xPilot: A Platform-Based System-Level Synthesis for Reconfigurable SOCs

Prof. Jason Cong
cong@cs.ucla.edu
UCLA Computer Science Department

Outline

◆ Motivation
◆ xPilot system framework
◆ Behavior-level synthesis in xPilot
  ▪ Advantages of behavioral synthesis
  ▪ Scheduling
  ▪ Resource binding
◆ System-level synthesis in xPilot
  ▪ Synthesis for ASIP platforms
  ▪ Design exploration for heterogeneous MPSoCs
◆ Conclusions
Field-Programmable SOCs are Here: Altera Stratix II FPGA

90nm Stratix II FPGA

Adaptive Logic Blocks

M512 SRAM Block

M4K Block

High-Speed I/O Channels

Dynamic Power Alignment

I/O Channels with External Memory Interface Circuitry

60,440 Equivalent Logic Elements

2,544,192 Memory Bits

Courtesy Altera

Field-Programmable SOCs are Here: Xilinx Virtex-4 FPGA

Xilinx FPGA

H.264/AVC hardware blocks

PowerPC 405 (PPC405) core

450 MHz, 700+ DMIPS RISC core

(32-bit Harvard architecture)

Courtesy Xilinx
What about FP-SOC Design Tools

◆ Synthesis
  ▪ Behavior-level synthesis: from behavior specification (e.g. C, SystemC, or Matlab) to RTL or netlists
  ▪ System-level synthesis: from system specification to system implementation

◆ Verification
  ▪ Behavior-level verification
  ▪ System-level verification

ESL Tools – A Lot of Interests ...

G. Smith, Chief EDA Analyst Gartner Dataquest, June 2005
  Increase in revenue based on "Electronic System Level (ESL) design"
  http://www.reuters.com/ovei/dac.showArticle.jhtml?artid=10164302406
GartnerDataquest’s ESL Landscape, 2005

**Behavioral Level**

<table>
<thead>
<tr>
<th>Algorithmic Methodology</th>
<th>Processor/Memory Methodology</th>
<th>Control Logic Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral Algorithmic Design</td>
<td>Behavioral Processor/Memory Design</td>
<td>Behavioral Control Design</td>
</tr>
</tbody>
</table>

**Architectural Level — Architectural Design**

<table>
<thead>
<tr>
<th>Algorithmic Methodology</th>
<th>Processor/Memory Methodology</th>
<th>Control Logic Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic Design and Entry</td>
<td>Processor/Memory Design and Entry</td>
<td>Processor/Memory Design and Entry</td>
</tr>
<tr>
<td>Algorithmic Synthesis</td>
<td>ESL Synthesis</td>
<td>ESL Synthesis</td>
</tr>
<tr>
<td>Application Engine Compiler</td>
<td>Communications Compiler</td>
<td>Application Engine Compiler</td>
</tr>
<tr>
<td>Algorithmic Power Analysis</td>
<td>Application Engine Compiler</td>
<td>Processors/Memory Power Analysis</td>
</tr>
</tbody>
</table>

**Architectural Level — Platform-Based Design**

<table>
<thead>
<tr>
<th>Algorithmic Methodology</th>
<th>Processor/Memory Methodology</th>
<th>Control Logic Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Engine Compiler</td>
<td>Processor/Memory Platform Design</td>
<td>Processor/Memory Platform Design</td>
</tr>
<tr>
<td>Algorithmic Modeling</td>
<td>Processor/Memory Modeling</td>
<td>Processor/Memory Modeling</td>
</tr>
</tbody>
</table>

---

**xPilot: Platform-Based Synthesis System**

- **Uniqueness of xPilot**
  - Platform-based synthesis and optimization
  - Communication-centric synthesis with interconnect optimization
Outline

- Motivation
- xPilot system framework
  - Behavior-level synthesis in xPilot
    - Advantages of behavioral synthesis
    - Scheduling
    - Resource binding
  - System-level synthesis in xPilot
    - Synthesis for ASIP platforms
    - Design exploration for heterogeneous MPSoCs
- Conclusions

