A Hardware/Software Co-reconfigurable Multimedia Architecture

Yong-Kyu Jung
IEEE Senior Member
Texas A&M University-College Station
Contents

- Introduction
- Related Work
- HW/SW Co-reconfiguration
- Implementation of SID
- Results and Evaluation of SID
- Conclusion
- Q & As
Introduction

- Multimedia System Design Paradigm Shift
  - Design Requirements: real-time/potable/small/complex processing → Heterogeneous MPSoCs
  - Time-to-Market: Design not from scratch/continuous upgrade

- Multimedia System Design Goals
  - Increase ILP/TLP (Parallel/distributed processing)
  - Reconfigure HW accelerators/RFUs
  - Rapid Development (HW/SW reusability/compatibility)
  - Reduce code size (for potable applications)

- Current Approaches for Multimedia HW Design
  - ASIP, RISP, Ext. Proc/ISEs, PDCS, Platforms, etc.
  - EDA, Design automation, etc.
Application-specific System Design

**Hardware**
- **ASIP**
  - Fixed Application-specific Units
  - ASICs
- **RISP**
  - RFUs
  - FPGAs
- **Ext. µP**
  - Flexible Proc. Core
  - ASICs
- **Fixed GP-µP**
  - DBT HW

**Software**
- **New ISAs (instr. sets)**
- **New compilers/application SW**
- **Pre-defined operations by instructions**
- **Add on instructions to run RFUs**
- **Optimize u-ops for application**
- **Instruction set extension (ISE): update compiler**
- **Better optimized u-ops than RISPs**
- **Dynamic Binary Translation (DBT)/existing compiler/SW**
- **SW DBT/new compiler/SW**

IEEE Embedded Systems for Real-time Multimedia 2006 (ESTIMedia'06)

Oct. 26~27, 2006
# Reconfigurability in different approaches

<table>
<thead>
<tr>
<th>Technique</th>
<th>HW/SW DBT</th>
<th>RISP/ISE</th>
<th>SID†</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overhead</strong></td>
<td>DBT logic or new compiler</td>
<td>Reconfiguration logic and time</td>
<td>LUTs††</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Improvement New Processors</td>
<td>RFUs/ISE</td>
<td>Optimized micro-arch./operations</td>
</tr>
<tr>
<td><strong>New Compiler</strong></td>
<td>Req. or not Req.</td>
<td>Required ††††</td>
<td>Not Req.</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>Limited††††</td>
<td>Non-guaranteed</td>
<td>Guaranteed</td>
</tr>
</tbody>
</table>

- RFUs: Reconfigurable Functional Units; LUTs: Look-up Tables;
- †: SID represents HW/SW co-reconfiguration technique;
- ††: LUTs used in SID can be located into instruction caches;
- †††: since new instructions are added, compiler needs to be modified; and
- ††††: it is difficult to match instr.s including execution cycle and interrupt accuracy.
However, ...

- **SW gets larger and more complex**
  - Expensive to redevelop/reverify new and/or modified SW
- **Frequent updates (both HW/SW)**
  - New algorithms, applications & standards
- **Less compatible btw processors from different vendors**
  - Limit embedded system developers to migrate performance/cost effective new systems (processors)
- **MPSoCs**
  - Remap/verify parallel distributed SW
- **Need closer HW/SW collaboration**
  - Current approaches: HW (processors) → SW (compilers) → embedded SW developers remain to work at ISA-level → less precise collaboration with HW (micro-architecture)
Proposed Solution

HW/SW Co-reconfiguration Technique

- The Technique that allows software developers “to retarget their reconfigurable multimedia systems that are not necessary to employ programmable devices”.
- Precisely control their target HW/SW continuously and seamlessly

Applicable Target Processors

- Single Processor: DSP/EP
- Multiprocessors: Dual/Quad-DSP/EPs
  - Homo-/heterogeneous MPSoCs
HW/SW Co-reconfiguration Technique

Targeting types of the technique

- **Reconfigurable Instruction Decoder (RID):**
  - For single/homogeneous multiprocessor (DSP or EP) embedded systems

- **Smart Instruction Decoder (SID):**
  - For heterogeneous (DSP+EP) MPSoCs

Functions

- **Decodes any binary instructions** of source ISAs as instructions defined target ISAs
- **Maintains static binary reformation, translation and/or optimization** in ISA level
- **HW/SW co-reconfiguration** in micro-architecture level
- **SW developers access the micro-ops implemented in the target processors**
SID HW/SW Co-reconfigurable Architecture

