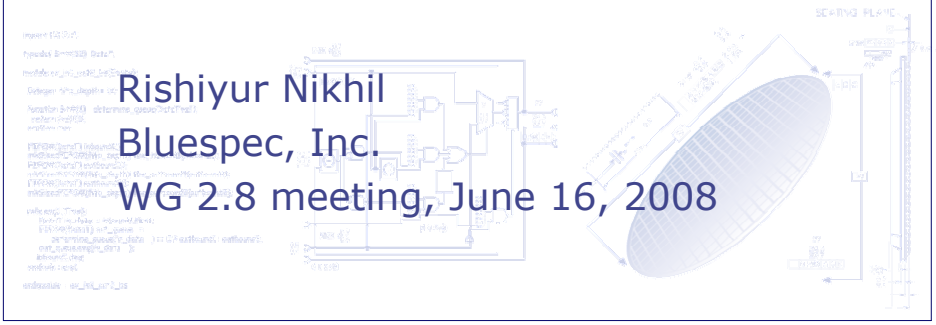
bluespec

ESL SYNTHESIS

On "Control Adaptivity": how to sell the idea of Atomic Transactions to the HW design community



Rishiyur Nikhil
Bluespec, Inc.
WG 2.8 meeting, June 16, 2008

Background: The BSV language (Bluespec SystemVerilog) is founded on composable Atomic Transactions (atomic rewrite rules).

This talk is about how we're slowly finding the vocabulary, the voice, the story to sell the idea to the HW design community (it's taken several years!).

bluespec

Arguments that go over like a ton of bricks

Atomic Transactions are good because ...

...they're new; modern; cool; different

What's "atomic"?

Thank you, but I don't have the time

... it helps you reason about correctness using invariants

What do you mean by "reason"? What's an "invariant"? What's "compositional"?

... they give compositional concurrency

... they give consistent access to your shared resources

I don't use "shared resources" (!!!)

... see how elegantly they solve the Dining Philosophers problem!

... see how elegantly they solve the Santa Claus problem!

Cute! But I can't see the relevance to what I do.

... they're good for your soul

Do I have a "soul"?

bluespec

3

In summary ...

Atomic Transactions are good because ...

... *(summarizing slides that follow)* they make your designs *control-adaptive*. And here's how that helps you in writing executable specs, in modeling, in refining models to designs, in reusable designs, and in maintainable and evolvable designs.

Ah! I see!

And maybe you should also tell me about atomicity, reasoning, invariants, compositionality, shared resources!

... and don't forget about dining philosophers, Santa Claus and my soul!

bluespec

4

GCD in BSV

```
module mkGCD (I_GCD);
   Reg#(int) x <- mkRegU;
   Reg#(int) y <- mkReg(0);

   rule swap ((x > y) &&  (y != 0));
       x <= y;  y <= x;
   endrule
   rule subtract ((x <= y) && (y != 0));
       y <= y - x;
   endrule

   method Action start(int a, int b) if (y==0);
      x <= a;  y <= b;
   endmethod
   method int result() if (y==0);
       return x;
   endmethod
endmodule
```

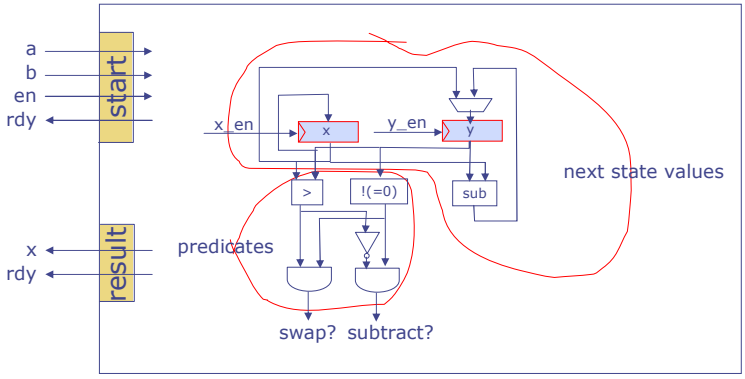
State

Internal behavior

External interface

bluespec

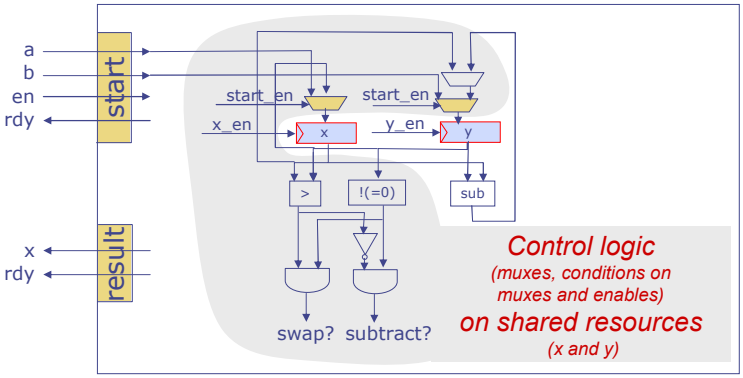
Generated Hardware



x_en = swap?
y_en = swap? OR subtract?

