bluespec

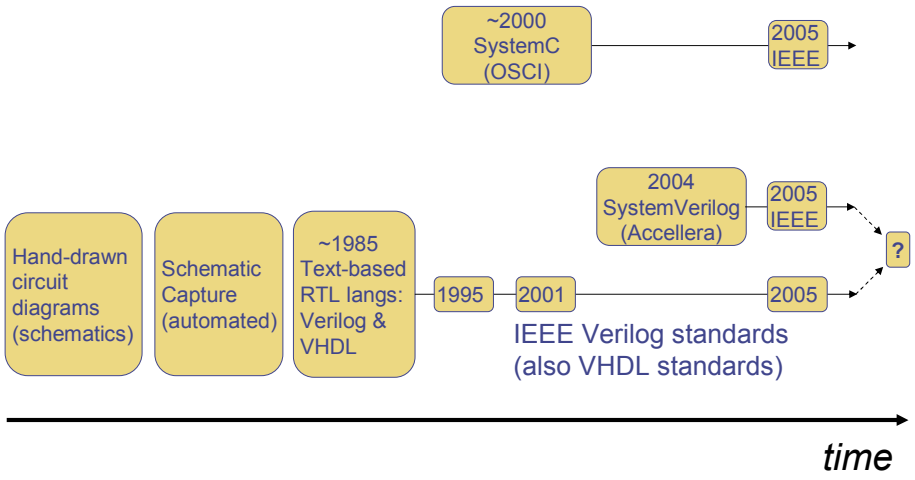
ESL SYNTHESIS

Bringing declarative programming into a commercial tool for developing integrated circuits

Rishiyur S. Nikhil
CTO, Bluespec, Inc.
www.bluespec.com, Waltham, Massachusetts

CUFP, September 21, 2006

Evolution of HDLs (Hardware Description Languages) and Modeling Languages



(RTL = Register-Transfer Level)

bluespec

The Bluespec saga (still playing!)

