Architectural Synthesis and Exploration using Term Rewriting Systems

Arvind
James C. Hoe

Laboratory for Computer Science
Massachusetts Institute of Technology
http://www.csg.lcs.mit.edu

Outline

- Introduction
- Term Rewriting Systems (TRS) as a Hardware Description Language
- Hardware Synthesis from Term Rewriting Systems
- Results

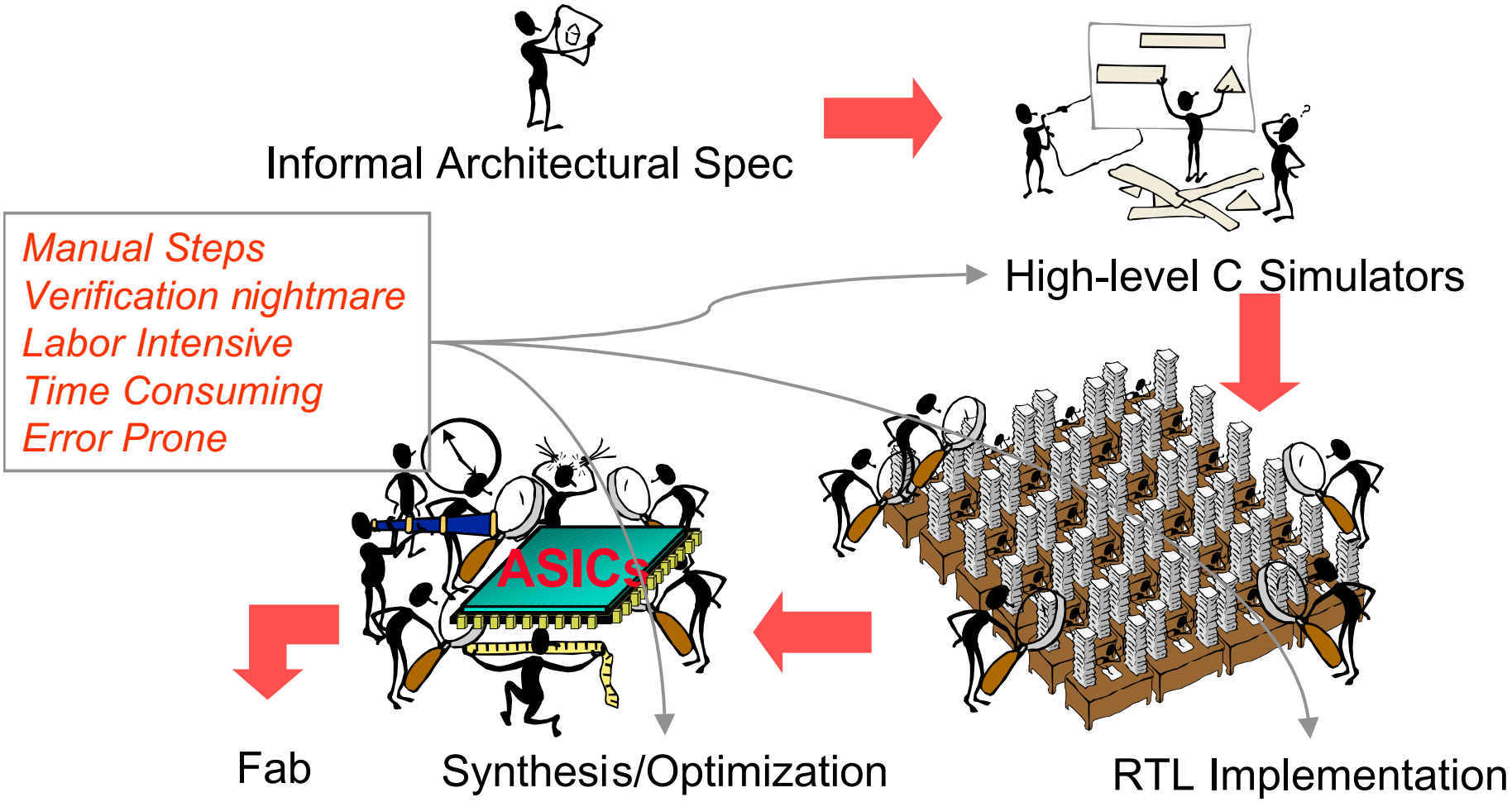
Internet/Communication Space

- Rapidly changing functionality and performance requirements necessitate rapid hardware development
  - ATM, frame-relay, Gigabit Ethernet, packet-over-SONET protocols
  - voice-over-IP, video, streaming data,

