Bluespec-1: Design Affects Everything

Arvind

Computer Science & Artificial Intelligence Lab

Massachusetts Institute of Technology

Based on material prepared by Bluespec Inc, January 2005

Chip costs are exploding because of design complexity

SoC failures costing time/spins

Issues Found on First Spin ICs/ASICs

Functional Logic Error	43%
Analog Tuning Issue	20%
Signal Integrity Issue	17%
Clock Scheme Error	14%
Reliability Issue	12%
Mixed Signal Problem	11%
Too Much Power	11%
Has Path(s) Too Slow	10%
Has Path(s) Too Fast	10%
IR Drop Issues	7%
Firmware Error	4%
Other	3%
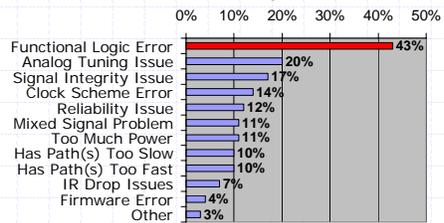
Source: Aart de Geus, CEO of Synopsys
Based on a survey of 2000 users by Synopsys
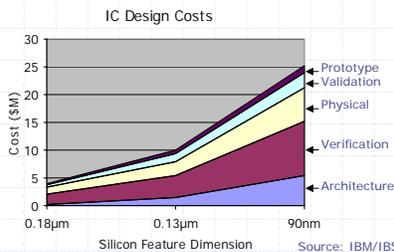
IC Design Costs

Design and verification dominate escalating project costs

Source: IBM/IBS, Inc.

1

Common quotes

- ◆ "Design is not a problem; design is easy"

- ◆ "Verification is a problem"
- ◆ "Timing closure is a problem"
- ◆ "Physical design is a problem"

Mind set

Almost complete reliance on post-design verification for quality

---

Through the early 1980s:

The U.S. auto industry
- ◆ Sought quality solely through post-build inspection
- ◆ Planned for defects and rework

Make   Inspect   Rework
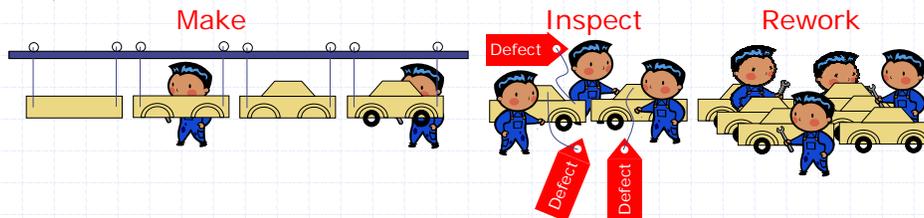
and U.S. quality was…

... less than world class

Image of Chevrolet Chevette removed due to copyright restrictions.

- ◆ Adding quality inspectors ("verification engineers") and giving them better tools, was not the solution
- ◆ The Japanese auto industry showed the way
  - ▪ *"Zero defect" manufacturing*

New mind set:
Design affects everything!

- ◆ A good design methodology
  - ▪ Can keep up with changing specs
  - ▪ Permits architectural exploration
  - ▪ Facilitates verification and debugging
  - ▪ Eases changes for timing closure
  - ▪ Eases changes for physical design
  - ▪ Promotes reuse

⇒ It is essential to

Design for Correctness

Why is traditional RTL too low-level?

Examples with dynamic and static constraints

Design must follow many rules ("micro-protocols")

Consider a FIFO (a queue)

first: *examine item at head of queue*

enq: *put an item into the queue*   deq: *remove an item from the queue*
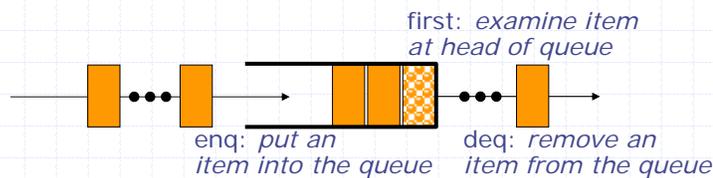
In the hardware,
there are a number of requirements
for correct use