Motivation (1)

- Design complexity is outgrowing the traditional RTL method
  - Behavioral synthesis – a critical technology for enabling the move to higher level of abstraction
  - Reasons for previous failures
    - Lack of a compelling reason: design complexity is still manageable a decade of ago
    - Lack of a solid RTL foundation
    - Lack of consideration of physical reality
Motivation (2)

- Behavioral synthesis provides combined advantages
  - Shorter verification/simulation cycle
  - Better complexity management, faster time to market
  - Rapid system exploration
    - Quick evaluation of different hardware/software boundaries
    - Fast exploration of multiple micro-architecture alternatives
  - Higher quality of results
    - Platform-based synthesis & optimization
    - Full consideration of physical reality

Advantages – Better Complexity Management

- Shorter verification/simulation cycle
  - Simulation speed 100X faster than RTL-based method [NEC, ASPDAC04]
- Significant code size reduction
  - RTL design ~300KL → Behavioral design 40KL [NEC, ASPDAC04]

<table>
<thead>
<tr>
<th>Design</th>
<th>C lines</th>
<th>VHDL lines</th>
<th>LE</th>
<th>Fmax(MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>90</td>
<td>600</td>
<td>1349</td>
<td>178.7</td>
</tr>
<tr>
<td>MCM</td>
<td>161</td>
<td>1260</td>
<td>2402</td>
<td>152.6</td>
</tr>
<tr>
<td>CACHE</td>
<td>293</td>
<td>1277</td>
<td>371</td>
<td>161.0</td>
</tr>
<tr>
<td>MOTION</td>
<td>130</td>
<td>1200</td>
<td>888</td>
<td>161.2</td>
</tr>
<tr>
<td>IDCT</td>
<td>236</td>
<td>7388</td>
<td>9351</td>
<td>162.9</td>
</tr>
<tr>
<td>DWT</td>
<td>180</td>
<td>1371</td>
<td>1862</td>
<td>147.3</td>
</tr>
<tr>
<td>EDGELOOP</td>
<td>329</td>
<td>7296</td>
<td>7440</td>
<td>100.1</td>
</tr>
</tbody>
</table>

- VHDL code generated by UCLA xPilot targeting Altera Stratix platform
- Over 10x code size reduction can be achieved
**Advantages – Rapid System Exploration (1)**

- Quick evaluation of various amounts of process level concurrency and different hardware/software boundaries

Example: Motion-JPEG implementation
- All HW implementation
- All SW implementation (using embedded processors)
- SW/HW co-design: optimal partitioning?
  - Repeated manual RTL coding is not solution!

---

**Advantages – Rapid System Exploration (2)**

- Fast exploration of multiple micro-architecture alternatives
  - Different hardware implementations can be easily obtained by varying the high-level spec. and applying different design constraints

<table>
<thead>
<tr>
<th>Target cycle time</th>
<th>State#</th>
<th>Fmax (MHz)</th>
<th>Cycle#</th>
<th>Latency (ns)</th>
<th>LE#</th>
<th>DSP#</th>
</tr>
</thead>
<tbody>
<tr>
<td>9ns</td>
<td>34</td>
<td>123.56</td>
<td>4830</td>
<td>39.1</td>
<td>1777</td>
<td>128</td>
</tr>
<tr>
<td>7ns</td>
<td>36</td>
<td>147.28</td>
<td>5211</td>
<td>35.4</td>
<td>1862</td>
<td>128</td>
</tr>
<tr>
<td>5.5ns</td>
<td>51</td>
<td>183.62</td>
<td>6926</td>
<td>37.8</td>
<td>1926</td>
<td>128</td>
</tr>
</tbody>
</table>

- Platform: Altera Stratix
- RTL synthesis & place-and-route: Altera QuartusII v5.0
- Simulation: Mentor ModelSim SE6.0
**Advantages – Higher Quality of Results (1)**