- **Two Reconfiguration Processes**
  - The off-chip static (OSR) reconfiguration
  - The on-chip dynamic (ODR) reconfiguration

- **Order of Reconfiguration Processes**
  - Sequential OSR for DSP and for EP
  - Concurrent ODR for both DSP and EP
  - Achieve HW/SW Co-reconfiguration with Fixed Decoder

- **Implementation of SID**
  - One or Multiple of the same duplicated HW that consists of
    - Look-up tables (LUTs)/Micro-operation units (MOUs)
    - Pipelined front-ends including the fixed decoder(s)
  - integrated with micro-architectures of the target multimedia processors.
SID for H-MPSoC (DSP+EP)

Off-chip Static Reconfiguration (OSR)

- Target ISA
- Static Binary Optimization
- Static Binary Translation
- Source ISAs
- Instr. Set Reformation
- u-op Generation

SID-Software

On-chip Dynamic Reconfiguration (ODR)

- EP-Pre-fetch
- Unaligned Binary Streams
- DSP-Pre-fetch
- EP-L_Mem
- DSP-L_Mem
- DSP-LUTs
- EP-LUTs
- DSP - Pipelined Micro-architecture
- EP - Pipelined Micro-architecture

SID-Hardware

Micro-architecture of a Heterogeneous MPSoC (DSP + Embedded)

- Aligned Binary Streams
- DSP-Fetch
- EP-Fetch
- DSP-Decode
- EP-Decode
- EP-Pre-fetch
- DSP-Pre-fetch
- EP-L_Mem
- DSP-L_Mem
- EP-LUTs
- DSP-LUTs
- DSP-Micro-Operation Unit
- EP-Micro-Operation Unit

IEEE Embedded Systems for Real-time Multimedia 2006 (ESTIMedia'06)
SID HW Reconfiguration

64-bit front-aligned binary instructions

64-bit front-aligned binary instructions

Fixed Instruction Decoder

Opcode Extraction

Instruction Identification

Instruction Decode

LUTs (fast SRAM)

Reconfiguration Signals

LUTs (ROM-1)

LUTs (ROM-n)

ISA History LUT

Decide Outputs

\( t_{\text{reconfig}} \)

\( t_{\text{LUT-write}} \)

\( t_{\text{MUX-switch}} \)

\( t_{\text{LUT-read}} \)
**HW/SW Co-reconfiguration Procedure**

- **OSR** statically reforms source ISA(s) to a target ISA.
- **OSR** performs an additional, optional static binary translation and optimization including ISEs.
- Decoding info generation for ODR (LUTs/MOUs)
- **ODR** downloads/switches the LUTs
- **ODR** dynamically decodes incoming binary instr.
Off-chip Static Reconfiguration (OSR)

**Operations of OSR**
- **Primary:** Reformation (RFM)
- **Optional:** Extended Instr. Set Design (EID); Security Coding (SCD); Translation (TRN); and Timing Optimization (OPT)

**The mandatory reconfiguration phase**
- Ensures generation of information as LUTs for ODR.

**The optional reconfiguration phase**
- Translates and optimizes the source ISA of the existing DSP/embedded processor to the target ISA.
- Design own instr sets (beyond ISEs)
- Remap the opcodes for security of the code.
OSR: Function/Timing Compatibility

- All ISAs primarily assure functional equality
- LUTs for Dynamic Reconfiguration
  - Comprehensive decode LUT
    - contains info. to decode all instr.s in both source and target ISAs.
  - Other LUTs
    - hold info. to continuously operate with the micro-operation unit (MOU) through subsequent HW blocks
- Classification of target ISAs by timing accuracy
  - Low timing accuracy: equal sequences of instr.s
  - Medium timing accuracy: + instr. cycles
  - High timing accuracy: + micro-ops cycles
On-chip Dynamic Reconfiguration (ODR)

- **Operations of ODR**
  - Dynamically decodes incoming binary streams as consecutive instructions
  - Generates decode outputs
    - Instr. bit-length
    - Instr. types & formats
    - Access pointer of MOU @ each pipeline stage

