Evolution of HDLs (Hardware Description Languages) and Modeling Languages

- ~1985: Text-based RTL langs: Verilog & VHDL
- 1995: IEEE Verilog standards (also VHDL standards)
- 2001: IEEE
- 2004: SystemVerilog (Accellera)
- 2005: IEEE
- 2005: IEEE
- ~2000: SystemC (OSCI)

(RTL = Register-Transfer Level)
### The Bluespec saga (still playing!)

<table>
<thead>
<tr>
<th>Year</th>
<th>Institution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~1996</td>
<td>MIT Research</td>
<td>• Language:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- &quot;TRAC&quot; (Hoe, Arvind)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TRS-based language</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Flat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Basic hardware types</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Tools:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Synthesis to RTL</td>
</tr>
<tr>
<td>2000</td>
<td>Sandburst Corp</td>
<td>• Language:</td>
</tr>
<tr>
<td></td>
<td>10Gb/sec Router ASICs</td>
<td>- &quot;Bluespec Classic&quot; (Augustsson)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Proprietary language</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Very Haskell-like syntax</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Modules with TRS rules and guarded interfaces</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Haskell types, type classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Numeric type classes to express &quot;size&quot; constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Monadic &quot;collection&quot; for Control/Status registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Tools:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Synthesis to RTL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Simulator</td>
</tr>
<tr>
<td>2003</td>
<td>Bluespec, Inc.</td>
<td>• Language:</td>
</tr>
<tr>
<td></td>
<td>Tools for Modeling,</td>
<td>- &quot;Bluespec SystemVerilog&quot; (Nikhil, Stoy, et. al.)</td>
</tr>
<tr>
<td></td>
<td>Design and</td>
<td>- All the Bluespec Classic features, but embedded in IEEE SystemVerilog</td>
</tr>
<tr>
<td></td>
<td>Verification</td>
<td>- Syntax with some extensions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- More sophisticated rule scheduling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Types and static analysis for multiple clock discipline</td>
</tr>
<tr>
<td>2006</td>
<td></td>
<td>• Tools:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Pure TRS-level simulator (free)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Simulator (clocked)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Synthesis to RTL</td>
</tr>
</tbody>
</table>

### Bluespec tools

- **SystemC [ESE]**
  - Parsing
  - Static Checking
  - Optimization
  - Scheduling
  - Power Optimization
  - RTL Generation

- **BSV**
  - Rapid, Source-Level Simulation and Interactive Debug of BSV

- **Bluespec Synthesis**
  - RTL
  - Cycle-Accurate w/Verilog sim

- **Bluespec Simulation**
  - Cycle-Accurate w/Verilog sim
The saga

- Epic in duration
  - (like a Bollywood movie!)
- Not yet epic in impact
- What does it take for wide adoption?

The adoption process

- Technical presentation in front of engineering managers, senior engineers
- If pass, do an “eval”
  - Training: 3 days of lectures and labs
  - Project: 2-6 weeks, with close support
- If pass ... maybe do another eval ... and another eval ...
- If pass, adopt for a small project
- ... the process can take months, years
Getting an audience for initial technical presentations

- Is not (too) hard, because every company is in a “productivity squeeze”, and knows they have to do something:
  - Skyrocketing costs for chip development, as Moore’s law increases complexity of what’s on a chip
  - Time-to-market windows shrinking

Problem: how to do an “elevator pitch” for productivity improvement?

- Raising the level of abstraction wins “in the large”, not on small toy examples

- But how do you demonstrate this during an “elevator pitch”?
  - Need small but convincing examples
  - With lots of explicit verbal reinforcement!
Simple example with concurrency and shared resources

- Process 0: increments register \( x \) when \( \text{cond0} \)
- Process 1: transfers a unit from register \( x \) to register \( y \) when \( \text{cond1} \)
- Process 2: decrements register \( y \) when \( \text{cond2} \)
- Each register can only be updated by one process on each clock. Priority: 2 > 1 > 0

Which one is correct?

