Rapid Architectural Exploration: Using Faster Design to Implement Faster Designs (Faster!)

MAPLD  September 15, 2008
A few assertions...

- **Fast design = faster designs**
  - The best architecture determines the fastest, smallest, lowest latency, highest performance design
  - Architectural exploration is key to achieving the best architecture

- **RTL ≠ fast design**
  - Hardware is concurrent, with shared resources
  - You cannot design hardware faster (generally) without raising the level of concurrency: especially complex algorithms, control logic and system composition

---

Atomic transactions: the only high-level abstraction for concurrency in hardware design

Faster design: faster implementation, faster changes, fewer bugs, powerful parameterization

Better designs: faster, smaller, lower latency, highest performance. You pick!
What is Bluespec?

- **Hardware design language equivalent of the advance from assembly language to C/C++**
  - Commensurate productivity & capability improvements over RTL, while keeping designers in 100% control of architecture
  - Synthesizable language extensions (BSV) to SystemVerilog
  - Bluespec Compiler (BSC) generates synthesizable Verilog and cycle-accurate models from BSV

- **General purpose solution**
  - Applicable to all levels of detail – from executable spec to implementation
  - Applicable to all component types – datapath, control, state machines, interconnect, transactors, testbenches, models, …
  - Seamless environment unifies architecture/modeling, implementation & verification

- **Proven: world-class systems and semiconductor companies providing impressive proof points**
  - Architectural exploration, synthesizable testbenches, RTL replacement
  - Processors, wireless, video, multimedia, memory controllers, …

And others (e.g. large microprocessor company, large computer systems company, large search company, …)
Bluespec is the next step in hardware abstraction

**Bluespec high-level language (BSV)**
- Raises level of thinking, allowing bigger, more complex problems to be attacked
- Eliminates tedium of low-level hardware control logic implementation
- Enables quick and easy design changes to try multiple alternative implementations
- Retains architectural expression for the QoR of hand-coded RTL

**Bluespec high-level compiler (BSC)**
- Quickly and correctly generates low-level control logic
- Eliminates a large number of bugs that would otherwise occur in low-level logic
- Supports end-user tool extensions through design database API

High-level language
\[ \xrightarrow{\text{BSV}} \]

High-level compiler
\[ \xrightarrow{\text{BSC}} \]

Low-level language
\[ \xrightarrow{\text{RTL code}} \]

Low-level compiler
\[ \xrightarrow{\text{Synthesizer}} \]

Logic gates
\[ \xrightarrow{\text{Netlists}} \]
Rapid Architectural Exploration Enables the Search for the Global Optimum
Architectural exploration is the first order of optimization

With limited or no architecture changes, your scope of optimization is limited to finding a local minimum.

Rapid, safe architectural changes enable a much wider scope of architectures to be explored for a more optimal solution.

Bluespec makes it possible to find the global power, area, timing and latency minimums.
Let’s Look at an Example

Using Bluespec for rapid FPGA development & high performance
2008 MEMOCODE Codesign Contest

Speedup the sorting of large lists of encrypted data.

27 commercial and research teams started the four week contest.

9 solutions were delivered.
The traditional approaches

2008 MEMOCODE Codesign Contest Results

Normalized Speedup

Reference: http://rijndael.ece.vt.edu/memocodesign08/everybodywins/

team sunita (2)  team eric  team rob  team vijay  team sunita (1)  team uljana  team marco  team brian

C + Impulse C  C + HDL  C

blue spec
But, what if you could design so quickly and cleanly in 4 weeks that you could:

- Put 100% in hardware (including the complex control)
- Skip system-level simulation and jump straight to FPGA
- And, still explore architectures
Bluespec: an unfair advantage

2008 MEMOCODE Codesign Contest Results

Normalized Speedup

Reference: http://rijndael.ece.vt.edu/memocontest08/everybodywins/

Bluespec-based team Kermin beat second place by an order of magnitude (>10X)
Core Technology:
Atomic Transactions
Bluespec’s core technology: atomic transactions, the only high-level abstraction for HW concurrency

For decades: in Operating Systems, Databases, Distributed Systems

Recently: for software for multi-core/multi-threaded architectures

"I think we ultimately will see atomic transactions in most, if not all, languages. That's a bit of a guess, but I think it's a good bet."

