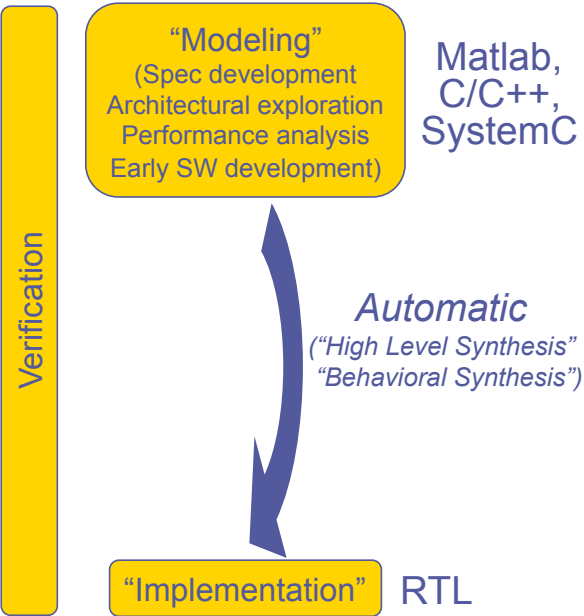


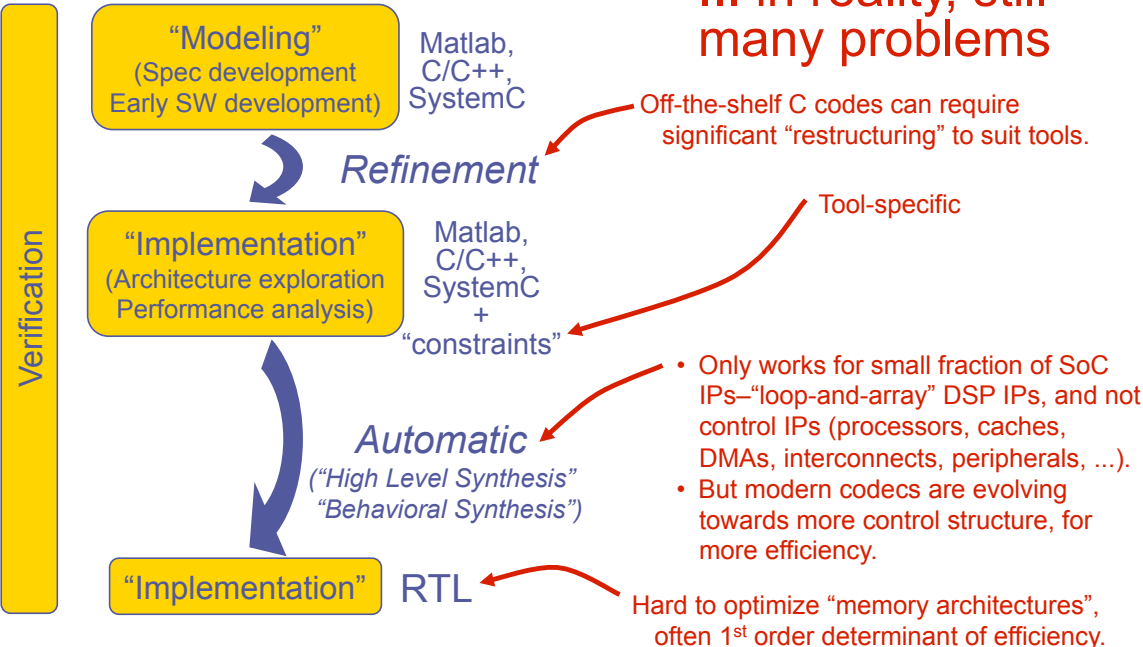
SoC methodology: a partial bright spot?

Nice vision, but ...

