Post-Silicon Patchable Hardware

Masahiro Fujita

VLSI Design and Education Center (VDEC)
The University of Tokyo

July 22nd, 2011
Respin is becoming more frequent.
Manufacturing Cost

Respin risk is increasing dramatically

[Respin risk is increasing dramatically]

[Mask Set Cost (US$)]

$5M

$4M

$3M

$2M

$1M

90nm
65nm
45nm
32nm

[Nikkei Electronics, 2008]
Causes for Respins

IC/ASIC Designs Having One or More Re-spins by Type of Flaw

[Collett International Research 2005]

Logic and functional errors are the leading cause

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Conventional SoC Design Flow

- **Bug Fix**
- **Bug Localization**
- **Verification/Simulation**
  - Pre-Silicon RTL Verification

**High-Level Description**

**High-Level Synthesis**

**Machine-Generated RTL**

**Logic Synthesis**

**Place & Route**

**SoC Design**

75% of the whole development time [Source: Intel 2007]

- **Bug Fix**
- **Bug Localization**
  - Need to Understand RTL
- **Error Detection**
- **Post-Silicon RTL Validation**
Proposed Patchable SoC Design Flow

Bug Fix

Bug Localization

Verification/Simulation

Pre-Silicon High-Level Verification

High-Level Description

High-Level Synthesis of Patchable HW

Logic Synthesis Place & Route

Patchable SoC Design

Error Detection

Bug Localization

Bug Fix

Patch Compilation

Post-Silicon High-Level ECO

No Respin Needed!
**Proposed Patchable Hardware**

**Efficeum** offers behavioral-level programmability using a patchable controller.

- **Patchable Controller**
  - Hardwired FSM
  - Patch FSM

- **Custom Datapath**
  - ALU1
  - ALU2

**Partially-Programmable Circuit (PPC)** offers logic-level programmability using a mixed gate/LUT circuit.

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Efficeum:
An Energy-Efficient Patchable Accelerator
For Post-Silicon Engineering Changes
Energy Efficiency vs. Programmability

Energy Efficiency of 90nm OFDM

- Fixed-function HW: 200GOPS/W
- Embedded Proc.: 4GOPS/W  \textbf{50X!}
- Laptop Proc.: 0.05GOPS/W  \textbf{4,000X!}

>100GOPS High Performance
\sim 1W Power/Thermal Constraints

Energy efficiency (in [GOPS/W] or [J/op])

- How much computation can be done in a given energy
- Slowing down the chip reduces power but not efficiency
Fixed-Function Accelerator

- Achieves high energy efficiency by customization:
  - Hardwired controller → **No reprogrammability**
  - Highly-customized datapath → **Low flexibility**
Proposed Patchable Accelerator

- Behavioral reprogrammability by control patching
- Increased flexibility by adding register file via data bus
Patch Logic

Program Counter

PC1

PC2

PC1'

PC2'

Hardwired Controller

Control Signal

Signal Memory

Control Signal Patch

Program Counter Patch

Patch Memory
Patching Example (1/2)

Scheduling Result of Initial Design

<table>
<thead>
<tr>
<th>PC</th>
<th>ALU1</th>
<th>ALU2</th>
<th>MUL1</th>
<th>Next PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>×</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td></td>
<td>×</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dataflow graph for Initial Design

Hardwired logic

Patch logic

Fujita Lab. – VLSI Design and Education Center - University of Tokyo 13
Patching Example (2/2)

Scheduling Result After Engineering Change

<table>
<thead>
<tr>
<th>PC</th>
<th>ALU1</th>
<th>ALU2</th>
<th>MUL1</th>
<th>NextPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td>2 4</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td>-</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td></td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
<td></td>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Patching-Based Post-Silicon ECO Flow

1. **C Program**
   - High-Level Synthesis
   - Fixed-Function HW
   - Inserting RF & Patch Logic

2. **Post-Silicon ECO (Spec. Change & Bug Fix)**
   - Writing into Patch Memory
   - Efficeum

3. **Post-ECO Program**
   - Computing the Difference Between Two Programs
   - Patch Compilation