- Platform-based synthesis & optimization
  - The quality of a RTL design is platform-dependent
  - Designers often lack the complete and detail knowledge of the target platform

<table>
<thead>
<tr>
<th>Resource</th>
<th>Area</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDSUB-24b</td>
<td>25 LUTs</td>
<td>2.27</td>
</tr>
<tr>
<td>ADDSUB-32b</td>
<td>33 LUTs</td>
<td>2.61</td>
</tr>
<tr>
<td>MUX8to1-24b</td>
<td>120 LUTs</td>
<td>2.92</td>
</tr>
<tr>
<td>MUX16to1-24b</td>
<td>264 LUTs</td>
<td>4.658</td>
</tr>
<tr>
<td>DSPMUL-18bx18b</td>
<td>2 DSP Blocks</td>
<td>3.833</td>
</tr>
<tr>
<td>DSPMUL-24bx24b</td>
<td>8 DSP Blocks</td>
<td>7.688</td>
</tr>
</tbody>
</table>

- Platform: Altera Stratix
- RTL synthesis & place-and-route: Altera QuartusII v5.0

**Motivation – Higher Quality of Results (2)**

- Communication-centric synthesis & optimization with full consideration of physical reality
  - System performance & power is dominated by interconnect
  - It is difficult for designers to consider physical layout at the RT level

![Layout-aware performance optimization](Diagram1)

Overlapping computation with communication

![Layout-aware power optimization](Diagram2)

Overlapping computation with communication

- Binding solution 1: Both multipliers keep active
- Binding solution 2: 
  - mul₁ can be powered off when false branch is taken
**xPilot: Behavioral-to-RTL Synthesis Flow**

- Behavioral spec. in C/SystemC
- Frontend compiler
- SSDM
- RTL + constraints
- FPGAs/ASICs

**Presynthesis optimizations**
- Loop unrolling/shifting
- Strength reduction / Tree height reduction
- Bitwidth analysis
- Memory analysis ...

**Core synthesis optimizations**
- Scheduling
- Resource binding, e.g., functional unit binding register/port binding

**μArch-generation & RTL/constraints generation**
- Verilog/VHDL/SystemC
- FPGAs: Altera, Xilinx
- ASICs: Magma, Synopsys, ...

**SystemC-to-RTL Compilation Flow**

- SystemC specification
- xPilot front-end
- SystemC elaboration
  - AST
  - Netlist in XML
  - Behavioral IR (CDFG)
- SSDM
- Platform description

**xPilot synthesis engine**
- Output files (Timing/Area, RT VHDL & Constraints)


**Restricted Behavioral C Subset**

- **Data types:**
  - Primitive integer types: char, byte, short, int, long...
  - One-dimension arrays of primitive integer types

- **Operations:**
  - All arithmetic and logic operations: +, -, *, /, >>, &, ..., *

- **Control flow statements:**
  - while, for, switch-case, if-then-else, break, continue, return, ...

**Restricted Behavioral C Subset (cont.)**

- **Unsynthesizable**
  - Recursions
  - Pointers
  - Dynamic memory allocations and system calls
  - Irregular jumps, e.g., gotos
**System-level Synthesis Data Model**

- **SSDM (System-level Synthesis Data Model)**
  - Hierarchical netlist of concurrent processes and communication channels

- Each leaf process contains a sequential program which is represented by an extended LLVM IR with hardware-specific semantics
  - Port / IO interfaces, bit-vector manipulations, cycle-level notations

**Hardware-Specific SSDM Semantics**

- **Process port/interface semantics**
  - FIFO: `FifoRead()` / `FifoWrite()`
  - Buffer: `BuffRead()` / `BuffWrite()`
  - Memory: `MemRead()` / `MemWrite()`

- **Bit-vector manipulation**
  - Bit extraction / concatenation / insertion
  - Bit-width attributes for every operation and every value

- **Cycle-level notation**
  - Clock: `waitClockEvent()`
Platform Modeling & Characterization

Target platform specification

- High-level resource library with delay/latency/area/power curve for various input/bitwidth configurations
  - Functional units: adders, ALUs, multipliers, comparators, etc.
  - Connectors: mux, demux, etc.
  - Memories: registers, synchronous memories, etc.