What’s required to verify that they’re correct?
What if the priorities changed: \( \text{cond1} \) > \( \text{cond2} \) > \( \text{cond0} \)?
What if the processes are in different modules?
With Bluespec, the design is direct

(* 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

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

BSV:
Functional correctness follows directly from rule semantics (atomicity)
Executable spec (operation-centric)
Automatic handling of shared resource mux logic
Same hardware as the RTL

Now, let’s make a small change: add a new process and insert its priority

Process priority: 2 > 3 > 1 > 0
Changing the Bluespec design

Pre-Change

(* descending_urgency = "proc2, proc1, proc3" *)
rule proc0 (cond0);
x <= x + 1;
endrule
rule proc1 (cond1);
y <= y + 1;
x <= x - 1;
endrule
rule proc2 (cond2);
y <= y - 1;
endrule

(* 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;
x <= x + 1;
endrule
rule proc3 (cond3);
y <= y - 2;
x <= x + 2;
endrule

Changing the Verilog design

Pre-Change

always @(posedge CLK) begin
  if (cond2)
    x <= x - 1;
  else if (cond1)
    y <= y - 1;
  else if (cond3)
    x <= x + 1;
  if (cond2)
    y <= y - 1;
  else if (cond1)
    y <= y + 1;
end

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

**Executable specifications**

**RTL**

```plaintext
data_in data_out
push_req_n full
pop_req_n empty
clk
rst
```

**Bluespec**

```plaintext
FIFO
```

A small sample of the informal, written interface specification:

An error occurs if a push is attempted while the FIFO is full.

Thus, there is no conflict in a simultaneous push and pop when the FIFO is full. A simultaneous push and pop cannot occur when the FIFO is empty, since there is no pop data to prefetch. However, push data is captured in the FIFO.

A pop operation occurs when pop_req_n is asserted (LOW), as long as the FIFO is not empty. Asserting pop_req_n sets the 'empty' response to 0 and issues it on the next rising edge of clk. Thus, the RAM read data must be captured on the clk's following the assertion of pop_req_n.

**Executable interface specification:**

```plaintext
interface FIFOBuf#(x_type);
  method ActionValue#(x_type) deq();
endinterface
```

Method conditions and method scheduling analysis capture all the “protocol” constraints on the left.

---

**Elevating Design above RTL**

**Bluespec SystemVerilog**

**Behavioral**

Correct: Concurrency and Communications

- Rules and interface methods for complex control and concurrency:
  - Across multiple shared resources
  - Across module boundaries

**Fully synthesizable**

**Structural**

Correct: Construction and Configurability

- High-level types closer to spec
- Much more powerful “generate”
- Powerful parameterization
- Powerful static checking & formal semantics
- Advanced clock management

**VHDL/Verilog/SystemVerilog/SystemC**
Example: a butterfly switch (crossbar)

Basic building blocks:

Recursive construction: 1x1 \(\rightarrow\) 2x2 \(\rightarrow\) 4x4 \(\rightarrow\) \(\cdots\) \(\rightarrow\) \(N\times N\)

Butterfly switch: code excerpts

```
interface XBar #(type t);
    interface List#(Put#(t)) input_ports;
    interface List#(Get#(t)) output_ports;
endinterface
```

- The choice of the type \((t)\) of the data deferred to later
- Intuitive high-level structures for the interface:
  - Sub-interfaces (hierarchical)
  - Aggregation (lists, vectors of interfaces)
Butterfly switch: code excerpts

```
module mkXBar #(Integer logn, ...
    if (logn == 0) ... // BASE CASE
    else ... // RECURSIVE CASE
    endmodule: mkXBar
```

Parameters chosen at compile time:

- **Size**: `logn`
- **Comb. circuit**: `destinationOf`
- **Module**: `mkMerge2x1`
  - Encapsulates flow-control, arbitration, queueing behavior of the 2x1 merge
- **Interface, `XBar#(t)`**, instead of port lists
- **Type (t) of data externally defined**

Butterfly switch: code excerpts

```
module mkXBar #(Integer logn, ...
    if (logn == 0) ... // BASE CASE
    else ... // RECURSIVE CASE
    endmodule: mkXBar
```

- Arbitrary, powerful “generate” capability (here: conditional, recursion, loop)
- All constructs can be leveraged as parameters, objects or returned from functions (modules, interfaces, rules, ...)