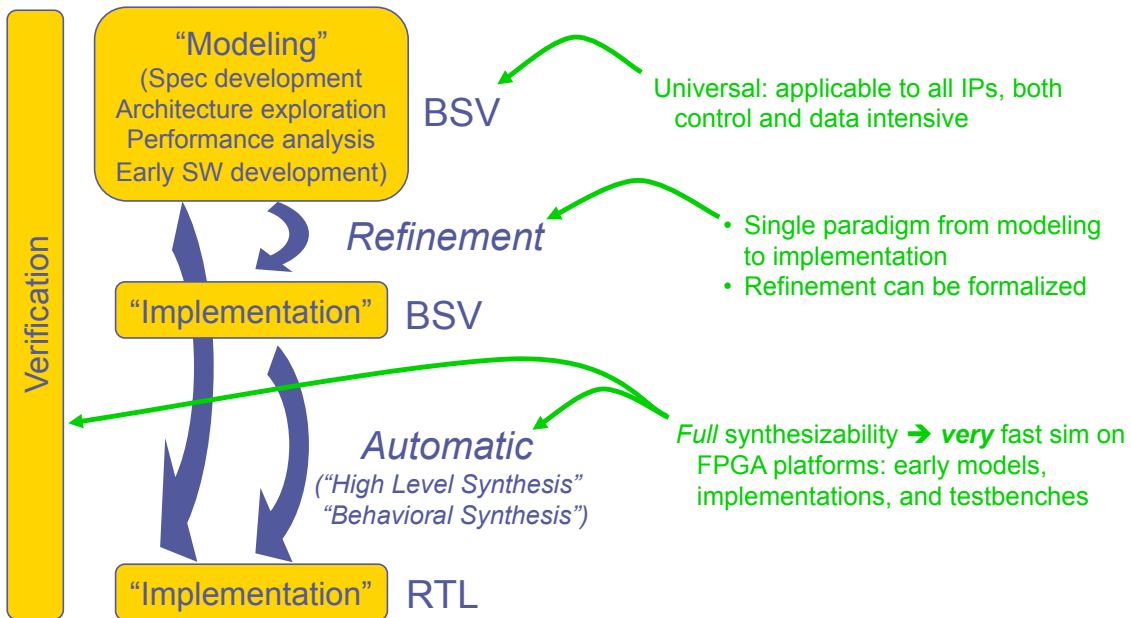


SoC methodology: a partial bright spot?

... in reality, still many problems



SoC methodology: a solution



5 BSV = Bluespec SystemVerilog, based on composable, parallel atomic transactions

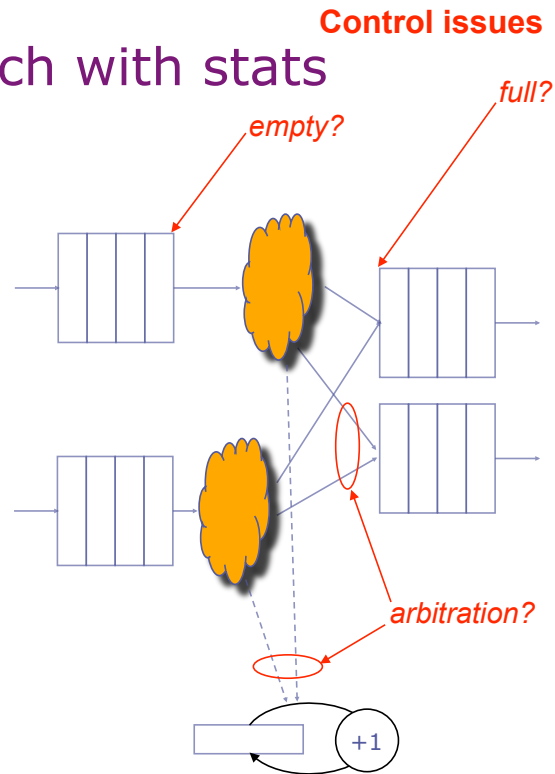


A flavor of the power of composable atomic transactions



Example: A 2x2 switch with stats

- ◆ Packets arrive on two input FIFOs, and must be switched to two output FIFOs
- ◆ Certain “interesting packets” must be counted
- ◆ Must have max throughput (no idle cycles)



7 **Dynamic, data-dependent access to shared resources!**

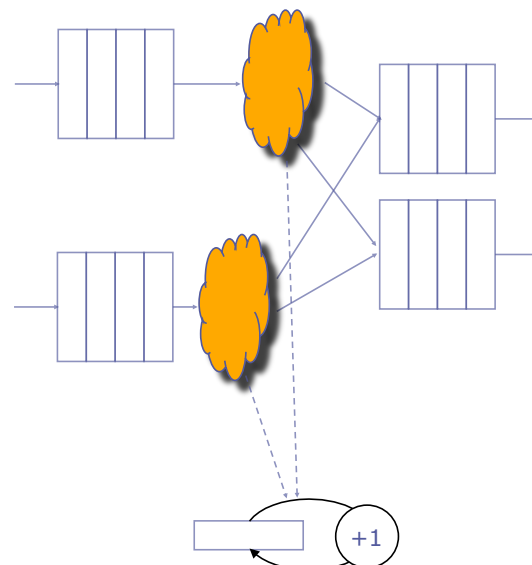


The meat of the BSV code

```

module mkSmallSwitch (IfcSmallSwitch);
...
// for-loop generates two rules
for (Integer j = 0; j < 2; j = j + 1)
  rule move_packet;
    let x = in[j].first(); in[j].deq();
    if (x[0] == 0)
      out[0].enq (x);
    else
      out[1].enq (x);
      if (interesting(x)) c <= c + 1;
  endrule
...
endmodule: mkSmallSwitch

