Increasing Design Productivity Through Core Reuse, Meta-Data Encapsulation and Synthesis

Adam Arnesen
Kevin Ellsworth
Derrick Gibelyou
Travis Haroldsen
Jared Havican
Marc Padilla

Brent Nelson
Michael Rice
Michael Wirthlin

2 Sept 2010
Introduction - Traditional Core Reuse

**Pros**
- Productivity increase through reuse
- Pre-verified IP
- Difficult to integrate

**Cons**
- Difficult to integrate cores
- Not accessible for domain expert

**IP Cores in HDL**

**Manually wrap and interconnect IP**

**Top Level Design**

**Simulate and Debug**

**Synthesize and Download**
Introduction – SoC (Bus-based) Core Reuse

- Productivity increase through automated core reuse
  - Productivity increases primarily observed in System-on-Chip (SoC)
  - Utilizes standard bus interconnect
  - Meta-data used to encapsulate low-level details of core interface.

- Several design tools exist
  - Xilinx EDK
  - Altera SOC Builder
  - Mentor Graphics Platform Express
Introduction – General Core Reuse

- Reuse is more difficult when using cores with non-standard interfaces.
  - Point to point
  - Broadcast
  - Handshaking
  - Timed Data Exchanges

- How do we automatically reuse cores with arbitrary interfaces?
Addressing General Core Reuse

- Machine-Readable Characterization
  - Parameterization
  - Low-level interface details
  - Timing and protocol information

- CAD Tool
  - Understand characterization
  - Automatically generate control and interconnect circuitry
  - Create fully integrated finished design
Overview of our Approach

- Parameterization and Meta-data Representation
  - Low-level parameterization
  - Dependent Parameterization
  - Domain-Specific (high-level) parameterization
  - IP-XACT and Extensions
  - Temporal Parameterization

- Library of Cores

- The Design Environment (Ogre CAD Tool)

- Results
  - Manual Radio Construction
  - Automatic Radio Generation
Parameterization and Reuse

- 1 core used in multiple designs
- Simplify for domain expert
- Requires robust metadata descriptions
Low-Level Parameterization

- Traditional method of parameterizing IP
  - Bit-width
  - Operating mode
- Can be difficult to reuse for a non-hardware designer
- Done by leveraging standard IP-XACT parameterization
High-Level Parameterization (New)

Loop Filter Parameterization

- Accumulation Width: 32
- Loop Bandwidth: 0.01
- Loop Damping Factor: 1.0
- Phase Detector Gain: 6.0
- DDS Gain: -1.0
- Samples Per Symbol: 2
- K Precision: 44
- Order: 2

VHDL (from library)

IP-XACT XML with CHREC Extensions
Dependent Parameterization

- Dependences in parameters allow high-level parameters to be translated into low-level parameters
- Leverage IP-XACT mathematical expressions (XPath)
- Leverage VHDL Functions in modules
- Further allows parameterization targeted to a domain expert

<spirit:value
spirit:resolve="dependent"
spirit:dependency="(id('operandlength')
div id('wordlength')) + 1"
spirit:id="opreg_numwords">147
</spirit:value>
IP-XACT and Extensions

- Naming Convention
- Port Description
  - Naming
  - Direction
  - Vector-based bit widths
- Parameterization
  - Mathematical Dependencies
  - Choice Dependencies
- File Sets
- Bus-based interconnect standards
- Extensions are allowed in various areas in the schema
Temporal Parameterization

- Homogeneous Synchronous Data Flow (HSDF) model of Computation
  - Latency
  - Data Introduction Interval (II)
  - Sample Delay
Library of Cores: Descriptions

- Many versions of blocks in library
- Versions differ by:
  - Latency
  - Throughput
  - Area
- Differences allow for design-space exploration