Burton Smith, Technical Fellow, Parallel Computing

Very recently: HW support for Transactional Memory in processors
Atomic transactions drive rapid architectural exploration

Two very unique, atomic-transaction-powered drivers:

- Control-adaptive parameterization & powerful “generate” capabilities enable a designer to create a single specification that can describe a family of architectures

- When the specification changes or is used to generate a specific architecture, the detailed control logic is automatically rebuilt and the design constraints (types, connectivity, scheduling) are automatically rechecked so correct functionality is achieved much sooner

Faster Expression & Changes

Fewer Bugs

Quicker Analysis
Hardware is highly concurrent
What makes hardware so:
• Error-prone
• Brittle
• Complex
• Hard to develop, verify
• Change?
Coordinating all the accesses to the shared resources!
Coordinating access to shared resources requires detailed, low-level implementation of muxes, arbiters and scheduling logic.
Bluespec generates the arbitration & control logic to coordinate access to shared resources.
Which lets you think about the HW…
...like this:
...and this:
…and this:
...and this:
…NOT like this:
Bluespec automates all of this:

(100% under your control and without any extra logic)
...making hardware so much:
  • Less buggy
  • More flexible
  • More scalable
  • Simpler
  • More reusable
  • Less costly to develop & verify
Reed Solomon Results
Abhinav Agarwal, Alfred Ng

WiMAX requirement is to support a throughput of 134Mbps

<table>
<thead>
<tr>
<th></th>
<th>Bluespec</th>
<th>Xilinx IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent Gate Count</td>
<td>267,741</td>
<td>596,730</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>108.5</td>
<td>91.2</td>
</tr>
<tr>
<td>Steady State (Cycles/Block)</td>
<td>276</td>
<td>2073</td>
</tr>
<tr>
<td>Data rate (Mbps)</td>
<td>701.3</td>
<td>89.7</td>
</tr>
</tbody>
</table>

For the same area!

Lower is better
Higher is better

Source: Arvind, 2008 DAC HLS Workshop, “HLS as an Enabling Technology: Some Complex Examples”
Atomic transactions make Bluespec general purpose, practical, and highly beneficial – a unique combination

**IPs done in BSV (and with good QoR)**

Models of:
- “RISC” processor
- MIPS
- Itanium
- PowerPC
- ARM

- L2 cache ctrl
- Distributed cache coherence
- DDR2 ctrl
- SRAM ctrl

**System Bus**

- Processor
- DRAM
- Memory Controller
- DSP/accelerators

- OCP
- AXI
- AHB

- Network proc
- Queuing engines
- Sorting queue
- Arbiter
- IP lookup
- Debug controller

**Peripheral Bus**

- Serial Controller
- Audio
- Video
- Flash/Mem I/F
- Bus Controller

- PCI Express, USB
- I2C, MIPI HSI,
- MIPI Unipro

- FIR filter
- Pixel processor
- Waveform generator
- Pong

Bluespec SystemVerilog (BSV)

- Complex Datapaths (e.g. processor/controller)
- Control

**Loop/array Algorithms**

C-based synthesis
Bluespec Summary

Atomic transactions:
the only high-level abstraction for concurrency in hardware design

Faster design:
faster implementation, faster changes, fewer bugs, powerful parameterization

Better designs:
faster, smaller, lower latency, highest performance. You pick!
Key differences over RTL

Bluespec SystemVerilog

Behavioral
Correct: Concurrency and Communications
Atomic transactions (rules) and interface methods simplify complex control & concurrency:
• Across multiple shared resources
• Across module boundaries

Structural
Correct: Construction and Configurability
• Control-adaptive, extreme:
  - Parameterization
  - “generate” capability
• High-level types closer to spec
• Powerful static checking & formal semantics
• Advanced clock management

VHDL/Verilog/SystemVerilog/SystemC
Simple example with concurrency and shared resources

Each register can only be updated by one process on each clock

Process priority: $2 > 1 > 0$

Verilog

```verilog
always @posedge CLK begin
  if (!cond2 && cond1) x <= x - 1;
  else if (cond0) x <= x + 1;
  if (cond2) y <= y - 1;
  else if (cond1) y <= y + 1;
end
```

Bluespec SystemVerilog

```verilog
(* descending_urgency = "proc2, proc1, proc0" *)

rule proc0 (cond0);
  x <= x + 1;
endrule

rule proc1 (cond1);
  y <= y + 1;
  x <= x - 1;
endrule

rule proc2 (cond2);
  y <= y - 1;
endrule
```

What’s required to verify that each is correct?