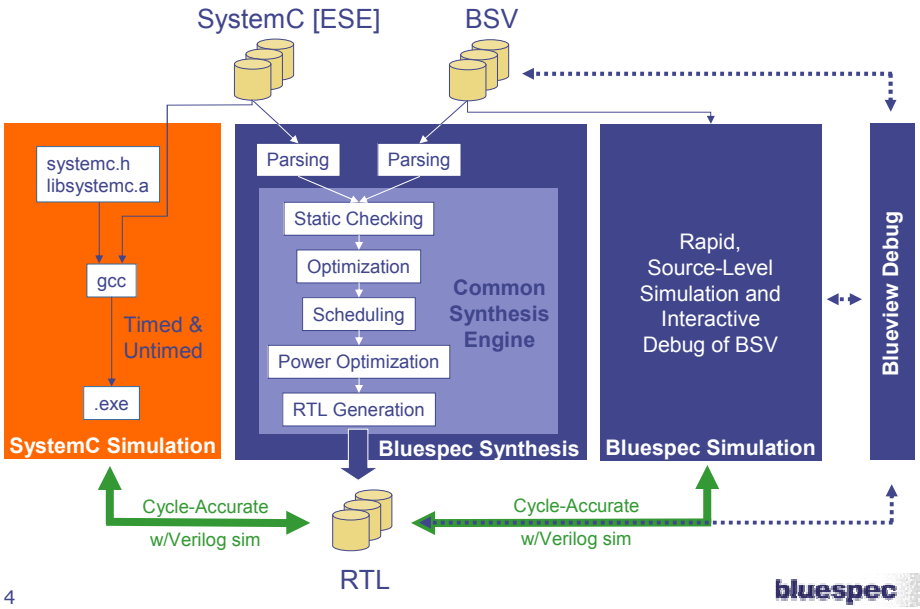
~1996 — 2000 — 2003 — 2006

MIT Research	Sandburst Corp 10Gb/sec Router ASICs	Bluespec, Inc. Tools for Modeling, Design and Verification	
Language: ▪ **"TRAC" [Hoe, Arvind]** ▪ TRS-based language ▪ Flat ▪ Basic hardware types	Language: ▪ **"Bluespec Classic" [Augustsson]** ▪ Proprietary language ▪ Very Haskell-like syntax ▪ Modules with TRS rules and guarded interfaces ▪ Haskell types, type classes ▪ Numeric type classes to express "size" constraints ▪ Monadic "collection" for Control/Status registers	Language: ▪ **"Bluespec SystemVerilog" [Nikhil, Stoy, et. al.]** ▪ All the Bluespec Classic features, but embedded in **IEEE SystemVerilog** syntax with some extensions ▪ More sophisticated rule scheduling ▪ Types and static analysis for multiple clock discipline	Language: ▪ **"ESE" [Nikhil, Mednick, Stoy, et. al.]** ▪ TRS extensions to **IEEE SystemC**
Tools: ▪ Synthesis to RTL	Tools: ▪ Synthesis to RTL ▪ Simulator	Tools: ▪ Synthesis to RTL ▪ Simulator	Tools: ▪ Pure TRS-level simulator (**free**) ▪ Simulator (clocked) ▪ Synthesis to RTL

3

bluespec

Bluespec tools



4

bluespec

The saga

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

bluespec

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*

bluespec

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

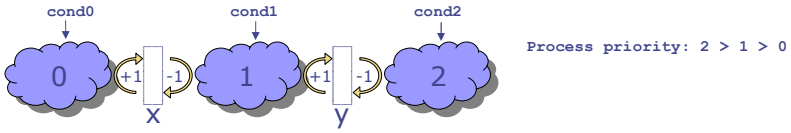
bluespec

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!

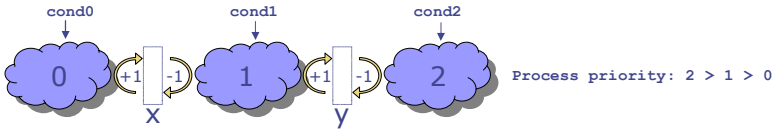
bluespec

Simple example with concurrency and shared resources



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

bluespec



Which one is correct?

```
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
```

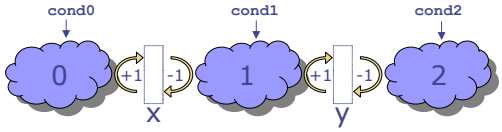
✓

```
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
```

What's required to verify that they're correct?
What if the priorities changed: cond1 > cond2 > cond0?
What if the processes are in different modules?

bluespec

With Bluespec, the design is direct



Process priority: 2 > 1 > 0

```
(* 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:

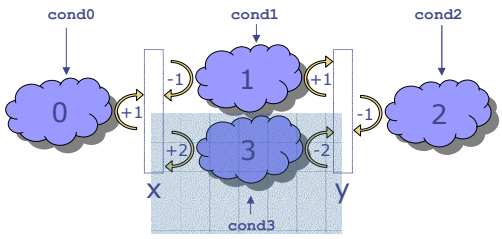
- ***State-centric (for synthesizability)***
- ***Scheduling clutter***

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***

bluespec

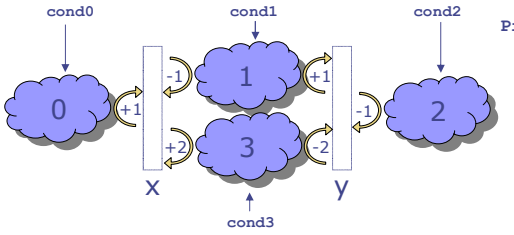
Now, let's make a small change: add a new process and insert its priority



Process priority: 2 > 3 > 1 > 0

bluespec

Changing the Bluespec design



Process priority: 2 > 3 > 1 > 0

Pre-Change

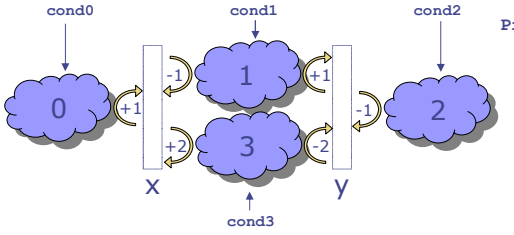
```
(* descending_urgency = "proc2, proc1, proc0" *)