```

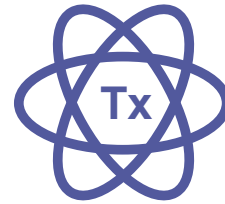


8 Rules are *atomic* → complex control is automatically synthesized



Atomic transactions are the best known tool to tame complex 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



11

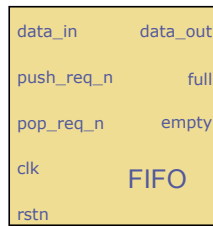
BSV's atomic transactions *compose* across module boundaries using *atomic interface methods*

12

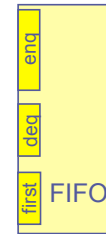


Atomic Interface Methods

RTL



BSV



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.

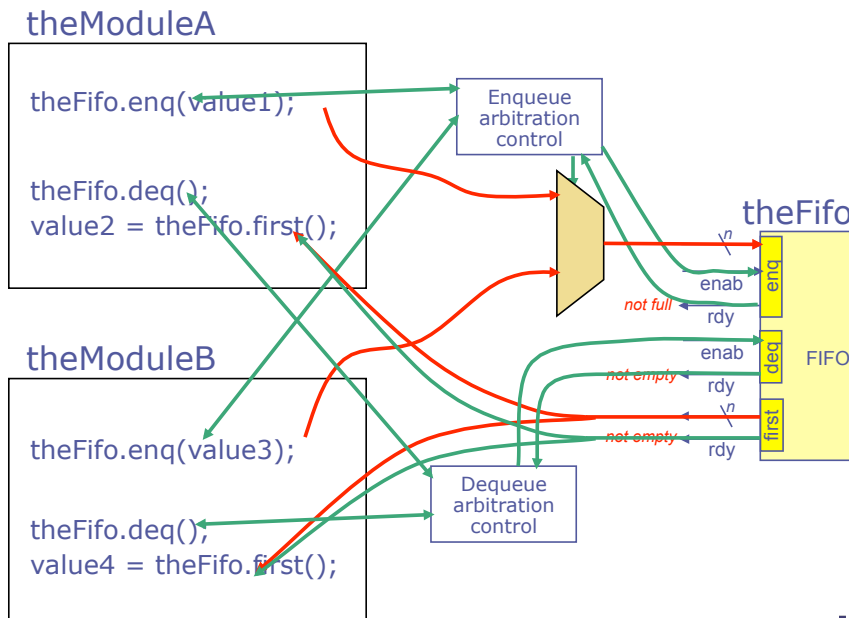
A pop operation occurs when pop_req_n is asserted (LOW), as long as the FIFO is not empty. Asserting pop_req_n causes the internal read pointer to be incremented on the next rising edge of clk. Thus, the RAM read data must be captured on the clk following the assertion of pop_req_n.

All BSV interfaces are *transactional*

```
interface FIFO #(t);
  method Action   enq (t x);
  method x_type   first ();
  method Action   deq();
endinterface
```



Atomicity of interface methods encapsulates the complex control logic necessary for correct module composition (and, by implication, IP reuse)



Bluespec generates correct control logic to interface properly at every module instantiation

RTL

Push when FULL?
Pop when EMPTY?
Simultaneous PUSH/POP?
Arbitrations?
Properly connected?
Grabbing data correctly?
Putting data in correctly?

data_in	data_out
push_req_n	full
pop_req_n	empty
clk	FIFO
rstn	

Logic around every instantiation is at risk & every corner case for every aspect of every instantiation's interface must be exercised!

BSV

enq
deq
FIFO
first

Logic around every instantiation is correct by construction (control logic arising out of atomicity semantics)

bluespec

15

Let's swap in a different FIFO with the same interface ports, BUT ...

- The new FIFO allows simultaneous enq/deq when EMPTY instead of when FULL (→ change to external control logic)

RTL

The control logic around every instantiation must change & be retested!

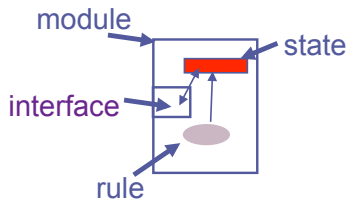
BSV

The surrounding control logic is automatically resynthesized from atomic semantics.