- **Trade-offs in ODR operations**
  - Decoding time (*latency*);
  - Timing compatibility and accuracy; and
  - Configuring micro-operations through the target micro-architecture (*fixed/reconfigurable*)
## Trade-offs in SID Reconfiguration Operations

<table>
<thead>
<tr>
<th>Reconf. Ops</th>
<th>Reconf. Method</th>
<th>Reconf. Speed</th>
<th>Memory Size†</th>
<th>ISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT Download</td>
<td>Dynamic</td>
<td>Low</td>
<td>Small</td>
<td>R-I*</td>
</tr>
<tr>
<td>LUT Switch</td>
<td>Dynamic</td>
<td>Mediu m</td>
<td>Large</td>
<td>R-I*</td>
</tr>
<tr>
<td>Instr. Set Merge††</td>
<td>Static</td>
<td>High</td>
<td>Small</td>
<td>M-I**</td>
</tr>
</tbody>
</table>

†: LUT memory size, ††: multiple instruction sets can be shared the same LUT, *
*: instr.s for reconfiguring source instr. sets, and **: a single united instr. set by merging multiple instr. sets. This may require redesigning the instr.s used in all instr. sets merged.
SID Implementation

- **OSR:**
  - Used a combination of computerized ledger sheets
  - Focused on automated generation of configuration information for ODR
  - Understood and interpreted source ISAs (DSP-TMS320C55 and an EP-ARM11) was the key

- **ODR and the rest of the pipeline: VHDL**
  - was meant to evaluate important design factors, i.e., chip area, speed-up, and power dissipation.
## Summary of TI-C55 & ARM Instruction Sets

### DSP (TI-C55) Instruction Set

<table>
<thead>
<tr>
<th>Operation Types of Instructions</th>
<th>DSP Special</th>
<th>Mem/Reg access</th>
<th>Control/ Branch</th>
<th>Arithmetic/ Logical/Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Operation</td>
<td>110</td>
<td>135</td>
<td>192</td>
<td>36</td>
</tr>
<tr>
<td>Length of Instructions (bit)</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td># of instructions</td>
<td>6</td>
<td>100</td>
<td>198</td>
<td>87</td>
</tr>
<tr>
<td>Length of Opcodes (bit)</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td># of instructions</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>148</td>
</tr>
</tbody>
</table>

### Embedded Processor (ARM) Instruction Set

<table>
<thead>
<tr>
<th>Operation Types of Instructions</th>
<th>DSP Special</th>
<th>Mem/Reg access</th>
<th>Control/ Branch</th>
<th>Arithmetic/ Logical/Shift</th>
<th>Misc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Operation</td>
<td>38</td>
<td>76</td>
<td>40</td>
<td>12</td>
<td>35</td>
</tr>
<tr>
<td>Length of Opcodes (bit)</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td># of instructions</td>
<td></td>
<td></td>
<td></td>
<td>172</td>
<td>7</td>
</tr>
</tbody>
</table>

| Length of Instructions (bit)     | 8           | 16             | 24              | 32                        | 40    |
| # of instructions                | 6           | 100            | 198             | 87                        | 2     |
| Length of Opcodes (bit)          | 4           | 5              | 6               | 7                         | 8     |
| # of instructions                | 2           | 1              | 2               | 148                       | 240   |
SID HW Prototype in VHDL

- Implemented as a **soft RTL synthesizable VHDL core**
- Verified with **functional** (ModelSim) and **timing** (Quartus II) simulations
- **The front-end pipeline** (pre-fetch, the fetch, and the decode) were implemented as a HW part of SID.
- **The rest of the pipeline** (from the dispatch to the last write-back) was also implemented to estimate performance of SID.
- **An FPGA version of SID** was evaluated as a case study.
  - SID will be implemented as an ASIC for more practical evaluation
Evaluations of SID

- **Target ISAs: C55 and ARM**
  - to prove the conceptual performance improvement of SID

- **The platform:**
  - the 7-stage pipelined processors + memories (LUTs/MOUs/I-MEM)

- **Evaluation of OSR:**
  - binary compatibility

- **Evaluation of ODR:**
  - decoding operations and integration
  - for multi-processors
    - an area increase; speed; and power consumption
Testbenches

- **Generation of Testbenches**
  - More than 3000 and 2000 instr.s in C55 and in ARM, respectively, were generated and tested.
  - Functional verification of the heterogeneous MPSoC