- Chip layout description
  - On-chip resource distributions
  - On-chip interconnect delay/power estimation

Scheduling – Problem Statement

- Scheduling problem in behavioral synthesis
  - Given:
    - A control data flow graph (CDFG) which captures the behavior of the input description
    - A set of scheduling constraints: resource constraints, latency constraints, frequency constraints, relative IO timing constraints, etc.
  - Goal:
    - Assign the operations to control states so that a particular design objective (performance / power) is optimized while all the constraints are satisfied.

- Highlights of our scheduling engine
  - Applicable to a wide range of application domains
    - Computation-intensive, memory-intensive, control-intensive, partially timed, etc.
  - Offers a variety of optimization techniques in a unified framework
    - Operation chaining, behavioral template, relative scheduling, physical layout consideration, etc.
Scheduling — Overall Approach

**Overall approach**

- Current objective: high-performance
- Use a system of integer difference constraints to express all kinds of scheduling constraints
- Represent the design objective in a linear function

**Dependency constraint**

\[ v_1 \to v_2 : x_2 - x_3 \geq 0 \]
\[ v_2 \to v_3 : x_2 - x_4 \geq 0 \]
\[ v_3 \to v_5 : x_2 - x_5 \geq 0 \]
\[ v_4 \to v_5 : x_3 - x_5 \geq 0 \]

**Frequency constraint**

\[ (v_2, v_3) : x_3 - x_2 \geq 1 \]

**Resource constraint**

\[ (v_2, v_3) : x_3 - x_2 \geq 1 \]

**Platform characterization:**
- adder (+/-) 2ns
- multiplier (*): 5ns

**Target cycle time:** 10ns

**Resource constraint:** Only ONE multiplier is available

\[
\begin{bmatrix}
1 & 0 & -1 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 1 & -1 \\
0 & 1 & 0 & 0 & -1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
\leq
\begin{bmatrix}
0 \\
-1 \\
0 \\
0 \\
-1
\end{bmatrix}
\]

**Totally unimodular matrix:** guarantees integral solutions

Scheduling — Design Framework

**CDFG**

- **xPilot scheduler**
- **Constraint equations generation**
  - Relative timing constraints
  - Dependency constraints
  - Frequency constraints
  - Resource constraints...
- **Objective function generation**
- **System of pairwise difference constraints**
- **Linear programming solver**
- **LP solution interpretation**

**Target platform modeling**
- (resource library & chip layout)

**User-specified design constraints & assignments**

**STG (State Transition Graph)**
**Unified Resource Binding**

- An efficient architectural exploration framework
- Simultaneous functional unit, register, and port binding
- Emphasize on the interconnect and steering logic networks
- Guided by a flexible cost evaluation engine to achieve different objectives, e.g., performance, area, power, etc.
- Extendable to exploit physical layout information

---

**Resource Binding—Problem Statement**

- **Resource binding problem**
  - *Given:* (1) A scheduled control data flow graph, i.e., STG; (2) Design constraints: performance, delay, or power, etc.
  - *Goal:* Assign the operations and variables to functional units and register, respectively, so that their executions or lifetimes are not conflicted, and all of the design constraints are satisfied.

- **Properties of the problem**
  - FU and register binding are highly correlated
  - Simultaneous FU and register binding considering interconnection is very difficult

---

**Two binding solutions:**

- *Which one is better?*
- *The answer depends on:*
  1. How large are the MUX and ALU (platform-dependent)
  2. Performance and area constraints
**Distributed Register-File (DRF) Microarchitecture**

- Regular datapath structure
- Provides opportunities to hide large MUX into register-files
- Computations and communications are localized
  - Allow replicated values among islands
  - Enables efficient optimizations to control interconnects among islands

**Advantages of DRF Microarchitecture**

- Discrete register result
- MUX implementation may be very expensive (e.g., on FGPAs)