bluespec

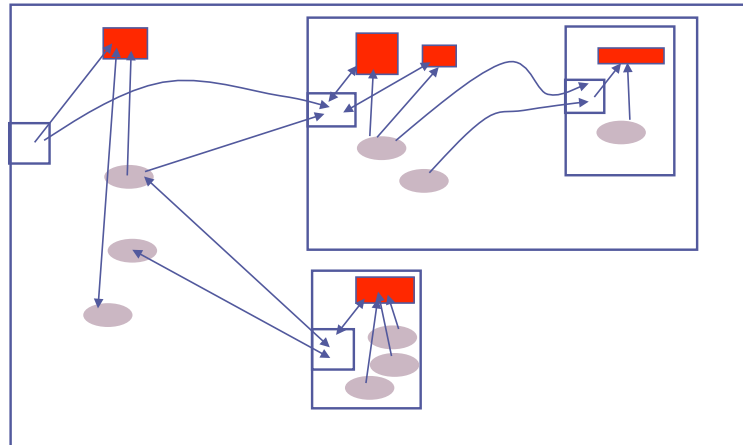
16

Disciplined composition of modules into subsystems and systems



Modules contain rules, which use methods provided by submodules in their interfaces. Methods, too, can use other methods.

Rules, which compose across the system, are guaranteed atomic.



17

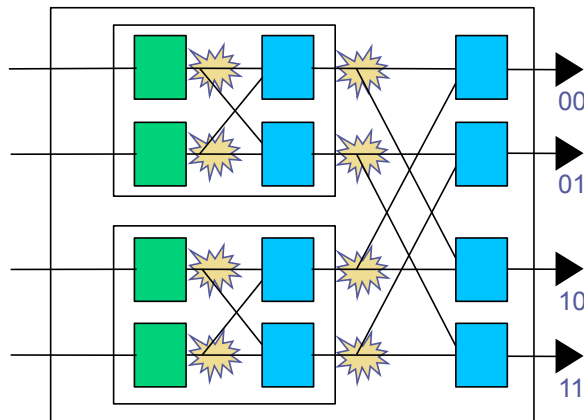
bluespec

BSV's extreme parameterization enables a single source for a family of microarchitectures

18

bluespec

Example: a butterfly switch (crossbar)




Basic building blocks:   

Recursive construction: 1x1 → 2x2 → 4x4 ... → NxN

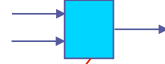
19



Butterfly switch: module (< 60 lines, fully synthesizable)

Packet routing function 

Parameterization with ANY type

2→1 merge submodule 

```

module mkXBar #(function UInt #(32) destinationOf (t x),
               module #(Merge2x1 #(t)) mkMerge2x1)
  ( XBar #(n, t) );
  if (logn == 0) ... // BASE CASE
    FIFO#(t) f <- mkFIFO;
    ...
  else ... // RECURSIVE CASE
    ...
    XBar#(nhalf, t) upper <- mkXBar ( ... );
    XBar#(nhalf, t) lower <- mkXBar ( ... );
    ...
    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
  
```

High level interfaces

Turing-complete "generate" over arbitrary design elements

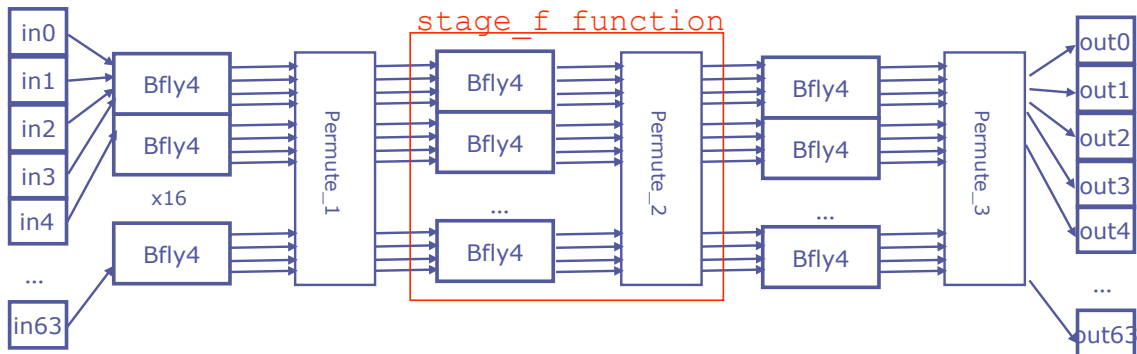
Recursive module instantiation

Generate behavior (rules) with loops

20

Example: IFFT (in 802.11a and other apps)