- **Example for C55**
  - Seven instr.s including an undefined one (illegal opcode)
  - For variable-length instr. test
    - Five different lengths from 8- and 40-bit
  - For different opcode length test
    - Three different opcodes, such as 4-, 7-, and 8-bit
## DSP-C55 Testbench for SID

<table>
<thead>
<tr>
<th>PO</th>
<th>Mnemonics / Hexadecimal</th>
<th>Binary</th>
<th>Instr / opcd / e-opcd (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MOV ACx, Xmem, Ymem</td>
<td>10000000 XXXMMMMYY YMMM10SS</td>
<td>24 / 8 / 2</td>
</tr>
<tr>
<td></td>
<td>{80}372(&lt;B&gt;)†</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SQA[R] [ACx,] ACy</td>
<td>0101010E DDSS001%</td>
<td>16 / 7 / 3</td>
</tr>
<tr>
<td></td>
<td>{55}6(&lt;3&gt;)††</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MAC[R][40] [uns(Xmem[]), [uns(Cmem[]), ACx :: MPY[R][40] [uns(Ymem[]), [uns(Cmem[]), ACy</td>
<td>10000010 XXXMMMMYY YMMM01mm uuDDDDDg%</td>
<td>32 / 8 / 2</td>
</tr>
<tr>
<td></td>
<td>{82}CAB&lt;7&gt;†††26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>mmap / {98}</td>
<td>10011000</td>
<td>8 / 8 / 0</td>
</tr>
<tr>
<td>4</td>
<td>BCC P24, cond</td>
<td>01101000 xCCCCCC CCPPPPPPPPPPPPPPPPPP</td>
<td>40 / 8 / 0</td>
</tr>
<tr>
<td></td>
<td>{68}73AACC55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Undefined** / {FD}</td>
<td>11111101</td>
<td>8 / 8 / 0</td>
</tr>
<tr>
<td>6</td>
<td>MOV Smem, dst</td>
<td>1010FD00 AAAAAA00</td>
<td>16 / 4 / 0</td>
</tr>
</tbody>
</table>
A Testbench for ARM

- Example for 32-bit fixed-length ARM instr.s
  - Seven instr.s including an undefined one (legal opcode with illegal ext. opcode)
  - For different opcode length test
    - Two different opcode lengths, 4- and 8-bit
  - For different ext. opcode length/location test
    - Three different ext. opcode lengths, 4-, 1- and 0-bit
    - Two different ext. opcode locations, 7-4 and 4 (little-endian)
## EP-ARM Testbench for SID

<table>
<thead>
<tr>
<th>PO</th>
<th>Mnemonics / Hexadecimal</th>
<th>Binary</th>
<th>Instr / opcd / e-opcd (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MLAL / Rd := (Rm *Rs)+Rn</td>
<td>Cond00001UASRdHiRdLoRs1001Rm</td>
<td>32 / 8 / 4</td>
</tr>
<tr>
<td></td>
<td>A{0B}38C&lt;9&gt;3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>LDR / Rd := (addr)</td>
<td>Cond01IPUBWL</td>
<td>32 / 8 / 0</td>
</tr>
<tr>
<td></td>
<td>F{59}699CE</td>
<td>RnRdOffset</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>STM / (Push)</td>
<td>Cond100PUSWL</td>
<td>32 / 8 / 0</td>
</tr>
<tr>
<td></td>
<td>0{86}51DC5</td>
<td>RnRegisterlist</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MRC / Rn:=cRn{&lt;op&gt;cRm}</td>
<td>Cond1110CPOpcLCRnRdCPNCP1CRm</td>
<td>32 / 8 / 1</td>
</tr>
<tr>
<td></td>
<td>3{E}6DC6C&lt;A&gt;††††</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BL / R14:=R15, R15:=addr</td>
<td>Cond101Loffset</td>
<td>32 / 4 / 0</td>
</tr>
<tr>
<td></td>
<td>1{A}48885D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CMP/CPSR flags:=Rn-Op2</td>
<td>Cond0011010S</td>
<td>32 / 8 / 0</td>
</tr>
<tr>
<td></td>
<td>E{34}5ACE1</td>
<td>RnRdOperand2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Undefined**</td>
<td>Cond00001UASRdHiRdLoRs0110Rm</td>
<td>32 / 8 / 4</td>
</tr>
<tr>
<td></td>
<td>6{0E}3CF&lt;6&gt;0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simulation Result I