- DRF result:
  - Datapath with more regularity
  - Hide MUX into the register file
  - Especially effective for FPGA designs
**Platform-Based Interface Synthesis**

- **Focus on sequential communication channels**
  - Data must be read and written in the same order
    - Example: FIFO (FSL in VirtexII), Bus (in both Stratix and Virtex)
  - Order may have dramatic impact on performance
    - Best order should guarantee that no data transmission on critical path are delayed by non-critical transmission
- **Interface synthesis for sequential communication channels**
  - Consider both the behavior model and communication topology to detect the optimal transmission order
  - Automatically do interface generation for sequential communication units, as well as code transformation for behavior models

---

**Overall Approach to Interface Synthesis**

- **Reduce the order detection problem to a min-latency scheduling problem:**
  - Merge the CDFGs of all processes
  - Each element to be transferred on FIFO are transformed to a special operation T
  - Only one T can be scheduled at each step.
- **Example shown on right, assuming only 1 cycle is needed for FIFO operation**

Scheduling result, order is (1,3,2)
Power Optimization – Architecture Exploration

- Developed a quantitative evaluation framework for design of power-efficient FPGAs using multi-Vdd/Vth for power reduction [FPGA’03, FPGA’04, ISLPED’04, TCAD’05]
  - Voltage island methodology for ASIC type designs
  - High Vt for configuration transistors (not speed critical)
  - Programmable Vdds for FPGA type designs
  - Evaluation of
    - Multiple Vdd/Vth selection
    - Granularity of voltage islands

- Evaluation of clock gating & power gating
  - Study the right gating granularity
  - Work together with multiple Vdd/Vth architecture

Power Optimization – Synthesis for a Fixed Architecture

- Novel algorithms to minimize both dynamic and static power
  - Consider both temporal and physical locality information
  - Carry out simultaneous scheduling, binding, and placement
  - Study the interdependency and interaction between architecture designs and high-level synthesis
  - Discover a large stream of architecture alternatives and their impacts on power optimization

- Control-flow intensive design power optimization
  - Utilization profiling for different computation blocks
  - Smart assignment algorithms to assign blocks into gating regions and/or voltage islands

- Multiplexer optimization, memory optimization, and speculated execution, etc. for power minimization
Example: Functional Unit Binding with Voltage Assignment

Given:
- A scheduled data flow graph (DFG)
- A module (functional unit) library with dual Vdds
- The Vdd of each module can be changed dynamically while executing different operations

Goal:
- Assign low Vdd to the maximum number of operations with switching-activity consideration
- Minimize total switching power through functional unit binding

Constraint:
- Latency constraint
- Resource constraint

Motivational Example

Resource = 3 multipliers, and latency = 9 control steps
Which set of operations to extend?
- Honor data dependency
- Maximum number under latency and resource constraints
- The best such set in terms of switching-activity reduction during FU binding later on
- Need to consider voltage assignment and FU binding simultaneously to achieve optimal solution
**Optimal Solution based on Network Flow Transformation**  
*Chen, Cong, Xu, ASPDAC’05*

Comparability graph $G_c$

- $L = 100$
- $C(v_i, v_j) = -L \times (1 - W_{ij})$
- $T = L \times |V_c|$
- $-T$ for maximum number of extensions

Flow network $N_G$ with two Vdds

---

**Experimental Results — Benchmark Suite**

- **Benchmark suite**
  - PR, MCM:
    - DSP kernels: pure additions/subtractions and multiplications
  - CACHE
    - Cache controller: control-intensive designs with cycle-accurate I/O operations
  - MOTION:
    - Motion compensation algorithm for MPEG-1 decoder: control-intensive with modest amount of computations
  - IDCT:
    - JPEG inverse discrete cosine transform: computation intensive
  - DWT:
    - JPEG2000 discrete wavelet transform: computation intensive with modest control flow
  - EDGELOOP:
    - Extracted from H.264 decoder: a very complex design, features a mix of computation, control, and memory accesses
**SystemC/C-to-FPGA Design Flow (Altera)**