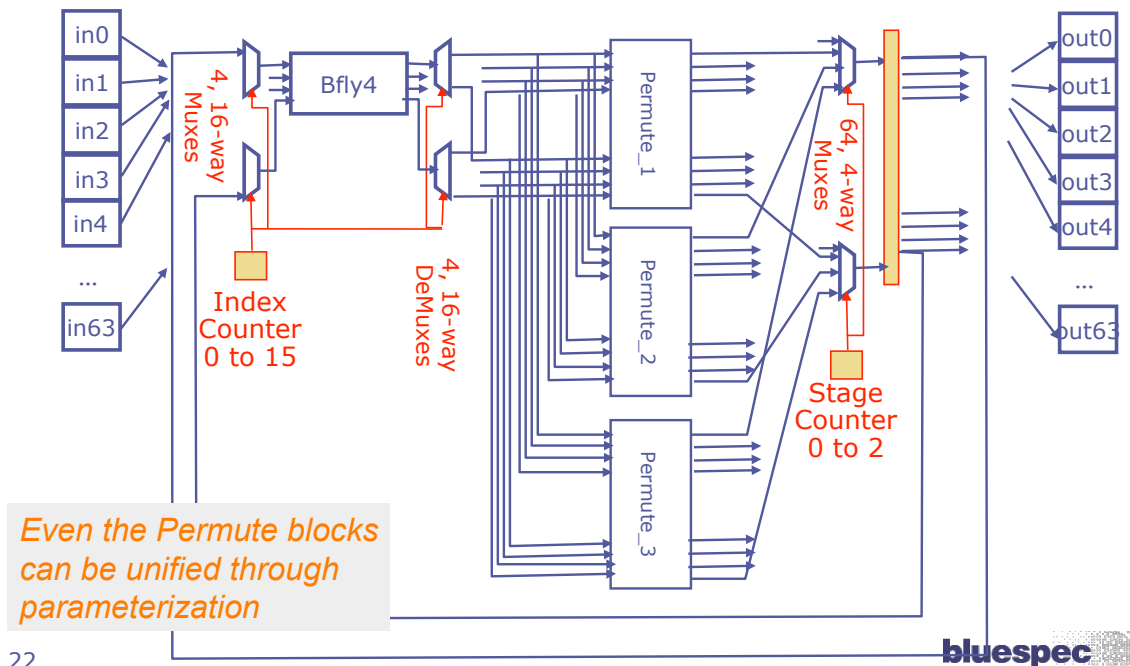
Microarchitectures: from single combinational function ...



21



... to a superfolded circular pipeline: Just one Bfly-4 node!



22



Because of such parameterization, encapsulation, and reuse,

- ◆ All these designs were done in less than *one* day, with a single parameterized source!
- ◆ Very quick exploration of area and power tradeoffs
- ◆ Transparent, predictable, controllable microarchitecture, despite high-level spec

Combinational
Pipelined
Folded (16 Bfly 4s)
Super Folded (8 Bfly 4s)
Super Folded (4 Bfly 4s)
Super Folded (2 Bfly 4s)
Super Folded (1 Bfly 4)

23

Nirav Dave, Mike Pellauer, Steve Gerding, Arvind
MEMOCODE 2006



The underlying rule-based atomicity semantics are crucial!

- ◆ Each microarchitecture variation changes the resource sharing
 - Synthesis based on atomicity allows control logic to *track* these variations automatically
 - I.e., **Control Adaptive** parameterization
- ◆ Latency insensitive methodology (elastic pipes, GALS) allows robust plug and play of microarchitecture choices

24



Example: H.264 decoder

- ◆ Complete decoder, not just kernel blocks
- ◆ Range of implementations from a common source
 - from QCIF: 176x144 @ 15 frames/sec
 - to 1080p: (1280x1080)p @ 60 frames/sec
- ◆ Synthesized at 180 nm
- ◆ < 10K lines of BSV source code
 - Original reference code: > 80K lines of C
 - H.264 slice of FFMPEG: 20K lines of C (unsynthesizeable)

This BSV code is open sourced: <http://csg.csail.mit.edu/oshd>

25



BSV:
well placed for formal verification

26



Rules: formality and refinement

- ◆ Many formal specification languages use the same computation model, because it is **parallel**, and because atomicity enables reasoning about correctness with **invariants**
 - UNITY (Chandy&Mishra), TLA+ (Lamport), Event B (Abrial), ...
- ◆ The Rules computation model can be used from high levels of abstraction (executable specs) to lower levels (implementations)
 - Vast literature on *provably correct refinement*