bluespec

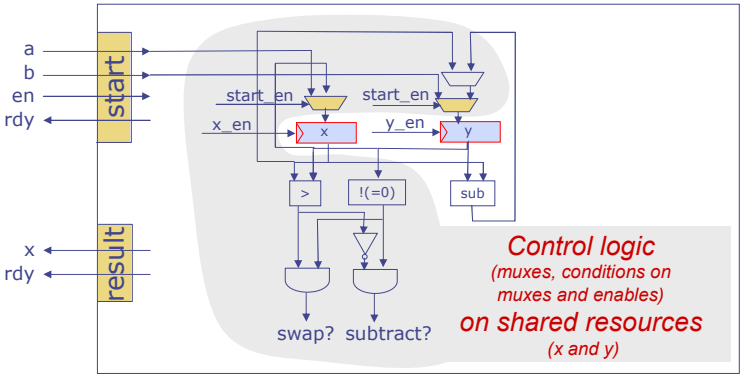
Generated Hardware Module



x_en = swap? OR start_en
y_en = swap? OR subtract? OR start_en
rdy = (y==0)

bluespec

Generated Hardware Module



When coding this in Verilog, this control logic is explicit:

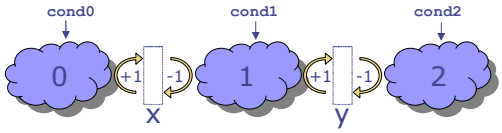
```
always
   @(posedge CLK)
   if (start_en) x <= a;
   else if (swap_en) x <= y;
```


bluespec

Summarizing, we have Point 1:

BSV's Atomic Transactions (rules) are just another way of specifying control logic that you would have written anyway

bluespec



Resource-access scheduling logic i.e., control logic

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

  if (cond0 && !cond1)
    x <= x + 1;
end
```

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

  if (cond0 && (!cond1 || cond2) )
    x <= x + 1;
end
```

Better scheduling

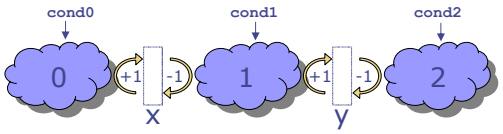
** There are other ways to write this RTL, but all suffer from same analysis*

Fundamentally, we are scheduling three potentially concurrent atomic transactions that share resources.

Note that control of x is affected by a "non-adjacent" condition (cond2), because of atomicity of the intervening process 1. This is fundamentally what makes RTL so fragile!

bluespec

In BSV

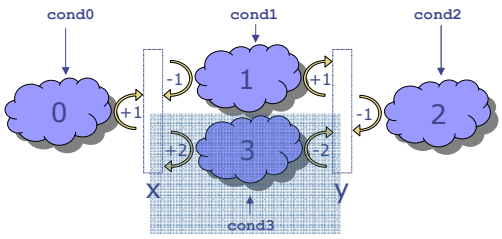


Process priority: 2 > 1 > 0

```
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, proc1, proc0" *)
```


bluespec

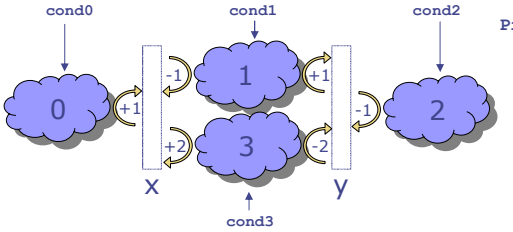
Now, let's make a small change: add a new process and insert its priority



Process priority: 2 > 3 > 1 > 0

bluespec

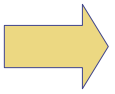
Changing the BSV design



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
```

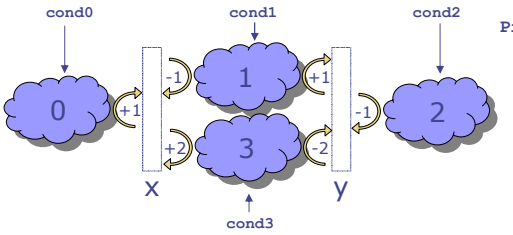