DATA_IN $\sqrt{n}$
ENAB
not full RDY   enq

ENAB
not empty RDY   deq   FIFO

DATA_OUT n
not empty RDY   first

4
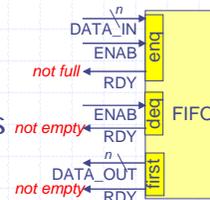
Requirements for correct use

Requirement 1:   deq ENAB only when RDY (not empty)

Requirement 2:   first DATA_OUT only when RDY (not empty)

Requirement 3:   enq ENAB simultaneously with DATA_IN

Requirement 4:   enq ENAB only when RDY (not full)
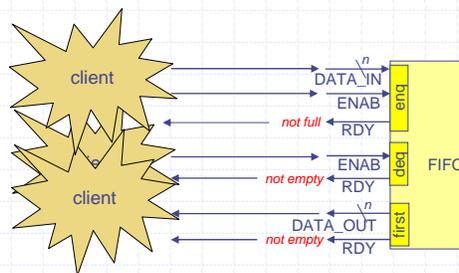
Correct use of a shared FIFO

- Needs a multiplexer in front of each input ( )
- Needs proper control logic for the multiplexer

5
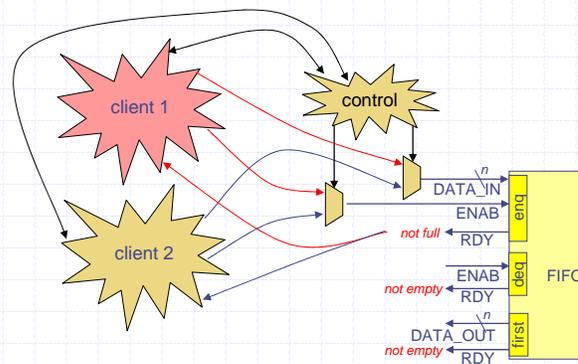
Concurrent uses of a FIFO

enq ENAB ok if deq ENAB, even if not RDY ??

client 1

DATA_IN
ENAB
not full RDY

enq

ENAB
not empty RDY

deq FIFO

DATA_OUT
not empty RDY

first

client 2
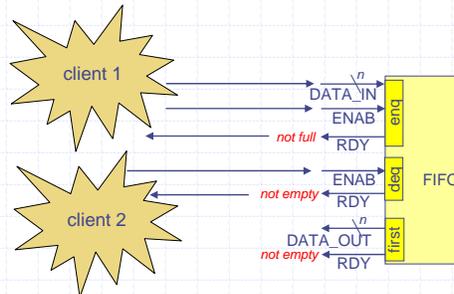
Example from a commercially available FIFO IP component

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.

data_in   data_out
push_req_n   full
pop_req_n   empty
clk
rstn

These constraints are taken from several paragraphs of documentation, spread over many pages, interspersed with other text

6

A High-Bandwidth Credit-based Communication Interface

I/F Control $Credit = C_1$	You can have X credits	I/F Control $Credit = C_2$
Module A	I can send up to X items	Module B
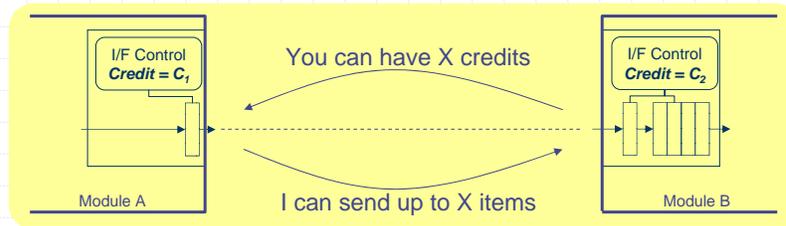
- Static correctness constraints:
  - Data types agree on both ends?
  - Credit values agree (C1 == C2)?
  - Credit values automatically sized to comm latency?
  - B's buffer properly sized (C2)?
  - B's buffer pointers properly sized (log(C2))?

Why is Traditional RTL low-level?