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

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

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

```
(* 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
```

?

bluespec

Changing the Verilog design



Process priority: 2 > 3 > 1 > 0

Pre-Change

```
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
```

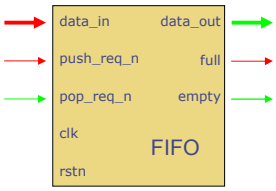
```
always @(posedge CLK) begin
  if ((cond2 && cond0) || (cond0 && !cond1 && !cond3))
    x <= x + 1;
  else if (cond3 && !cond2)
    x <= x + 2;
  else if (cond1 && !cond
    x <= x - 1

  if (cond2)
    y <= y - 1;
  else if (cond3)
    y <= y - 2;
  else if (cond1)
    y <= y + 1;
end
```

?

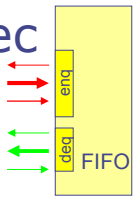
bluespec

Interface Methods: Executable specifications

RTL



Bluespec



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 causes the internal read pointer to be incremented on the next rising edge of clk. Thus, the RAM read data must be captured on the clk following the assertion of pop_req_n.

Executable interface specification:

```
interface FIFOBuf#(x_type);
 method Action                enq(x_type x);
 method ActionValue#(x_type) deq();
endinterface
```

Method conditions and method scheduling analysis capture all the "protocol" constraints on the left

bluespec

Elevating Design above RTL

Bluespec SystemVerilog

Behavioral

Fully synthesizable

Structural

Correct: Concurrency and Communications

Rules and interface methods for complex control and concurrency:

- Across multiple shared resources
- Across module boundaries

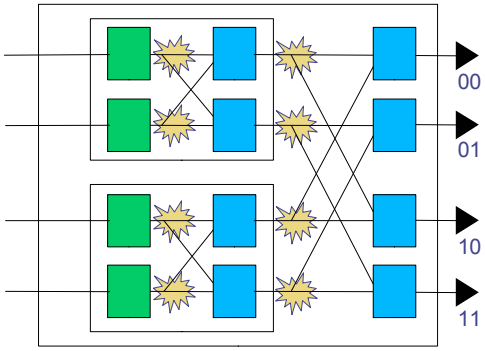
Correct: Construction and Configurability

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

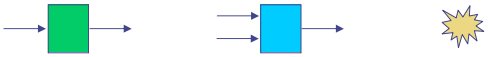
VHDL/Verilog/SystemVerilog/SystemC

bluespec

Example:
a butterfly switch (crossbar)



Basic building blocks:



Recursive construction: 1x1 ➔ 2x2 ➔ 4x4 … ➔ NxN

bluespec

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)

bluespec

Butterfly switch: code excerpts

```
module mkXBar #(Integer logn,
                function Bit #(32) destinationOf (t x),
                module #(Merge2x1 #(t)) mkMerge2x1)
               (XBar #(t))
   …
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

bluespec

Butterfly switch: code excerpts

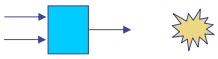
```
module mkXBar #(Integer logn, …)
   if (logn == 0) …  // BASE CASE
      FIFO#(t) f <- mkFIFO;
      …
   else … // RECURSIVE CASE
      XBar#(t) upper <- mkXBar (logn-1, …);
      XBar#(t) lower <- mkXBar (logn-1, …);
      …
      for (Integer j = 0; j < n; j = j + 1) …
         rule route; …
            if (! flip) merges [j]       .iport0.put (x);
            else        merges [jFlipped].iport1.put (x);
         endrule
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, …)

bluespec

Butterfly switch

Key Implementation Notes:
- Advanced parameterization & extremely powerful "generate"