```
(* descending_urgency = "proc2, proc3, proc1, proc0" *)

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

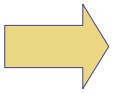
Changing the Verilog design



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
```

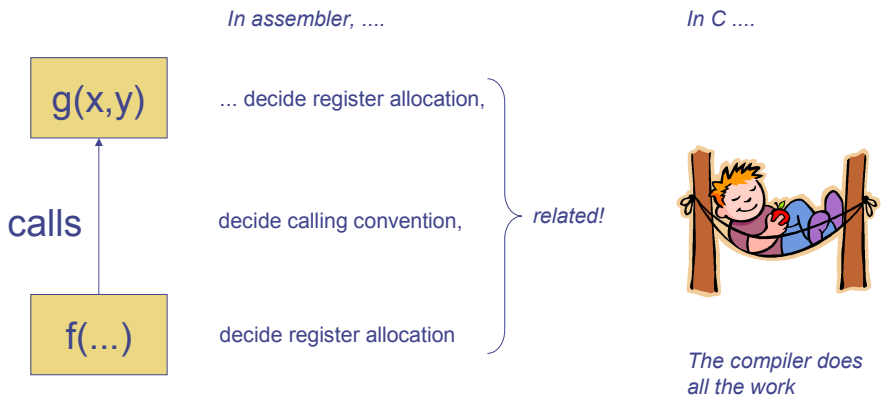


```
always @(posedge CLK) begin
  if ((cond2 && cond0) || (cond0 && !cond1 && !cond3))
    x <= x + 1;
  else if (cond3 && !cond2)
    x <= x + 2;
  else if (cond1 && !cond0)
    x <= x - 1

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

?

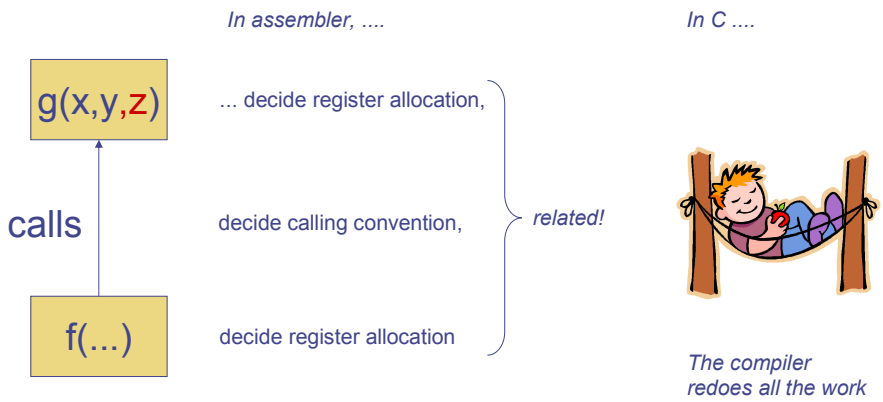
bluespec

A software analogy



bluespec

A software analogy

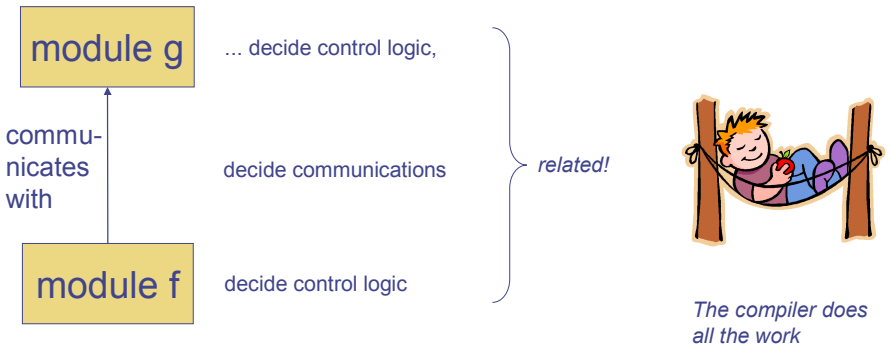


bluespec

In HW design

In Verilog, ...

In BSV ....