- Hardware for dynamic constraints must be *designed explicitly*
- Design assumptions must be *explicitly verified*
- Design assumptions must be *explicitly maintained* for future changes
- If static constraints are *not checked* by the compiler then they must also be *explicitly verified*

In Bluespec SystemVerilog (BSV) …

◆ Power to express complex static structures and constraints
  ▪ Checked by the compiler
◆ "Micro-protocols" are managed by the compiler
  ▪ The compiler generates the necessary hardware (muxing and control)
  ▪ Micro-protocols need less or no verification
◆ Easier to make changes while preserving correctness

➔ *Smaller, simpler, clearer, more correct code*

Bluespec SystemVerilog (BSV)

Bluespec SystemVerilog

High-level description of FSMs
  Rules, Interface Methods
Static elaboration, verification
  Types, Procedures

SystemVerilog

Structure
  Modules, interfaces, types
HW semantics
  Cooperating FSMs
+ Assertions

Structure
  Modules, interfaces, types
HW semantics
  Cooperating FSMs
+ Assertions

Low-level description of FSMs
  Processes, cycle counting, explicit management of shared resources

Low-level description of FSMs
  Processes, cycle counting, explicit management of shared resources

8

Bluespec Tool flow

Bluespec SystemVerilog source

Bluespec Compiler

Blueview

C

Verilog 95 RTL

Bluespec C sim

Cycle Accurate

Verilog sim

RTL synthesis

VCD output

gates

Legend
- files
- Bluespec tools
- 3rd party tools

Debussy Visualization

Place & Route

Physical

Tapeout

http://csg.csail.mit.edu/6.884/   L07-17

Bluespec: State and Rules organized into *modules*

module

interface

All *state* (e.g., Registers, FIFOs, RAMs, …) is explicit.
Behavior is expressed in terms of atomic actions on the state:

     Rule: condition ➔ action

Rules can manipulate state in other modules only *via* their interfaces.

February 22, 2005   http://csg.csail.mit.edu/6.884/   L07-18

9
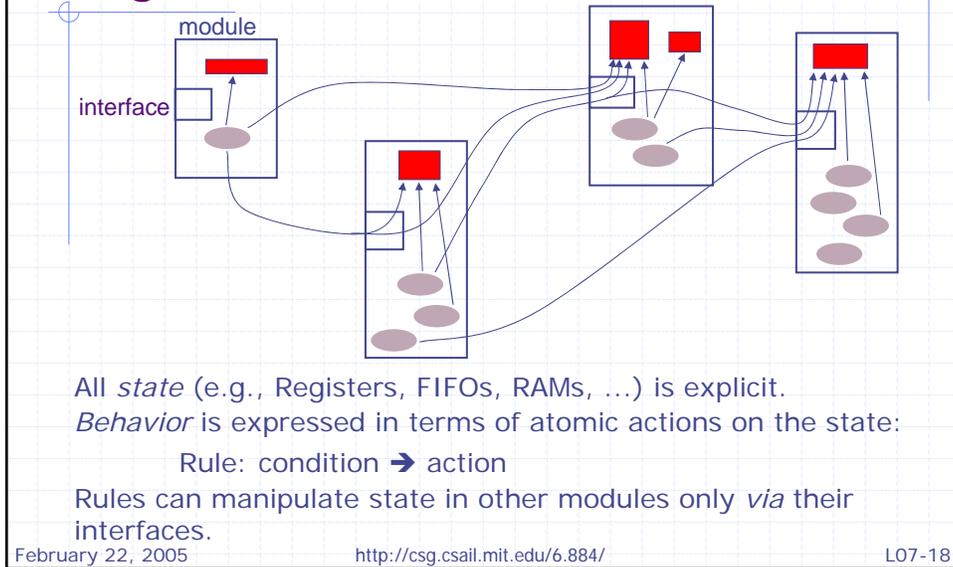
Programming with rules: A simple example

Euclid's algorithm for computing the Greatest Common Divisor (GCD):

15	6	
9	6	*subtract*
3	6	*subtract*
6	3	*swap*
3	3	*subtract*
0	*answer:* 3	*subtract*