- SystemC/C specification
- Front-end compiler
- xPilot behavioral synthesis
- SSDM (System-Level Synthesis Data Model)
- SSDM/CDFG
- Behavioral synthesis
- SSDM/FSMD
- RTL generation
- FSM with Datapath in VHDL
- Floorplan and/or multi-cycle path constraints
- Altera QuartusII v5.0
- Stratix/StratixII device configurations

**Experimental Results – Altera**

<table>
<thead>
<tr>
<th>Designs</th>
<th>Line Count</th>
<th>Resource Usage</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>VHDL</td>
<td>LE</td>
</tr>
<tr>
<td>PR</td>
<td>90</td>
<td>600</td>
<td>1349</td>
</tr>
<tr>
<td>WANG</td>
<td>90</td>
<td>727</td>
<td>1105</td>
</tr>
<tr>
<td>LEE</td>
<td>141</td>
<td>696</td>
<td>1585</td>
</tr>
<tr>
<td>MCM</td>
<td>161</td>
<td>1260</td>
<td>2402</td>
</tr>
<tr>
<td>DIR</td>
<td>190</td>
<td>1352</td>
<td>3489</td>
</tr>
</tbody>
</table>

- Device setting: Stratix
- Target frequency: 200 MHz
Experimental Results – Comparison with SPARK on Altera Stratix FPGA

- SPARK [UCI/UCSD, 2004], a state of the art academic high-level synthesis tool

<table>
<thead>
<tr>
<th>Designs</th>
<th>SPARK Resource Usage</th>
<th>xPilot Resource Usage</th>
<th>Fmax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LE</td>
<td>COMB</td>
<td>Lonely-Reg</td>
</tr>
<tr>
<td>PR</td>
<td>1108</td>
<td>815</td>
<td>0</td>
</tr>
<tr>
<td>WANG</td>
<td>1217</td>
<td>942</td>
<td>0</td>
</tr>
<tr>
<td>LEE</td>
<td>1367</td>
<td>1052</td>
<td>0</td>
</tr>
<tr>
<td>MCM</td>
<td>2908</td>
<td>2248</td>
<td>0</td>
</tr>
<tr>
<td>DIR</td>
<td>2425</td>
<td>2034</td>
<td>0</td>
</tr>
<tr>
<td>Ave Ratio</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- On average, xPilot resource binding achieves designs with similar area, and 1.68x higher frequency over Spark

SystemC/C-to-FPGA Design Flow (Xilinx)

- SystemC/C specification
- Front-end compiler
- xPilot behavioral synthesis
- SSDM (System-Level Synthesis Data Model)
- SSDM/CDFG
- Behavioral synthesis
- RTL generation
- FSM with Datapath in VHDL
- Floorplan and/or multi-cycle path constraints
- Xilinx ISE i7.1
- VirtexII-Pro/Virtex-4 device configurations
**Experimental Results – Xilinx**

<table>
<thead>
<tr>
<th>Designs</th>
<th>Line Count</th>
<th>Resource Usage</th>
<th>Fmax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>VHDL</td>
<td>Slices</td>
</tr>
<tr>
<td>PR</td>
<td>90</td>
<td>600</td>
<td>331</td>
</tr>
<tr>
<td>WANG</td>
<td>90</td>
<td>727</td>
<td>357</td>
</tr>
<tr>
<td>LEE</td>
<td>141</td>
<td>696</td>
<td>356</td>
</tr>
<tr>
<td>MCM</td>
<td>161</td>
<td>1260</td>
<td>887</td>
</tr>
<tr>
<td>DIR</td>
<td>190</td>
<td>1352</td>
<td>979</td>
</tr>
</tbody>
</table>

- Device setting: xc2vp30 -7
- Target frequency: 200 MHz

**Synthesis from Behavior to DRF**

- Data import logic is the most “critical”
  - Operations bound to an island form a “chain” in DFG
  - Optimize complexity of inter-island connections
  - Min-cut chain partitioning → improve design quality