**Efficeum**

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Experimental Setup

- Example: 8x8 IDCT
- Technology: FreePDK 45nm
- Logic Synthesis: Synopsys Design Compiler Ultra
  - High effort options with gated clock optimization
- P&R: Cadence SoC Encounter
- Simulation: Synopsys VCS
- Power/timing analysis: Synopsys PrimeTime PX
  - Simulation results are used for power calculation
- Energy efficiencies (GOPS/W) are compared
Energy Efficiency Comparison

8x8 IDCT (FreePDK 45nm technology)

Offers a tradeoff between efficiency and programmability

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Area & Performance Comparison

- **Area [µm²]**
  - Up to 40% Reduction
  - 5X Smaller

- **Operating Frequency [MHz]**
  - 5% 20% Improvement

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Area Comparison

4x reduction

18% increase

Fully-programmable accelerator

Single-function Hardwired accelerator

Efficeum

(Technology: FreePDK 45nm (NCSU/Nangate), Operating Frequency: 200MHz)

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Power Comparison

6x reduction

13% increase

Fully-programmable accelerator

Single-function Hardwired accelerator

(Fully-programmable accelerator)

Efficeum

Controller
Multiplexers
Arithmetic
Register file
Local Store

(Technology: FreePDK 45nm (NCSU/Nangate), Operating Frequency: 200MHz)
Incremental High-Level Synthesis and Patch Compilation For High-Level ECO
Conventional High-Level Synthesis

- Several phases are applied separately
  - This prevents incremental synthesis

Allocation

- AD
- D1
- AD
- D2
- MUL
- 1
- SHFT
- 1

Scheduling

- Step 1
- Step 2
- Step 3

Binding

- ADD
- 1
- ADD
- 2
- MUL
- 1
- SHFT
- 1

Datapath

FSM

Registers

Datapath

FSM

Datapath

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Incremental High-Level Synthesis

- Each operation is scheduled and bound incrementally, and the hardware is enhanced accordingly.

**Incremental Scheduling & Binding**

- Step 1
- Step 2
- Step 3

**Datapath**

**FSM**

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Patch Compilation

- Same as incremental synthesis, except the datapath enhancement is not allowed (only FSM is enhanced)
Incremental Scheduling Procedure

- Extension of Swing Modulo Scheduling for VLIW compilers [Llosa et al., PACT ‘96]
Swing Scheduling Procedure

- Mix of top-down and bottom-up scheduling

Phase 1: Bottom-Up Scheduling of Critical Path and Their Ancestors

Phase 2: Top-Down Scheduling of the Descendants of Scheduled Operations

Phase 3: Bottom-Up Scheduling of the Ancestors of Scheduled Operations
Incremental Step Insertion

- A novel technique enabling incremental swing scheduling
- During swing scheduling, a new control step is inserted between the scheduled steps when needed
Incremental Binding Procedure

- For each operation, all possible combinations of (resource, registers) are examined
- If no such binding is found, new interconnects between resource and registers are introduced

![Diagram](image)

- **Step 1**: ADD1, ADD2, MUL1, SHFT1
- **Step 2**: +, +, <<
- **Step 3**: ×

Enhanced Datapath

A multiplier exists but no register-to-multiplier interconnect exists
Experimental Setup

- The proposed method has been implemented in our high-level synthesis framework Cyneum
  - Incremental high-level synthesis
  - Patch compiler for Efficieum
- Example: 5 benchmark designs
  - C programs of about ~100 lines
  - Functions from IDCT, ADPCM, MPEG
  - Post-ECO examples are generated by random graph perturbation (next slide)
- Evaluated the quality of the method through the patch size and compilation time
Generating ECO Examples

The following graph perturbations are randomly applied to CDFG

Original CDFG

Perturbation 1: Opcode Change

Perturbation 2: Operand Change

Perturbation 3: Introducing A New Operation
Evaluation of Patch Compiler

- For each benchmark, random CDFG perturbation is applied $M$ times. For each $M$, 100 different post-ECO designs are generated and then patches are compiled.
PPC:
Increasing Yield Using Partially-Programmable Circuits

A collaborative work with Prof. Shigeru Yamashita (Ritsumeikan Univ.)
Design For Yield

- At physical level, there are many techniques available for yield/defect tolerance enhancement
- At logic level, the following techniques can be applied for each module
  - TMR: Voting
  - DMR: If one module is defective, the other can be used
  - Reconfigurable devices: synthesize not to use defective parts