27



Implications of BSV's full synthesizability

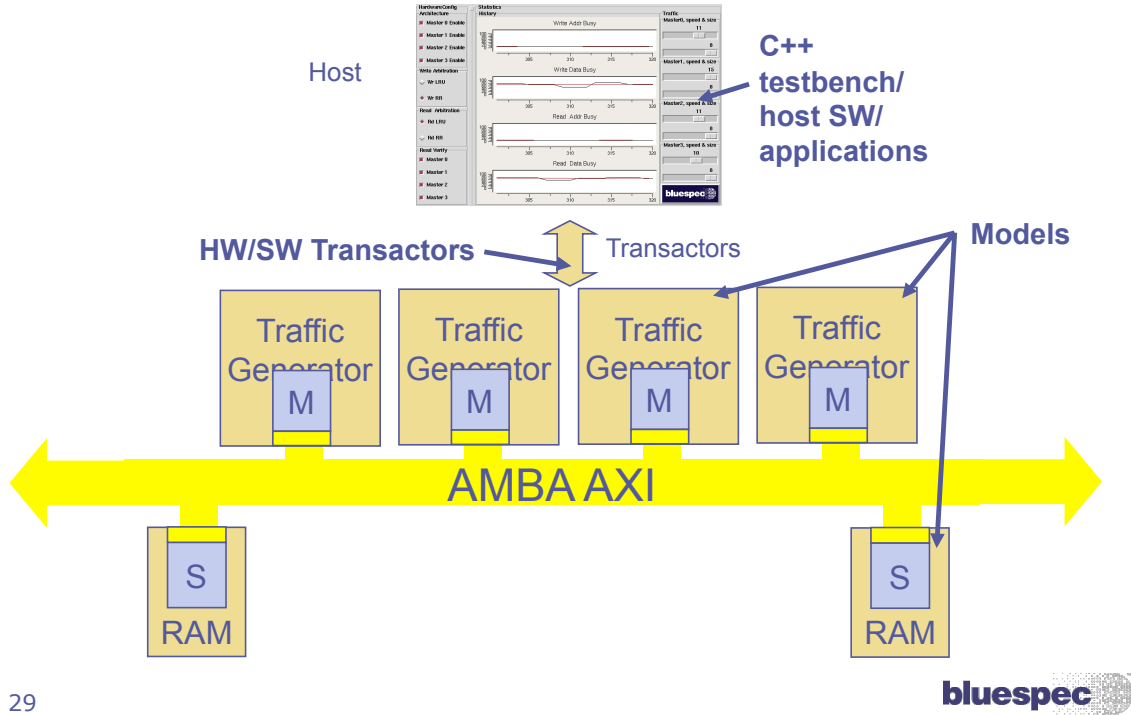
- ◆ Even high-level models can be executed on FPGAs
 - Early exploration
 - Early SW development
- ◆ Verification testbenches can be executed on FPGAs
 - Much faster than Verilog sim for verification

Bluespec provides 'push button' infrastructure to map components with TLM interfaces to FPGAs