GCD in BSV

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

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

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

GCD Hardware Module
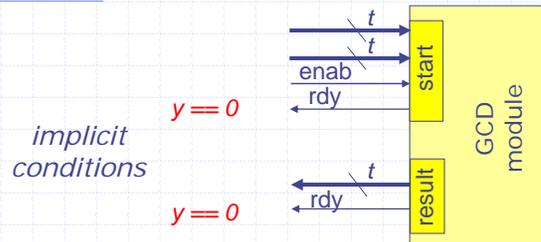

```
interface ArithIO #(type t);
    method Action start (t a, t b);
    method t result();
endinterface
```

Many different implementations can provide the same interface:

```
module mkGCD (ArithIO#(int));
```

Generated Verilog RTL: GCD

```
module mkGCD(CLK, RST_N,start__1, start__2, E_start_, ...)
  input CLK; ...
  output start__rdy; ...
  wire [31 : 0] x$get; ...
  assign result_ = x$get;
  assign _d5 = y$get == 32'd0;
  ...
  assign _d3 = x$get ^ 32'h80000000) <= (y$get ^ 32'h80000000);
  assign C___2 = _d3 && !_d5;
  ...
  assign x$set = E_start_ || P___1;
  assign x$set_1 = P___1 ? y$get : start__1;
  assign P___2 = _d3 && !_d5;
  ...
  assign y$set_1 =
      {32{P___2}} & y$get - x$get | {32{_dt1}} & x$get |
      {32{_dt2}} & start__2;
  RegUN #(32) i_x(.CLK(CLK), .RST_N(RST_N), .val(x$set_1), ...)
  RegN  #(32) i_y(.CLK(CLK), .RST_N(RST_N), .init(32'd0),  ...)
endmodule
```

Exploring microarchitectures

IP Lookup Module

IP Lookup block in a router

Line Card (LC)

LC

Arbitration

Packet Processor

SRAM
(lookup table)

Control Processor

IP Lookup

Queue Manager

Switch

Exit functions
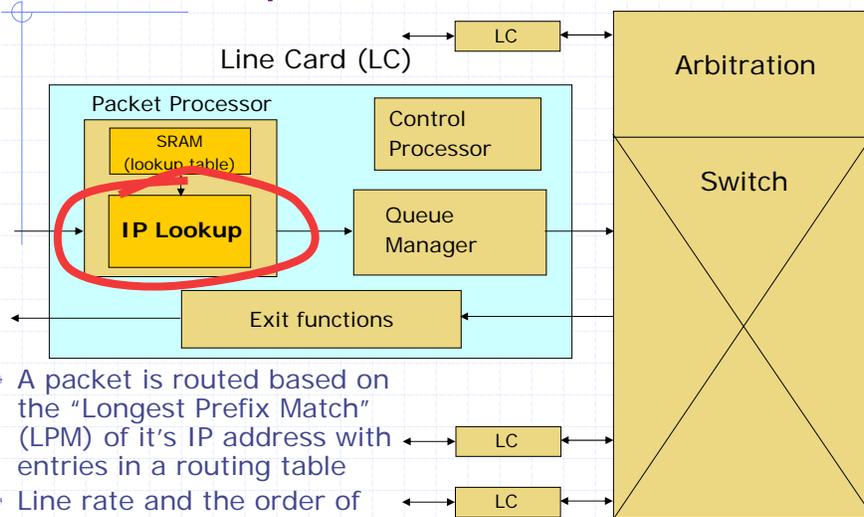
- A packet is routed based on the "Longest Prefix Match" (LPM) of it's IP address with entries in a routing table

LC

- Line rate and the order of arrival must be maintained

LC

line rate ⇒ 15Mpps for 10GE

Sparse tree representation

7.14.*.*	A
7.14.7.3	B
10.18.200.*	C
10.18.200.5	D
5.*.*.*	E
*	F

IP address	Result	M Ref
7.13.7.3	F	2
10.18.201.5	F	3
7.14.7.2	A	4
5.13.7.2	E	1
10.18.200.7	C	4

Real-world lookup algorithms are more complex but all make a sequence of dependent memory references.