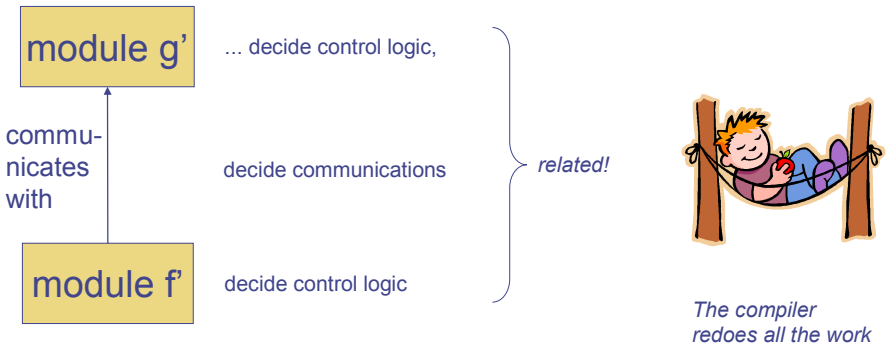
The compiler does all the work

bluespec

In HW design

In Verilog, ...

In BSV ....



The compiler redoes all the work

all over again (groan!)

bluespec

Summarizing, we have Point 2:

In RTL, control logic can be complex to write first time, and complex to change/maintain, because one needs to consider and reconsider non-local influences (even across module boundaries).

BSV, on the other hand, is "control adaptive" *because it automatically computes and recomputes the control logic as you write/change the design.*

bluespec

MEMOCODE 2008 codesign contest

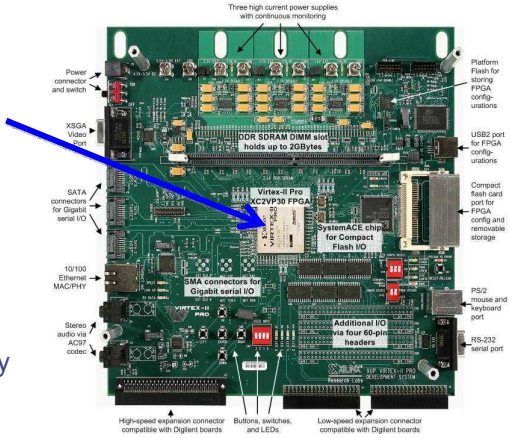
http://rijndael.ece.vt.edu/memocontest08/

Goal: Speed up a software reference application running on the PowerPC on Xilinx XUP reference board using SW/HW codesign

Application:
- decrypt,
- sort,
- re-encrypt

db of records in external memory

Time allotted: 4 weeks

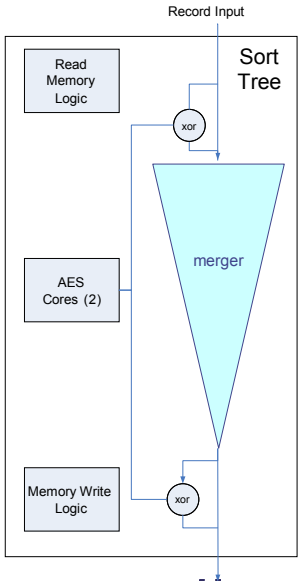


Xilinx XUP

bluespec

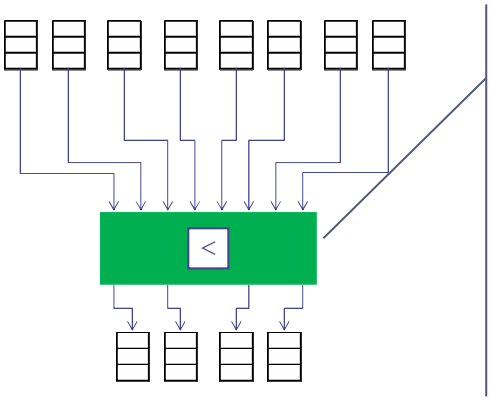
The winning architecture

- The "merger" has to be used repeatedly in order to mergesort the db
- A control issue: address generation and sequencing of memory reads and writes
- Observation: throughput in merger is limited by apex, so at each lower level, a single comparator can be shared
  ➔ can fit a deeper merger into available space



bluespec

Merger: each level



Loop:

1.Choose non-empty input pair corresponding to output fifo with room (scheduling)
2.Compare the fifo heads
3.Dequeue the smaller one and put it on output fifo

bluespec

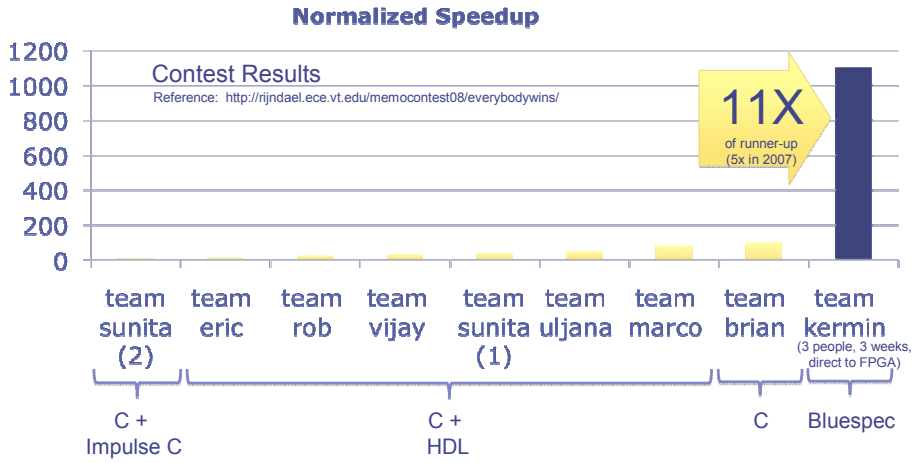
Scheduler as a Module Parameter

- Level 1 merger: must check 2 input FIFOs
  - Scheduler can be combinational
- Level 6 merger: must check 64 input FIFOs
  - Scheduler must be pipelined

➔ scheduler is a parameter of level merger

bluespec

MEMOCODE 2008 codesign contest



27 teams started the four week contest.
8 teams delivered 9 solutions.

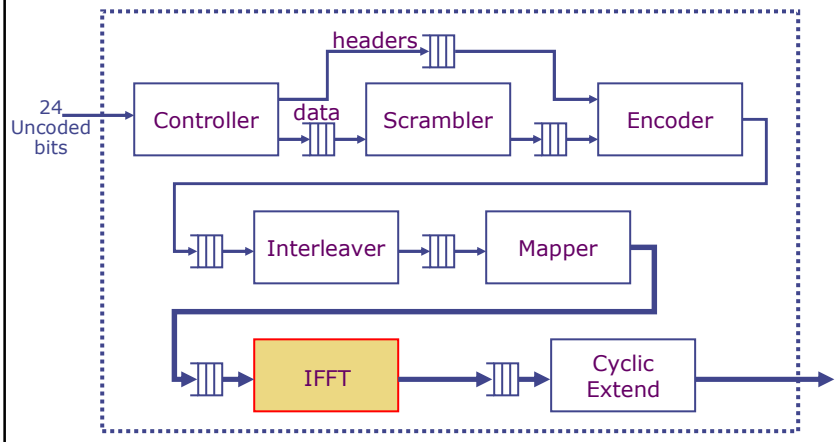
bluespec

Summarizing, we have Point 3:

Some control logic is too difficult even to contemplate without the support of atomic transactions

bluespec

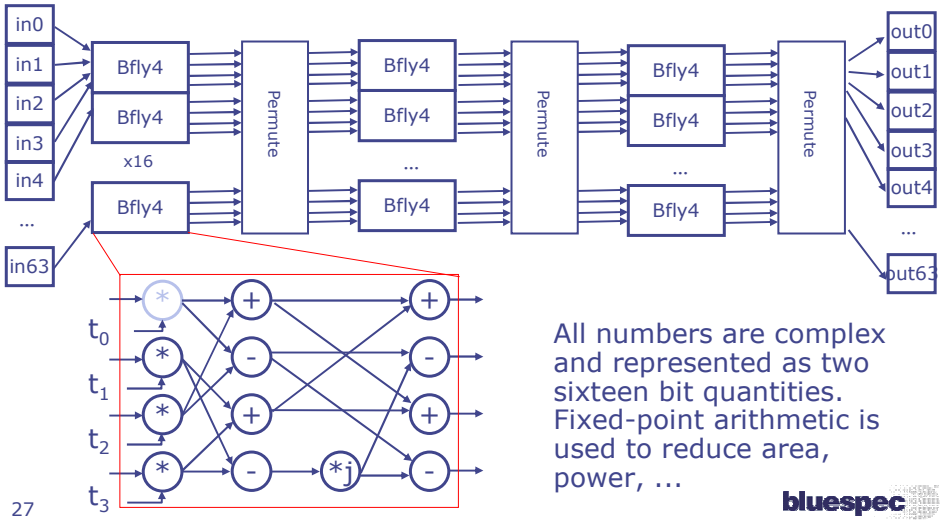
Another application of control adaptivity: parameterizing an 802.11a Transmitter