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Latency (cycles)</th>
<th>Block Delay (ns)</th>
<th>Max Freq. (MHz)</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slices</td>
<td>DSPs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cubic Interpolators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>43.8</td>
<td>365</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>54.6</td>
<td>164</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>43.1</td>
<td>185</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>34.5</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Farrow Interpolator</td>
<td>0</td>
<td>*</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Decision (QPSK)</td>
<td>0</td>
<td>0.9</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Timing Error Detectors</td>
<td>0</td>
<td>9.6</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12.5</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10.5</td>
<td>284</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop Filters</td>
<td>0</td>
<td>11.1</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18.0</td>
<td>167</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18.0</td>
<td>223</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCO</td>
<td>0</td>
<td>3.1</td>
<td>319</td>
<td></td>
</tr>
<tr>
<td>Calculate Mu</td>
<td>0</td>
<td>1.7</td>
<td>567</td>
<td></td>
</tr>
<tr>
<td>Phase Error Detectors</td>
<td>0</td>
<td>5.5</td>
<td>182</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5.9</td>
<td>338</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clockwise Rotations</td>
<td>0</td>
<td>5.4</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5.3</td>
<td>371</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDS</td>
<td>1</td>
<td>8.5</td>
<td>235</td>
<td></td>
</tr>
</tbody>
</table>

All data collected targeting a Xilinx Virtex4 device
* Data currently unavailable
Design Environment Tool Flow

1. Schematic Capture Tool (Simulink)
2. BYU Netlist
3. Generate Design VHDL
4. Generate FSM VHDL
5. Complete Radio
6. Full Radio Design in VHDL
7. Commercial Synthesis tool-flow
8. Virtex4 bit file

Synthesis Tool Flow

- Validate Parameters
- Schedule
- Resolve Bitwidths
- Schedule Dependency Graph (IMS)
- Data Dependency Graph
- Generate Data Dependency Graph
- Propagate High to Low-level Parameters
- BYU Extensions

IP-XACT
- DDS
- NCO
- Loop Filter
- PED
- BYU Extensions
Simulink-supported Design Entry

1) Instance Library Blocks

2) Instanced blocks are fully parameterizable

3) Add Interconnect

4) Clock and Reset ports do not need to be connected

5) Access CHREC synthesis tools from Simulink
Synthesis Approach

1) Design with basic blocks
2) Create global timing controller
3) Insert buffer and control registers
4) Insert data type conversions
5) Generate top-level synthesizable VHDL
Synthesis Approach (Controller)

- Create schedule
  - Align data dependencies
  - Align iteration loops
- Create timing controller (FSM)
  - Clock Enable
  - Data Valid
- Insert registers
**Synthesis Approach** (Data Types)

- **Convert Low level types**
  - Ex: Signed → std_logic_vector

- **Propagate bitwidths**

- **Match high level types to target**
  - Lab VIEW FPGA
  - System Generator
  - VHDL
Generate Output Files

- Synthesizable Design VHDL
- FSM VHDL
- Individual Core VHDL
- LabVIEW FPGA Files
  - CLIP Node XML
  - VHDL Wrapper
- System Generator Testbench Files
Radio Test Setup

Bit Error Rate Detector

QPSK Constellation

Noise Generator

Signal Generator

XtremeDSP FPGA Board (Virtex4)
Radio Performance

- Radios implemented and tested
  - 2.5 Msymbols/sec @ 70 MHz (IF)
  - QPSK (1 cycle)
    - 5 MHz clk
  - QPSK (15 cycle)
    - 75 MHz clk
  - QPSK (15 cycle) performed a little better due to wider bit widths
Productivity Results

- Begin with common library of reusable parameterized IP cores
- Create 2 QPSK designs by hand in VHDL
  - Days per radio
- Create many QPSK + other radio variations
  - Hours/radio
Conclusion

- Model of computation
- Parameterized IP Cores
- Meta-data descriptions

CAD Tool
- Understand Meta-data
- Synthesis algorithms
- Generate controllers
- Create interface logic

SIGNIFICANT IMPROVEMENTS IN DESIGN PRODUCTIVITY

Unified IP Reuse System
QUESTIONS?