---

**Parameters chosen at compile time:**

- **Size**: `logn`
- **Comb. circuit**: `destinationOf`
- **Module**: `mkMerge2x1`
  - Encapsulates flow-control, arbitration, queueing behavior of the 2x1 merge
- **Interface, `XBar#(t)`**, instead of port lists
- **Type (t) of data externally defined**
Butterfly switch

Key Implementation Notes:
- Advanced parameterization & extremely powerful “generate”

Parameters:

- The switch itself: < 60 lines of BSV code
- First working (tested) prototype: < 1 day (including simple testbench)
- Fully synthesizable:
  Synthesized to layout (Magma, TSMC 0.18u, 550 MHz)
  (see also whitepaper for full code)

Clocks and clock-gating

- Most chips today involve multiple clocks, and use gated clocks to reduce power consumption

- In Bluespec, use types, type-checking, and other static analysis to ensure clock discipline:
  - Circuits only talk to “same-clock” circuits
  - Communication between clock-domains always uses a “safe” intermediary (synchronizing circuit)
  - Clock gating integrated with rule and method conditions
  - [Stoy, Nanavati, Czeck, …]
Power management

- Power consumption is a major consideration in today’s chips, for mobile and machine-room applications

- In Bluespec, we exploit semantics of Rules to automatically insert gating circuits in the HW to eliminate unnecessary switching activity (and therefore reduce power)

Elevating Design above RTL

Bluespec SystemVerilog

- Behavioral
- Structural

Correct: Concurrency and Communications
Rules and interface methods for complex control and concurrency:
- Across multiple shared resources
- Across module boundaries

Correct: Construction and Configurability
- High-level types closer to spec
- Much more powerful “generate”
- Powerful parameterization
- Powerful static checking & formal semantics
- Advanced clock management

VHDL/Verilog/SystemVerilog/SystemC
Performance: no compromises tolerated

- Generated RTL must match hand-coded RTL in
  - silicon area
  - clock speed
  - power consumption

- Must be better for
  - Debugging, maintenance, reuse, modifiability for timing closure, ...

Language needs to be understood at multiple levels

- Types and type-checking (Haskell-like)
- Static elaboration (functional computation)
- Rule execution (pure, “untimed”)
- Clock-scheduled rule execution
How much time to let this all sink in?

- In University, we teach these concepts in semesters/quarters/terms/years
  - And students are motivated (by grades, if nothing else)
- With customers, we have 3 days
  - (only the basics possible)
  - And students may not be motivated at all
    - May be “assigned” by their managers
    - Any available objection, to avoid change

Objections: minor

- “I don’t get it”
- “We don’t need types. Hardware only uses bits!”
- “These look like software features; it can’t be relevant for hardware”
- “I can mentally translate your examples into VHDL”
- “What’s an atomic transaction?”
- “What’s an invariant?”
- “I get it, but it’s too hard for our engineers” (!)

Responses:
- Education, education, education
  - Internal education, too: all of your technical reps have to be able to engage in a sophisticated technical argument!
- Slowly, painfully, build public testimonials
- Fear: “Your competitor gets it!”
Objection: “it can’t work”

- “It works for your conveniently chosen examples; it won’t work for our applications”
- “In our small eval we completed the design in half the time. But there must be some gotchas we’ll encounter in real designs.”
- “Yes, my colleagues evaluated it, and liked it; but I don’t really trust them”
- Response:
  - Slowly, painfully, build a huge and diverse portfolio of successful designs, with competitive HW quality
    - Occasionally, much better than hand-coded RTL, because higher-level of abstraction suggest much better microarchitectures!