<table>
<thead>
<tr>
<th>Micro-Architecture</th>
<th>Slices</th>
<th>LUT</th>
<th>FF</th>
<th>RAM Blk#</th>
<th>MUL</th>
<th>Clock Period (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHEN-24 DRF (6 islands)</td>
<td>425</td>
<td>808</td>
<td>86</td>
<td>6</td>
<td>6</td>
<td>12.04</td>
</tr>
<tr>
<td>Base Line</td>
<td>798</td>
<td>928</td>
<td>1,235</td>
<td>0</td>
<td>66</td>
<td>10.37</td>
</tr>
<tr>
<td>PR-24 DRF (5 islands)</td>
<td>457</td>
<td>860</td>
<td>63</td>
<td>5</td>
<td>3</td>
<td>10.62</td>
</tr>
<tr>
<td>Base Line</td>
<td>701</td>
<td>858</td>
<td>1,076</td>
<td>0</td>
<td>48</td>
<td>11.69</td>
</tr>
</tbody>
</table>

- Device: Xilinx Virtex II -6; Target clock period: 10ns
- Observations: Large area (Slices and Multiplier blocks) reduction by using on-chip RAM blocks to implement register files, with small impact on Fmax
Initial Results of Interface Synthesis

- Target for sequential communication channels
  - In particular, FSL in VirtexII
- Consider two communicating processes
- 20+% performance improvement on average by optimizing communication ordering

Architecture Model for Low-Power FPGAs with Dual Vdds

- Dual-Vdd configuration on functional units
- Suitable to reduce both dynamic and static power in the data path
- Our model and algorithm can be extended to more than two Vdds
**Experimental Results (1)**

**Dual-Vdd/Single-Vdd Power and Energy Reduction Compared to the Base Case (Single-Vdd + 0% Latency Relaxation)**

**Experimental Results (2)**

**Power Reduction Percentages Compared to the Single-Vdd Case along Latency Relaxation**
Outline

- Motivation
- xPilot system framework
- Behavior-level synthesis in xPilot
  - Advantages of behavioral synthesis
  - Scheduling
  - Resource binding
- System-level synthesis in xPilot
  - Synthesis for ASIP platforms
  - Design exploration for heterogeneous MPSoCs
- Conclusions

Design Exploration for Heterogeneous MPSoC Platforms

- Heterogeneous MPSoCs exploration
  - Processors
    - Heterogeneous vs. homogeneous
    - General-purpose vs. application-specific
  - On-chip communication architecture (OCA)
    - Bus (e.g. AMBA, CoreConnect), packet switching network (e.g. Alpha 21364)
  - Memory hierarchy
**Configurable SoC Platforms**

- **General purpose processor cores + programmable fabric**
  - Tight integration using extended instructions (ASIPs)
    - Example: Altera Nios / Nios II
  - Loose integration using FIFOs/busses for communications
    - Example: Xilinx MicroBlaze, etc.

![Diagram of Nios II Embedded Processor](source: www.altera.com)

**ASIP Compilation: Problem Statement**

- **Given:**
  - CDFG $G(V, E)$
  - The basic instruction set $I$
  - Pattern constraints:
    - Number of inputs $|P(i)| \leq Nin$
    - Number of outputs $|P(O(i))| = 1$
    - Total area $\sum_{V \in V} area(p) < A$

- **Objective:**
  - Generate a pattern library $P$
  - Map $G$ to the extended instruction set $I \cup P$, so that the total execution time is minimized

![Diagram of ASIP Compilation](source: www.xilinx.com)
**Target Core Processor Model**

- **Core processor model**
  - Classic single-issue pipelined RISC core (fetch / decode / execute / mem / write-back)
    - The number of input and output operands of an instruction is pre-determined
    - An instruction reads the core register file during the execute stage, and commits the result during the write-back stage