SW ("C") version of LPM

```
int
lpm (IPA ipa)                  /*  3 memory lookups */
{
    int p;

    p = RAM [ipa[31:16]];      /*  Level 1: 16 bits  */
    if (isLeaf(p)) return p;

    p = RAM [p + ipa [15:8]];  /*  Level 2: 8 bits  */
    if (isLeaf(p)) return p;

    p = RAM [p + ipa [7:0]];   /*  Level 3:  8 bits  */
    return p;                  /* must be a leaf */
}
```

How to implement LPM in HW?
Not obvious from C code!

13

Slide 1

Longest Prefix Match for IP lookup:
3 possible implementation architectures

Rigid pipeline	Linear pipeline	Circular pipeline

Inefficient memory usage but simple design
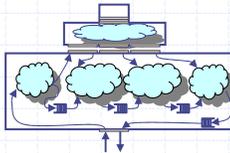
Efficient memory usage through memory port replicator
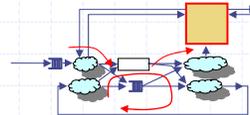
Efficient memory with most complex control

Designer's Ranking: ① ② ③

Which is "best"?

Arvind, Nikhil, Rosenband & Dave ICCAD 2004

Slide 2

Synthesis results

LPM versions	Code size (lines)	Best Area (gates)	Best Speed (ns)	Mem. util. (random workload)
Static V	220	2271	3.56	63.5%
Static BSV	179	2391 (5% larger)	3.32 (7% faster)	63.5%
Linear V	410	14759	4.7	99.9%
Linear BSV	168	15910 (8% larger)	4.7 (same)	99.9%
Circular V	364	8103	3.62	99.9%
Circular BSV	257	8170 (1% larger)	3.67 (2% slower)	99.9%

Synthesis: TSMC 0.18 μm lib

- Bluespec results can match carefully coded Verilog
- Micro-architecture has a dramatic impact on performance
- Architecture differences are much more important than language differences in determining QoR

V = Verilog; BSV = Bluespec System Verilog

14

Implementations of the same arch - Static pipeline: Two designers, two results

LPM versions	Best Area (gates)	Best Speed (ns)
Static V (Replicated)	8898	3.60
Static V (BEST)	2271	3.56

Replicated:

IP addr → result

MUX / De-MUX

FSM FSM FSM FSM

Each packet is processed by one FSM

Counter ↔ MUX / De-MUX

RAM

BEST:

IP addr

MUX

result

FSM

Shared FSM

RAM
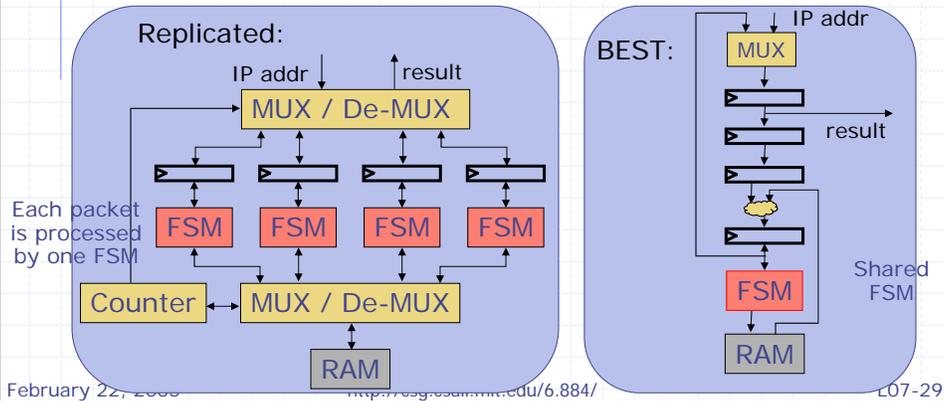
Reorder Buffer

Verification-centric design

15

Example from CPU design

- Speculative, out-of-order
- Many, many concurrent activities

Register File

Fetch — FIFO — Decode — FIFO — Re-Order Buffer (ROB) — FIFO FIFO — ALU Unit

Branch

FIFO FIFO — MEM Unit

Instruction Memory — FIFO | FIFO

Data Memory — FIFO | FIFO

ROB actions

Register File

Empty E
Waiting W
Dispatched Di
Killed K
Done

Get operands for instr

Writeback results

Re-Order Buffer

State	Instruction	Operand 1	Operand 2	Result

Head → W Instr A, W Instr B, W Instr C, W Instr D

Tail → E Instr

Decode Unit — Insert an instr into ROB

Resolve branches

Get a ready ALU instr → ALU Unit(s)

Put ALU instr results in ROB

Get a ready MEM instr → MEM Unit(s)

Put MEM instr results in ROB

But, what about all the potential race conditions?

- ◆ Reading from the register file at the same time a separate instruction is writing back to the same location
  - ■ Which value to read?
- ◆ An instruction is being inserted into the ROB simultaneously to a dependent upstream instruction's result coming back from an ALU
  - ■ Put a tag or the value in the operand slot?
- ◆ An instruction is being inserted into the ROB simultaneously to A branch mis-prediction must kill the mis-predicted instructions and restore a "consistent state" across many modules

---

Rule Atomicity

- ◆ Lets you code each operation in isolation
- ◆ Eliminates the nightmare of race conditions ("inconsistent state") under such complex concurrency conditions

Insert Instr in ROB
- Put instruction in first available slot
- Increment
- Get source
  - RF <

Dispatch Instr
- Mark instruction dispatched
- Forward t
unit

Write Back Results to ROB
- Write back results to instr result
- Write back tags
- Set to done

Commit Instr
- Write results to register file (or allow write for sto
- Set to Emp
- Increment

Branch Resolution
- ...
- ...
- ...
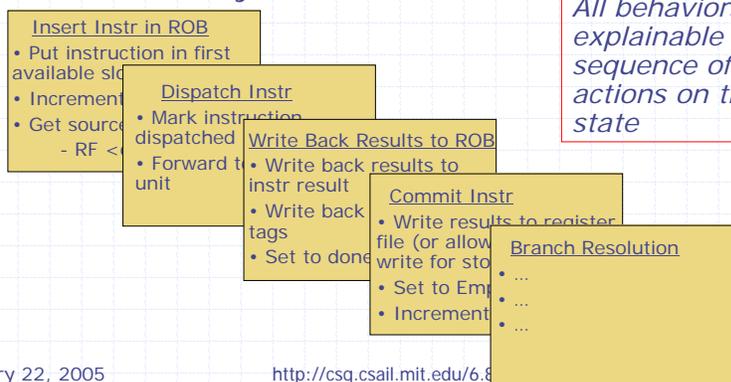
All behaviors are explainable as a sequence of atomic actions on the state

Synthesizable model of IA64

CMU-Intel collaboration

- ◆ Develop an Itanium μarch model that is
  - concise and malleable
  - *executable and synthesizable*
- ◆ FPGA Prototyping
  - XC2V6000 FPGA interfaced to P6 memory bus
  - Executes binaries natively against a real PC environment (i.e., memory & I/O devices)
- ◆ An evaluation vehicle for:
  - Functionality and performance: a fast μarchitecture emulator to run real software
  - Implementation: a synthesizable description to assess feasibility, design complexity and implementation cost

Roland Wunderlich & James Hoe @ CMU
Steve Hynal(SCL) & Shih-Lien Liu(MRL)

IA64 in Bluespec Wunderlich & Hoe
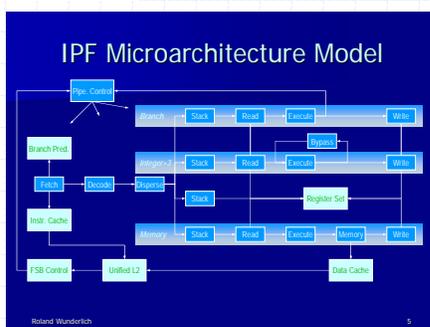
The model was developed in a few months by one student!