Unaligned Binary Instruction Processing

- **Pre-fetch**
  - Pre-fetch 64-bit-wide unaligned instr.s

- **Fetch**
  - Fetch the 64-bit binaries and store to an 128-bit instruction queue
  - Append the fetched binary or hold to fetch
  - In case of decoding error,
    - Variable-length instr. set: discards 8-bit (minimum length) after the 1\textsuperscript{st} bit location of the illegal instr.
    - Fixed-length instr. set: discards the entire-bit of the illegal instr. (e.g., 32-bit for ARM)
Simulation Result II

Aligned Binary Instruction Decoding

- **Decode**
  - Dynamically interprets according to the contents of CD-LUT.
  - The decoding results are:
    - Length, type and format of the instr. decoded,
    - An instr. truncated,
    - Access pointer of micro-operations, and
    - Status of the decoding result
  - Detects illegal instr.
  - Generates and forwards a decode error for the further process
### Simulation Waveform-C55

<table>
<thead>
<tr>
<th>Time</th>
<th>Address</th>
<th>Instruction Type/Fields</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000C</td>
<td>00000000</td>
<td>Opcode: “FD”</td>
<td></td>
<td>Decode error on the instr.</td>
</tr>
<tr>
<td>000D</td>
<td>00000000</td>
<td>Opcode: “FD”</td>
<td></td>
<td>Decode error on the instr.</td>
</tr>
<tr>
<td>000E</td>
<td>00000000</td>
<td>Opcode: “FD”</td>
<td></td>
<td>Illegal opcode “FD” is eliminated</td>
</tr>
<tr>
<td>000F</td>
<td>00000000</td>
<td>Opcode: “FD”</td>
<td></td>
<td>Illegal opcode “FD” is eliminated</td>
</tr>
<tr>
<td>0010</td>
<td>00000000</td>
<td>Opcode: “FD”</td>
<td></td>
<td>Illegal opcode “FD” is eliminated</td>
</tr>
</tbody>
</table>

- **Hold pre-fetch**
- **Append the aligned instr. fetched**
- **The opcodes extracted**
- **Instr. Type/fields**
- **Addr. of MOU for dispatch**
- **Reserved for future**
- **The instr. truncated**

---

**Notes:**

- Oct. 26~27, 2006
- IEEE Embedded Systems for Real-time Multimedia 2006 (ESTIMedia'06)
Simulation Waveform-ARM

Unaligned binary stream read from I-MEM
Fixed 64-bit binary pre-fetched
Program Counter @ the fetch
Valid outputs produced in the stages
Decode error on the instr. (op: "0E" & ext op: "6")
The opcodes extracted
Decode output
The instr. truncated
Hold pre-fetch
The instr. truncated
SID Optimization and Evaluation for TI's OMAP2420

**Evaluation Categories**

- **Optimization Results**
  - Area (0.01xLC): Dual-DSPs: SID with shared LUTs, OMAP: SID non-superpipelined, OMAP: SID superpipelined
  - Memory (0.1xK Bit): Dual-DSPs: SID with shared LUTs, OMAP: SID non-superpipelined, OMAP: SID superpipelined
  - Speed Sys_Clk (MHz): Dual-DSPs: SID with shared LUTs, OMAP: SID non-superpipelined, OMAP: SID superpipelined
  - Speed Mem_Clk (MHz): Dual-DSPs: SID with shared LUTs, OMAP: SID non-superpipelined, OMAP: SID superpipelined
  - Power (0.1xmw): Dual-DSPs: SID with shared LUTs, OMAP: SID non-superpipelined, OMAP: SID superpipelined
Advantages of SID

- Binary code compatibility/reusability
- Code security/optimization (size/cycle)
- Chip space for large/specific multimedia cores
- Mitigation of the energy budget in multimedia
- Performance improvement
- Precise control of multimedia HW by SW developers
- Diminishes ASIC development risk
  - Quick transition to ASIC
  - Reduce ASIC development cost
Conclusion

Reconfigurable architectures of the instruction decoders

- Retarget single/multiple multimedia proc.s
- ASIC Implementation for continuous upgrade and compatibility
- Lower overhead and more flexibility
  - Than RISPs with ISEs and HW-/SW-based DBT
- Automated binary interpretation in micro-architecture-level
  - Leverage HW and SW collaboration in multimedia architectures
Q&As

Thank you!!!