accounts for 85% area

bluespec

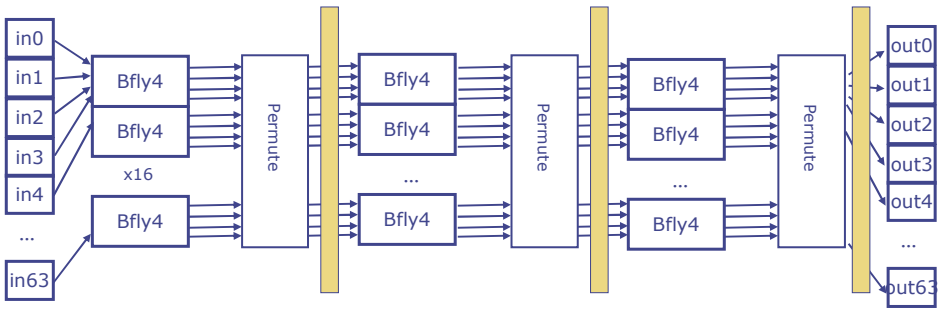
Combinational IFFT



All numbers are complex and represented as two sixteen bit quantities. Fixed-point arithmetic is used to reduce area, power, ...

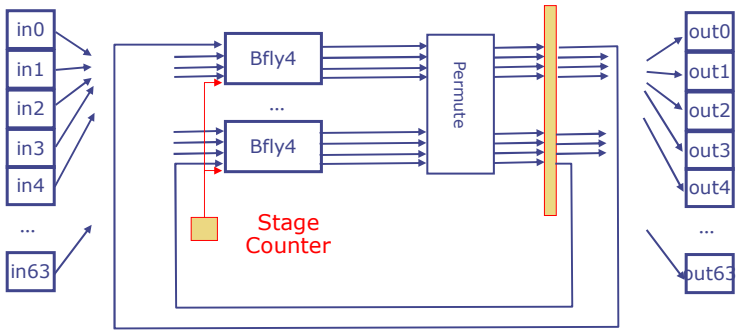
bluespec

Pipelined IFFT



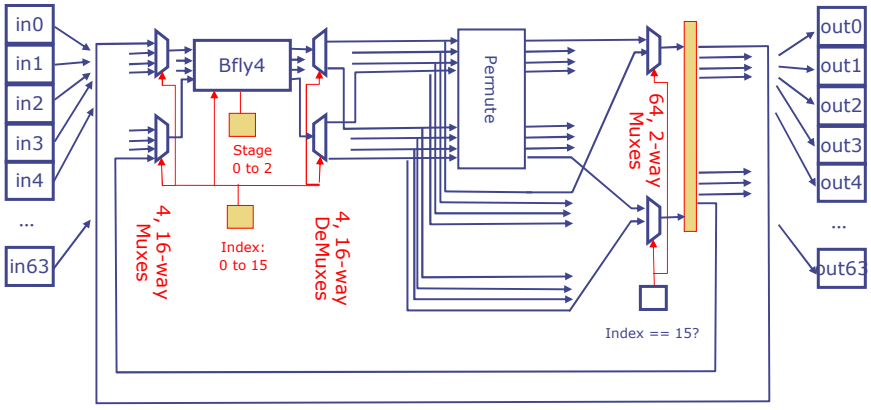
bluespec

Circular pipeline: Reusing the Pipeline Stage



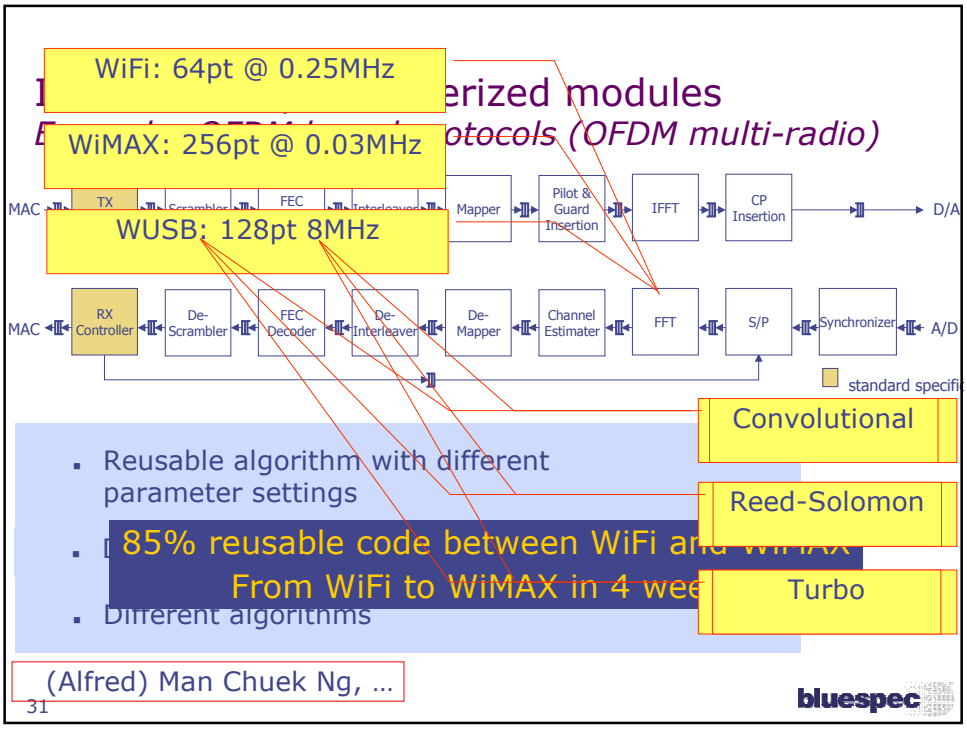
bluespec

Superfolded circular pipeline: Just one Bfly-4 node!