**ASIP Compilation Flow**

- **Front-end compilation**
  - C code
  - CDFG
  - 1. Pattern generation
  - 2. Pattern selection
  - Pattern library
  - 3. Application mapping & Graph covering
    - Optimized CDFG

- **Backend compilation**
  - Optimized assembly

- **Pattern Generation**
  - Satisfying input/output constraints

- **Pattern Selection**
  - Select a subset to maximize the potential speedup while satisfying the resource constraint

- **Application Mapping**
  - Graph covering to minimize the total execution time
Experimental Results on Altera Nios

- Altera Nios is used for ASIP implementation
  - 5 extended instruction formats
  - up to 2048 instructions for each format
- Small DSP applications are taken as benchmark

<table>
<thead>
<tr>
<th>Extended Instruction/</th>
<th>Speedup</th>
<th>Resource Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Estimation</td>
<td>Nios</td>
</tr>
<tr>
<td>fft br</td>
<td>9</td>
<td>3.29</td>
</tr>
<tr>
<td>iir</td>
<td>7</td>
<td>3.18</td>
</tr>
<tr>
<td>fir</td>
<td>2</td>
<td>2.40</td>
</tr>
<tr>
<td>pr</td>
<td>2</td>
<td>1.57</td>
</tr>
<tr>
<td>dir</td>
<td>2</td>
<td>3.28</td>
</tr>
<tr>
<td>mcm</td>
<td>4</td>
<td>4.75</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>3.08</td>
</tr>
</tbody>
</table>

Architecture Extension for ASIPs

- Data bandwidth problem
  - Limited register file bandwidth (two read ports, one write port)
  - ~40% of the ideal performance speedup will be lost
- Shadow-register-based architectural extension
  - Core registers are augmented by an extra set of shadow registers
    - Conditionally written during write-back stage
    - Low power/area overhead
  - Novel shadow-register binding algorithms are developed
Problem Statement: Mapping for Heterogeneous Integration with Multiple Processing Cores

◆ Given:
  - A library of processing cores $L$
  - Task graph $G(V, E)$
    - For each $v$ in $V$, execution time $t(v, p_i)$ on $p_i$
    - For each $(u, v)$ in $E$, communication data size $s(u,v)$
  - Cost (area/power) constraint $C$

◆ Problem:
  - Select and instantiate the processing elements from $L$
  - Generate the on-chip communication architecture and topology
  - Map the tasks onto the processing elements so that
    - The total latency is minimized while the final implementation cost is less than $C$

Preliminary Results on Motion-JPEG Example

Encoded JPEG Images

Xilinx XUP Board

<table>
<thead>
<tr>
<th>System</th>
<th>Cycle#</th>
<th>Fmax (MHz)</th>
<th>Exe Time (ms)</th>
<th>Area (Slice#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model #1</td>
<td>23812</td>
<td>126</td>
<td>0.189</td>
<td>4306</td>
</tr>
<tr>
<td>Model #2</td>
<td>14800</td>
<td>126</td>
<td>0.117</td>
<td>6345</td>
</tr>
</tbody>
</table>

Model #1: 5 Microblazes
FSL-based communication

Model #2: 4 Microblazes + DCT on FPGA fabrics
**Conclusions**

- xPilot can automatically synthesize behavior level C or SystemC presentation to RTL code with necessary design constraints
- Platform-based synthesis with physical planning provides
  - Shorter verification/simulation cycle
  - Better complexity management, faster time to market
  - Rapid system exploration
  - Higher quality of results
- xPilot can help to explore the efficient use of (multiple) on-chip processors
- xPilot can efficiently optimize the software for reconfigurable processors
- We are interested to engage with selected industrial partners to further validate and enhance the technology

**Acknowledgements**

- We would like to thank the supports from
  - National Science Foundation (NSF)
  - Gigascale Systems Research Center (GSRC)
  - Semiconductor Research Corporation (SRC)
  - Industrial sponsors under the California MICRO programs (Altera, Xilinx)
- Team members:
  - Yiping Fan
  - Guoling Han
  - Wei Jiang
  - Zhiru Zhang