What if the priorities changed: $\text{cond1} > \text{cond2} > \text{cond0}$?

What if the processes are in different modules?
What happens when you change the spec?

Verilog

```
always @(posedge CLK) begin
  if ((cond2 && cond0) ||
      (cond0 && !cond1 &&
       !cond3))
    x <= x + 1;
  else if (cond3 && !cond2)
    x <= x + 2;
  else if (cond1 && !cond2)
    x <= x - 1
  if (cond2)
    y <= y - 1;
  else if (cond3)
    y <= y - 2;
  else if (cond1)
    y <= y + 1;
end
```

Bluespec SystemVerilog

```
(* descending_urgency = "proc2, proc3, proc1, proc0" *)

rule proc0 (cond0);
  x <= x + 1;
endrule

rule proc1 (cond1);
  y <= y + 1;
  x <= x - 1;
endrule

rule proc2 (cond2);
  y <= y - 1;
endrule

rule proc3 (cond3);
  y <= y - 2;
  x <= x + 2;
endrule
```

Hand-written RTL:
- Complexity due to:
  - State-centric (for synthesizability)
  - Scheduling clutter
  - Brittle to change

BSV:
- Functional correctness follows directly from rule semantics
- Executable spec (operation-centric)
- Automatic handling of shared resource mux logic
- Same hardware as the RTL
- Changes are rapid and much safer!
Bluespec AzureIP™ for Bus Fabrics

Standard bus protocols, AMBA® AXI® & AHB and OCP, abstracted to…

High-level data type transactions

Designers interact with simple Get/Put transactional I/Fs

…high-level transactions & data types
Abstract Connections using advanced overloading

- Allows quick and easy assembling of systems

```verilog
interface CacheIfc;
    interface Server#(Req_t, Resp_t) ipc;
    interface Client#(Req_t, Resp_t) icm;
endinterface

module mkTopLevel (...)
    // instantiate subsystems
    Client #(Req_t, Resp_t) p <- mkProcessor;
    Cache_Ifc #(Req_t, Resp_t) c <- mkCache;
    Server #(Req_t, Resp_t) m <- mkMem;

    // instantiate connects
    mkConnection (p, c.ipc); // Server connection
    mkConnection (c.icm, m); // Client connection
endmodule
```

overloaded module
Implementation with rapid micro-architectural exploration

7 different micro-architectural implementations were created and explored within 5 days – and parameterized from a single design.

Control logic was automatically adapted and scheduled by the tool to support each approach – without impacting the adjacent blocks.

<table>
<thead>
<tr>
<th>802.11a Design (by IFFT block type)</th>
<th>Area (um^2)</th>
<th>Symbol Latency (cycles)</th>
<th>Throughput (clks/symbol)</th>
<th>Min frequency required (MHz)</th>
<th>Average Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combinational</td>
<td>4.91</td>
<td>10</td>
<td>4</td>
<td>1.0</td>
<td>3.99</td>
</tr>
<tr>
<td>Pipelined</td>
<td>5.25</td>
<td>12</td>
<td>4</td>
<td>1.0</td>
<td>4.92</td>
</tr>
<tr>
<td>Folded - 16 radix4</td>
<td>3.97</td>
<td>12</td>
<td>4</td>
<td>1.0</td>
<td>7.27</td>
</tr>
<tr>
<td>Folded - 8 radix4</td>
<td>3.69</td>
<td>15</td>
<td>6</td>
<td>1.5</td>
<td>10.9</td>
</tr>
<tr>
<td>Folded - 4 radix4</td>
<td>2.45</td>
<td>21</td>
<td>12</td>
<td>3.0</td>
<td>14.4</td>
</tr>
<tr>
<td>Folded - 2 radix4</td>
<td>1.84</td>
<td>33</td>
<td>24</td>
<td>6.0</td>
<td>21.1</td>
</tr>
<tr>
<td>Folded - 1 radix4</td>
<td>1.52</td>
<td>57</td>
<td>48</td>
<td>12.0</td>
<td>34.6</td>
</tr>
</tbody>
</table>

Optimal power

Original designer intuition