Too much overhead in area and performance
Objective of This Work

- Enhance the defect tolerance by using a Partially-Programmable Circuit (PPC)
  - PPC is a hybrid LUT/gate circuit
  - To correct a single defect, full programmability such as FPGAs is unnecessary
  - A defective wire can be made redundant by reprogramming LUTs in PPC

- Propose a design methodology
  - Synthesis of PPC
    - Where to put LUTs
  - How to reprogram LUTs for defective wires
PPC (Partially-Programmable Circuit)

Conventional Part consisting of conventional gates

Non-Programmable Part consisting of conventional gates

Programmable Part consisting of LUTs and MUXs

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
Defect Correction in PPC

Find out that a wire $c_i$ is defective

By reprogramming LUTs, the wire $c_i$ becomes redundant

Call $c_i$ as Robust Connection (RC)
LUTs are used partially in the circuit

There is no redundancy now
PPC Example: Redundancy Addition 1

By adding this wire, some wires become Robust Connections (RCs)

RC: Wires which can become redundant by reprogramming LUTs
NRC: Wires which are not RCs
PPC Example: Redundancy Addition 2

![Diagram of PPC Example: Redundancy Addition 2](image-url)
PPC Example: Redundancy Addition 3
Since we assume a single defect, multiple redundant connections to an LUT are multiplexed.

Colored wires: Robust Connection (RC)
Black wires: Non-Robust Connection (NRC)
CSPF: Flexibility of Logic Circuits

Logic function

CSPF

Truth table

Fujita Lab. – VLSI Design and Education Center - University of Tokyo
SPFD: Flexibility of LUT Circuits

$g_1$ 's flexibility by SPFDs

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
\end{array}
\]
Proposed Synthesis Method

Basic procedure

1. Perform a LUT mapping
2. Determine LUTs to keep
   - Needs a better heuristic
     - Reconvergence points, non-critical nodes
3. Perform a technology re-mapping
4. Adding redundant wires to LUTs
   - How to find good wires?
   - Currently, wires are identified exhaustively
Preliminary Experiments

1. Mapping to K-input LUTs (K=3,4,5)
2. Re-mapping with keeping LUTs at the outputs
3. For each LUT, if connecting a wire to the LUTs make another wire RC, then it is selected. Terminate if the number of LUT inputs is 6.
# Experimental Results

## Connections which are robust to stuck-at-0/1

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Stuck-at-0</th>
<th></th>
<th>Stuck-at-1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Robust</td>
<td>Non-Robust</td>
<td>Robust</td>
<td>Non-Robust</td>
</tr>
<tr>
<td></td>
<td>Original</td>
<td>Added</td>
<td>Original</td>
<td>Added</td>
</tr>
<tr>
<td>alu2</td>
<td>586</td>
<td>13</td>
<td>25</td>
<td>582</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>alu4</td>
<td>1093</td>
<td>28</td>
<td>92</td>
<td>1070</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>apex6</td>
<td>591</td>
<td>86</td>
<td>106</td>
<td>589</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>108</td>
</tr>
<tr>
<td>rot</td>
<td>421</td>
<td>161</td>
<td>218</td>
<td>411</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>172</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>228</td>
</tr>
<tr>
<td>too_large</td>
<td>472</td>
<td>15</td>
<td>256</td>
<td>453</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>275</td>
</tr>
<tr>
<td>vda</td>
<td>1251</td>
<td>153</td>
<td>221</td>
<td>1318</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>213</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>C880</td>
<td>185</td>
<td>32</td>
<td>144</td>
<td>212</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>117</td>
</tr>
<tr>
<td>C1355</td>
<td>219</td>
<td>225</td>
<td>143</td>
<td>390</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>176</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>182</td>
</tr>
<tr>
<td>C1908</td>
<td>626</td>
<td>37</td>
<td>66</td>
<td>599</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>93</td>
</tr>
<tr>
<td>C2670</td>
<td>710</td>
<td>59</td>
<td>176</td>
<td>704</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>182</td>
</tr>
<tr>
<td>C3540</td>
<td>1500</td>
<td>63</td>
<td>210</td>
<td>1462</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>52</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>248</td>
</tr>
</tbody>
</table>