Designs with Bluespec

BlueSpec has been used for every design listed:

- Processor
- Memory Controller
- DDR2 ctrl SRAM ctrl
- L2 cache ctrl
- "RISC" processor
- MIPS
- Itanium
- ARM
- PowerPC
- Power Management
- IDCT
- Motion compensator
- DES
- MPEG-4
- XPT
- DDR2 ctrl SRAM ctrl
- AMBA
- DMA ctrl
- PCIe
dm
- arbitration
- Bus converters
- System Bus
- Controller
- "Behavioral synthesis"
- 402.11a
- Network proc
- Queuing engine
- Sorting engine
- ArtNet
- IP lookup
- Debug controller
- Peripheral Bus
- "DSP/processor"
- "Datapaths (e.g. processor/ controller)"
- Bus Control
- Complex Datapaths (e.g. processor/ controller)
Objection: “doesn’t fit”

- Need to interop with Verilog/VHDL
- Need to fit into an “ecosystem” of testbench and other tools
- Generated RTL needs to be readable!

- Response: Do enormous amount of engineering, write lots of training material, to meet these objections

Objection: “competing approaches”

- SystemVerilog itself (it’s new!)
- SystemC itself (it’s new!)
- Assertions (correctness properties in temporal logic)
- “Behavioral Synthesis”
  - Old medicine (vectorization, automatic parallelization) in new bottles, but people are unaware of the history and limitations
  - “It’ll succeed any day now”

- Response: education, education, education
  - Most of these are not “competing” at all, when you understand the details
  - Attend trade shows, get onto panel discussions, write trade press articles, be punchy
Objection: “non-standard”

• “It isn’t standard”

• Responses:
  • Reuse familiar languages as much as possible!
    • [Bluespec Classic would be a total non-starter!]
  • Remind tradeoff: standards vs. innovation!
    • Innovations, by definition, don’t start life as standards
  • Insist that you’re language is not proprietary:
    • Participate in standards bodies (IEEE P1800 SystemVerilog, IEEE P1666 SystemC)
      • *Enormous* investment of time!
    • Donate your language features to standards
      (“Tagged Unions and Pattern Matching” in IEEE P1800 SystemVerilog comes from Bluespec)
Objection: “you pipsqueak!”

- “We can’t risk our chip on tools from you, a small startup company”
- “This won’t help my resumé at all”

Responses:
- Punch back (“Ok, if you’re comfortable delaying your productivity improvements by 2 years”)
- Risk-mitigation measures (e.g., escrow your tool)
- Look for risk-takers in companies (“early adopters”)

Using Haskell to implement our tools

- This is a purely internal choice (not visible to customers)

- Hard to recruit people:
  - who know Haskell well, and
  - know chip design reasonably well

- But, enormous productivity!
  - Our engineering team has just 7 developers (+ interns)!
- And, enormous stability and reliability of our products
  - In a field where tools are notoriously flaky!

- Some battles with space leaks, black holes, ...
  (sometimes seriously affecting capacity of designs that can be handled)
Typical user experience

Stage
Intrigue
Decision to try Bluespec

Resistance

Acceptance

Enthusiasm

“I started with the most simple parts of my DMA design, and after 2 or 3 weeks of practice, I’m sure I was more efficient with standard RTL language.”

“I had to design the complex arbitrations and the data streaming, coupled with some weeks of practice, things became easier. I designed the more complex parts in a couple of weeks, with a compact code (compared to RTL), which seems easy to configure and to maintain.”

“We were, of course, faster than with RTL (let’s say 2 times) for the overall design + verification time. Having to design such a new IP, I would prefer to start design with Bluespec rather than with RTL.”

You have to find and cultivate such people in each company!

Where we stand today

Paying customers:

Texas Instruments
Nokia
ST
Analog Devices

plus several in advanced “eval” stage (after several dozens of evals).

University program: Serious use at
MIT, CMU, UT Austin, UTokyo, Indian Institute of Science, Virginia Tech, ...

(Your university, too?)
Summary

- It's a long, hard, grind
- Have to educate engineers and managers about how abstraction and rigor will benefit them
- Have to educate in detail about why your tool is different
- No compromise on generated RTL quality
- Today, finally, we have a toe-hold with a small number of influential customers
  - Let’s hope it “catches” and grows (before we run out of money!)
  - If it does, it’ll be spectacular—it’ll revolutionize HW design!

End

Thank you!

Questions: nikhil@bluespec.com