Parameters: Type of data NxN 

- 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

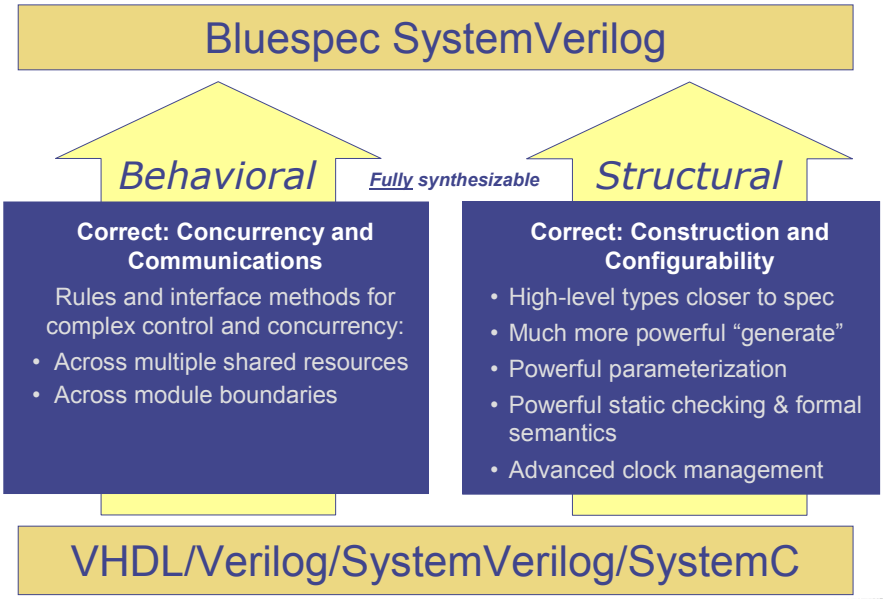
- Most chips today involve multiple clocks, and use gated clocks to reduce power consumption
- In Bluespec, use types, type-checking, and other static analysis to ensure clock discipline:
  - Circuits only talk to "same-clock" circuits
  - Communication between clock-domains always uses a "safe" intermediary (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)

bluespec

Elevating Design above RTL



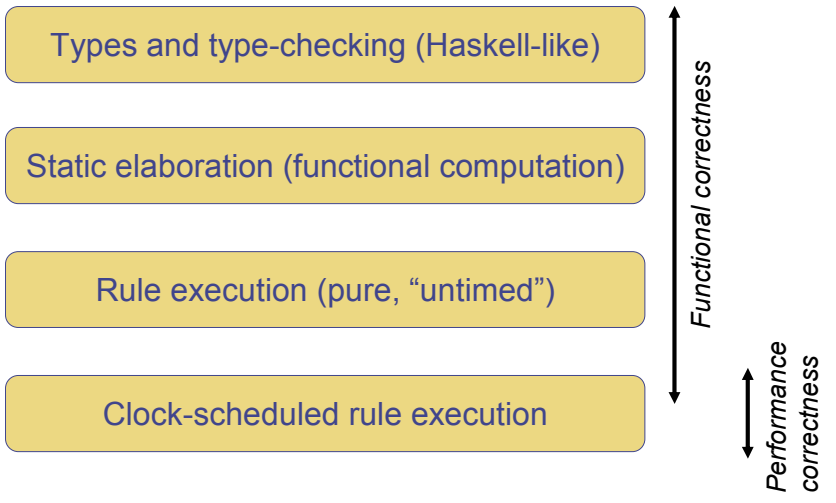
bluespec

Performance: no compromises tolerated

- Generated RTL *must* match hand-coded RTL in
  - silicon area
  - clock speed
  - power consumption
- Must be *better* for
  - Debugging, maintainance, reuse, modifiability for timing closure, …