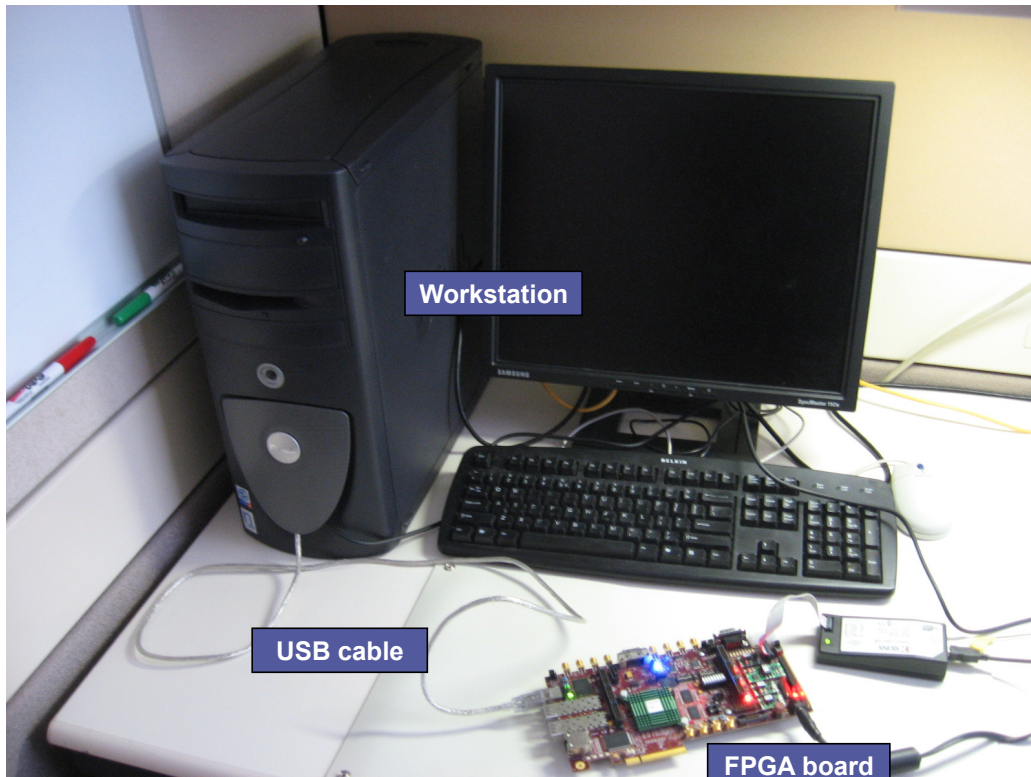
28



Example: AXI Virtual Platform Demo

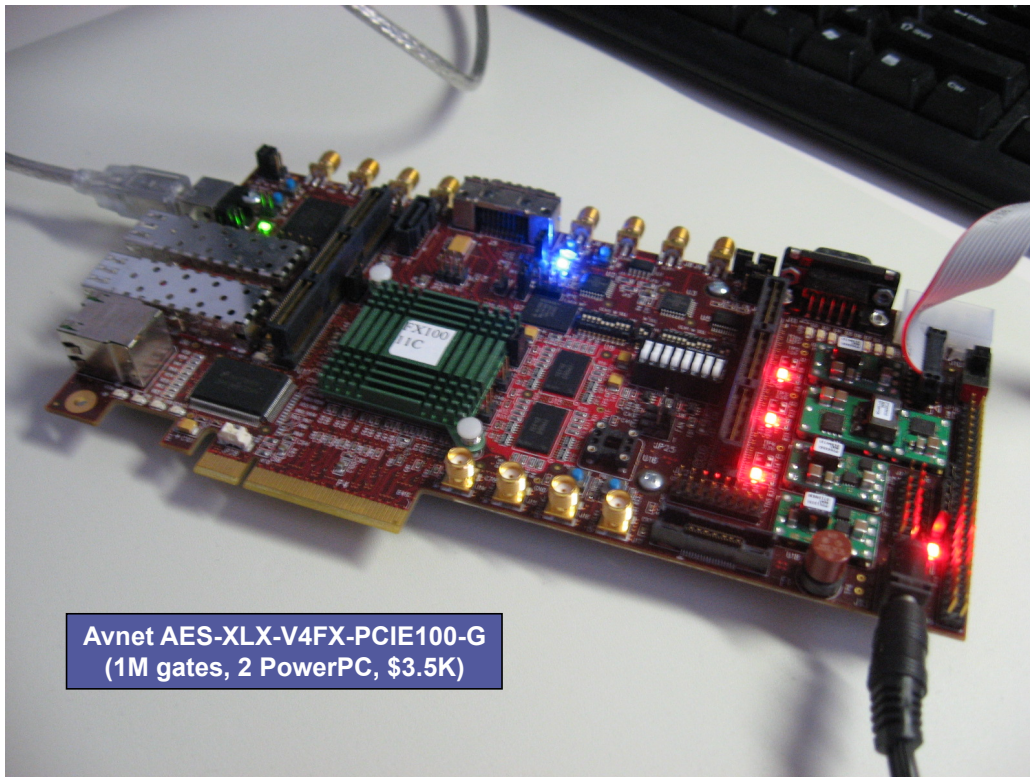


29



30

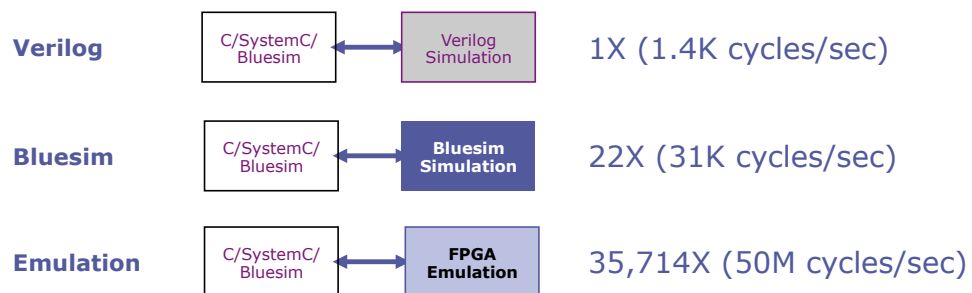




Avnet AES-XLX-V4FX-PCIE100-G
(1M gates, 2 PowerPC, \$3.5K)

AXI Demo Execution Speeds

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 Emulation

Some customer use cases:

- ◆ IP creation
- ◆ Modeling
- ◆ Architecture Exploration
- ◆ Verification

33



ASIC IP creation at three major semiconductor companies

- ◆ High performance video data mover for video subsystem¹

- ◆ System DMA for wireless handset platform¹
- ◆ Image DMA²
- ◆ LCD controller²

- ◆ Turbo Viterbi²

(1) *seen silicon; derivatives in progress*
(2) *in progress*

Why BSV?
• Complex concurrency
• Parameterization

34



Modeling at a major IP company

- ◆ Cycle-accurate model of a production LPDDR memory controller
 - Will be shipped to each customer of the IP company

Why BSV?