These different micro-architectures will have different (area, speed, power). Each may be the "best" for different target chips.

bluespec



Summarizing, we have Point 4:

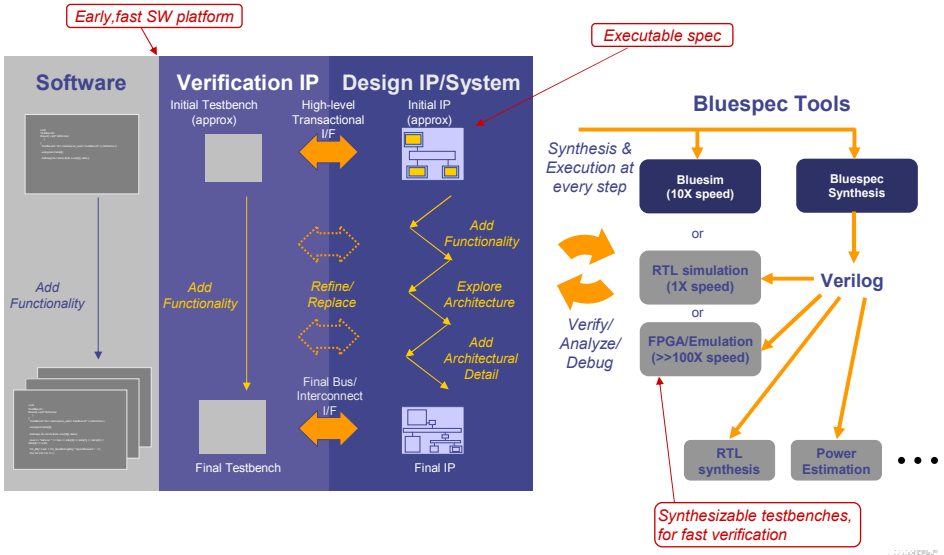
Parameterized architectures can deliver tremendous succinctness and reuse ...

... but each instance of a parameterized architecture has different resource contentions ➔ needs different control logic.

Consider the effort to redesign the control logic for each instance (RTL), vs. using regenerating it automatically due to control-adaptivity.