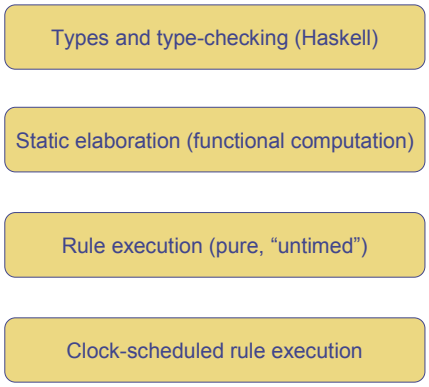
bluespec

Language needs to be understood at multiple levels



bluespec

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

bluespec

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!"

bluespec

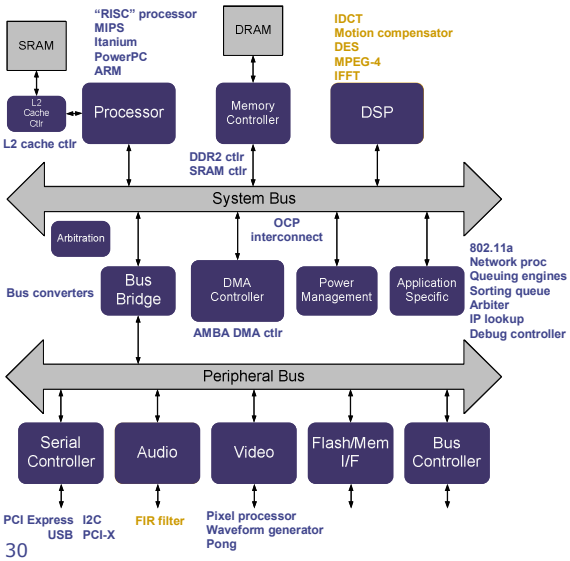
Objection: "it can't work"

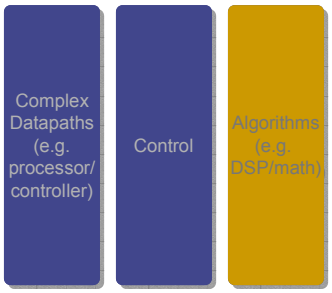
- "It works for your conveniently chosen examples; it won't work for our applications"
- "In our small eval we completed the design in half the time. But there must be some gotchas we'll encounter in *real* designs."
- "Yes, my colleagues evaluated it, and liked it; but I don't really trust them"
- Response:
  - 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!

bluespec

Designs with Bluespec

Bluespec has been used for <u>every</u> design listed:





"Behavioral synthesis"

bluespec

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

bluespec

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

bluespec

bluespec

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)

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")

bluespec

Using Haskell to implement our tools

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

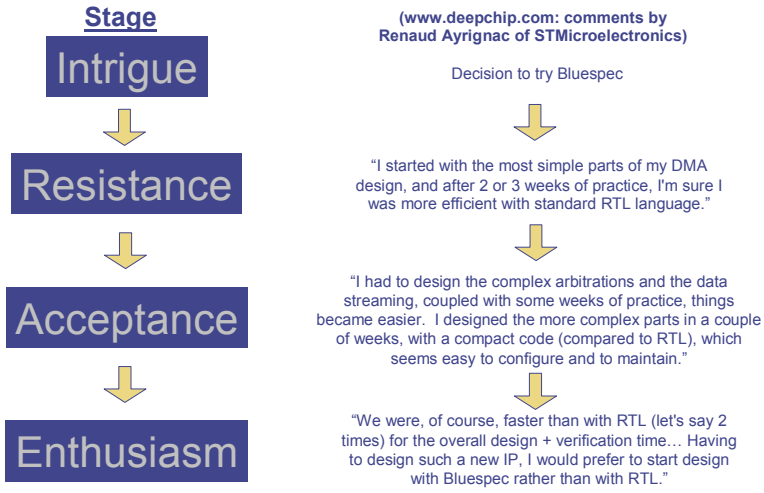
- Hard to recruit people:
  - who know Haskell well, *and*
  - know chip design reasonably well

- 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)

bluespec

Typical user experience



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

bluespec

Where we stand today

Paying customers:

Texas Instruments NOKIA Connecting People 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?)

bluespec

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!

bluespec

End

Thank you!

Questions: nikhil@bluespec.com

bluespec