- Very quick model creation (< 2 man months)
- Parameterization
- Fast sim (Bluesim)
- Potential for *much* faster sim (FPGA)
- Potential to replace IP creation as well

35



Architecture exploration of processor microarchitectures on FPGAs (@ a major microprocessor company)

- ◆ Background: existing microarchitect's workbench for exploring alternative microarchitecture for future processors
 - Written in C++, developed over a decade
 - Highly parameterized and configurable, to facilitate experimentation on alternatives
 - Heavily used, but running out of gas for simulation speed (multicore, multithreading)
- ◆ New: rewrite in BSV to synthesize and run on FPGAs,
 - expected performance advantage > **1000x**
- ◆ Status: Demonstrated for 5-stage pipeline model and pipelined out-of-order model; development continues

Why BSV?

- Only tool with expressive power to replace C++
- and be synthesizable to FPGA, for speed

36



Architecture exploration of processor microarchitecture on FPGAs (@ a major microprocessor/systems company)

- ◆ Goal: flexible platform for fast exploration of microarchitectures for multithreaded + multicore CPU systems
- ◆ Is being implemented in BSV in order to synthesize and run on FPGAs
 - Status: executing significant prefix of Linux boot sequence, on an FPGA platform, within 6 months of start of project

Why BSV?

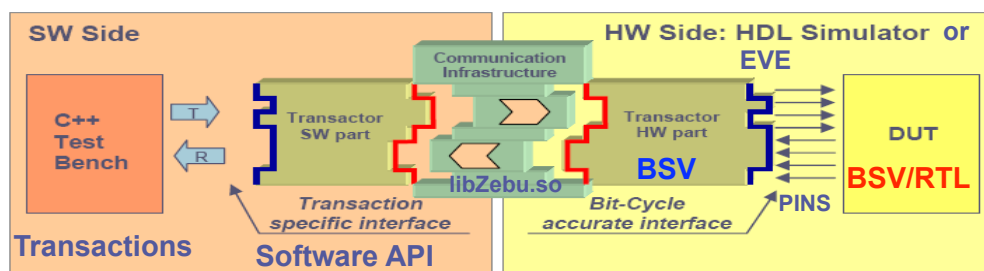
- Very quick creation of high level, architecturally accurate model
- and be synthesizable to FPGA, for speed

37

bluespec

Verification @ Qualcomm

- ◆ BSV for complex transactors on EVE platform
 - TLM interfaces
 - Functions: data transformation, clock management, timestamp management, statistics management
 - and, ... moving testbench functionality to EVE side



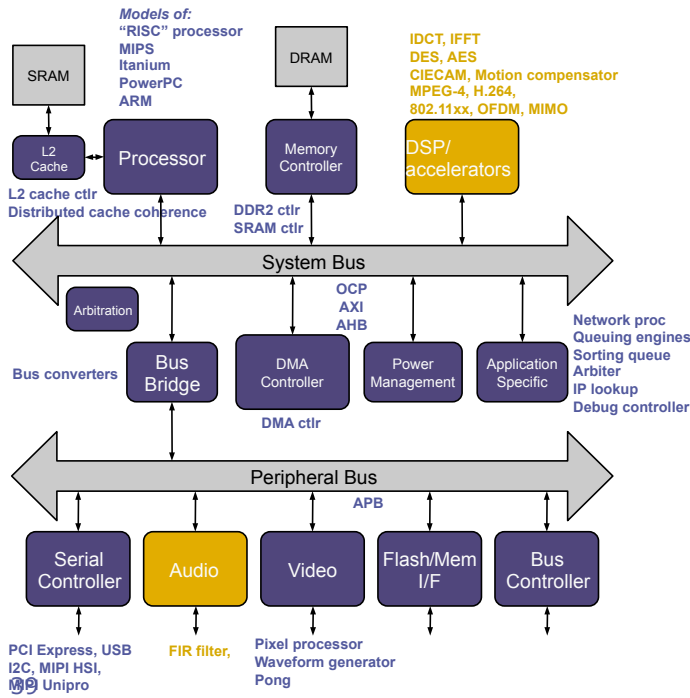
- Why BSV?
 - Quick creation of complex test infrastructure
 - and be synthesizable to FPGA, for speed

38

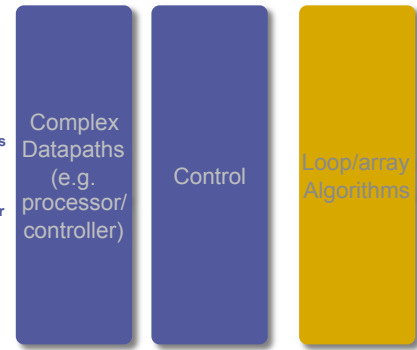
bluespec

It's general purpose and practical

Typical IPs in an SoC ; IPs done in BSV (with good QoR)



Bluespec SystemVerilog (BSV)



C-based synthesis



Summary: BSV → a high level, disciplined approach to SoC/IP design

