*, Prentice Hall, 2004
- P. Ienne, R. Leupers (eds.): *Customizable and Configurable Embedded Processor Cores*, Morgan Kaufmann, to appear 2006