bluespec

Control-adaptivity has a profound effect on design methodology



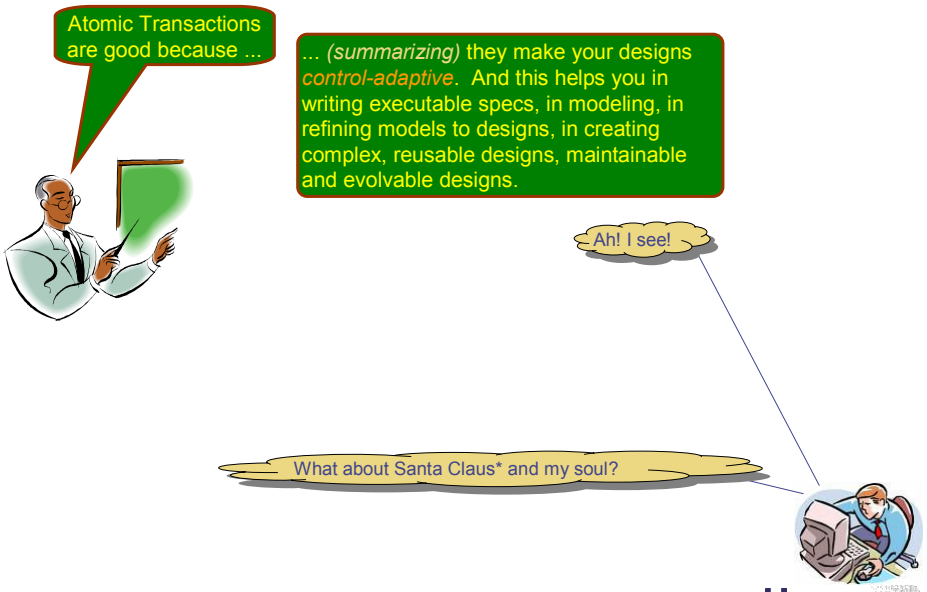
bluespec

Gradually expanding ways in which BSV is used

- Modeling for early SW development
  - Denali
    - Cycle-accurate LPDDR controller
- Modeling for architecture exploration
  - Intel, IBM, UT Austin, CMU
    - x86 and Power multi-thread/multi-core design
- Verification
  - Qualcomm
    - Testbenches, transactors
- and, of course, IP creation (original use)
  - TI, ST, Nokia, Mercury, MIT
    - DMAs, LCD controllers, Wireless modems, Video, ...

bluespec

In summary ...



* ask me if you want to see the Bluespec solution

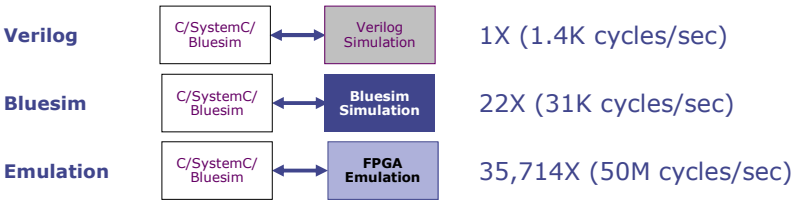
bluespec

Extra slides

bluespec

Example: ***IP verification*** (AXI demo on FPGA @ Bluespec)

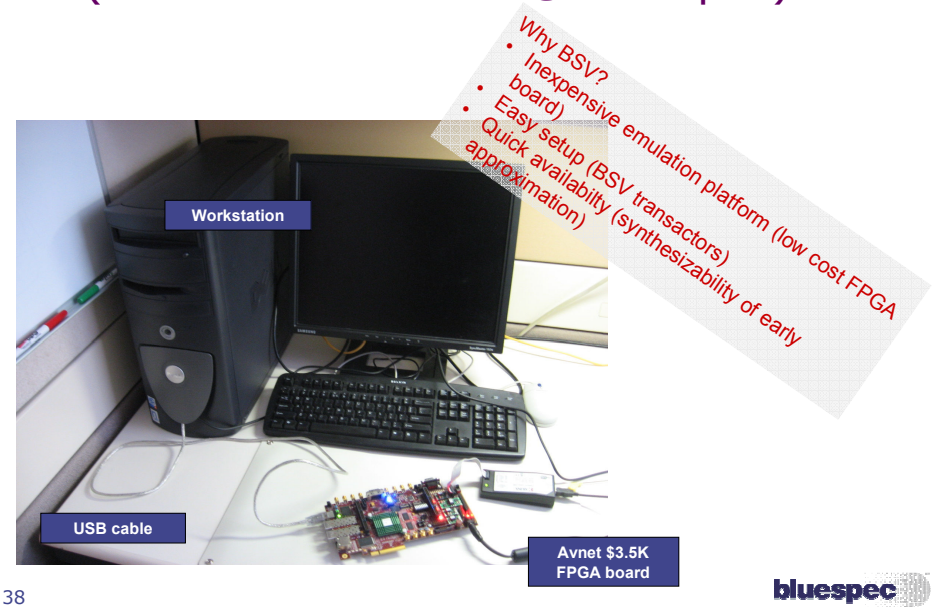
Bluespec lines of code:	2,000 (including comments)
ASIC gates:	125K
Virtex-4 FX100 slices:	4,723 (10% utilization)



22 day Verilog sim → 1 day Bluesim → 53 sec on FPGA

bluespec

Example: ***IP verification*** (AXI demo on FPGA @ Bluespec)



bluespec