    QoS issues dominant
  - merger of LAN and WAN infrastructures
- Currently addressed by
  - General-purpose or Embedded processors + ASICs
  - Network processors *(emerging)*

ASIC development time and cost is the limiting factor in product release

Current ASIC Design Flow



Time pressure means:
little architecture exploration & high technology risk

Our New Design Technology

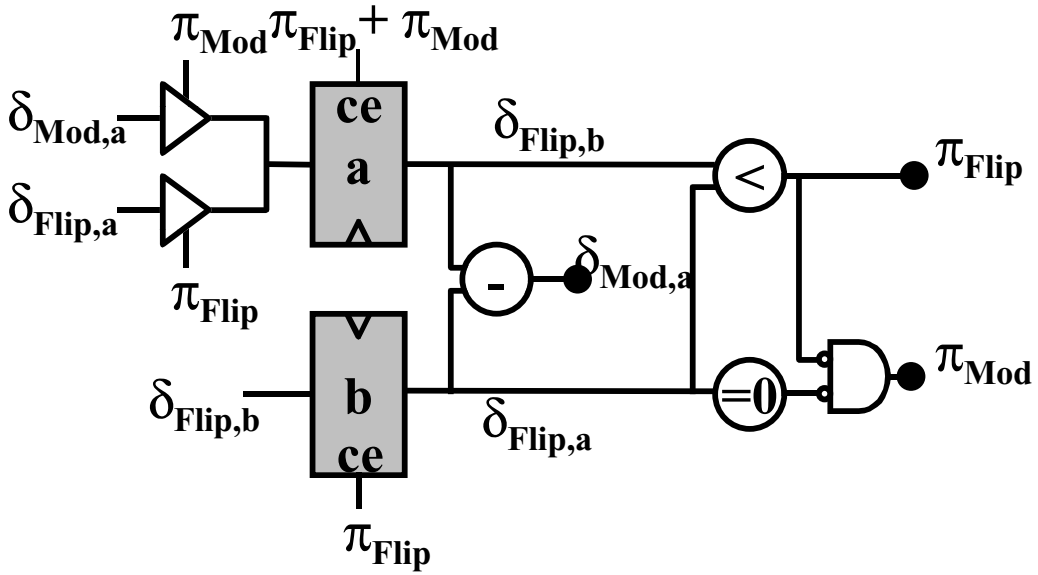
- Reduces time to market
  - Faster design capture
  - Same specification for simulation, verification and synthesis
  - Rapid feedback $\Rightarrow$ architectural exploration
- Enables rapid development of a large variety of chips with related designs

  $\Rightarrow$ complex systems-on-a-chip
- Reduces manpower requirement

Makes designing hardware as commonplace as writing software

State-Centric Descriptions

Schematics



Hardware description languages

```
always @ (posedge Clk) begin
        if (a >= b) begin
                a <= a - b;
                b <= b;
        end else begin
                a <= b;
                b <= a;
        end
end
```

what does it describe?

Operation-Centric Descriptions

Euclid's Algorithm

Gcd(a, b) if b≠0 ⇒ Gcd(b, Rem(a, b)) *(Rule$_1$)*

Gcd(a, 0) ⇒ a *(Rule$_2$)*

Rem(a, b) if a<b ⇒ a *(Rule$_3$)*

Rem(a, b) if a≥b ⇒ Rem(a-b, b) *(Rule$_4$)*

Execution:

Gc11d(2,4)

$\overset{R_3}{\Rightarrow}$ Gcd(4,2)

$\overset{R_4}{\Rightarrow}$ Gcd(2,Rem(2,2))

$\overset{R_3}{\Rightarrow}$ Gcd(2,0)

$\overset{R_1}{\Rightarrow}$ Gcd(4,Rem(2,4))

$\overset{R_1}{\Rightarrow}$ Gcd(2,Rem(4,2))

$\overset{R_4}{\Rightarrow}$ Gcd(2,Rem(0,2))

$\overset{R_2}{\Rightarrow}$ 2

Hardware description?

Operation-Centric Description:MIPS

MIPS Microprocessor Manual

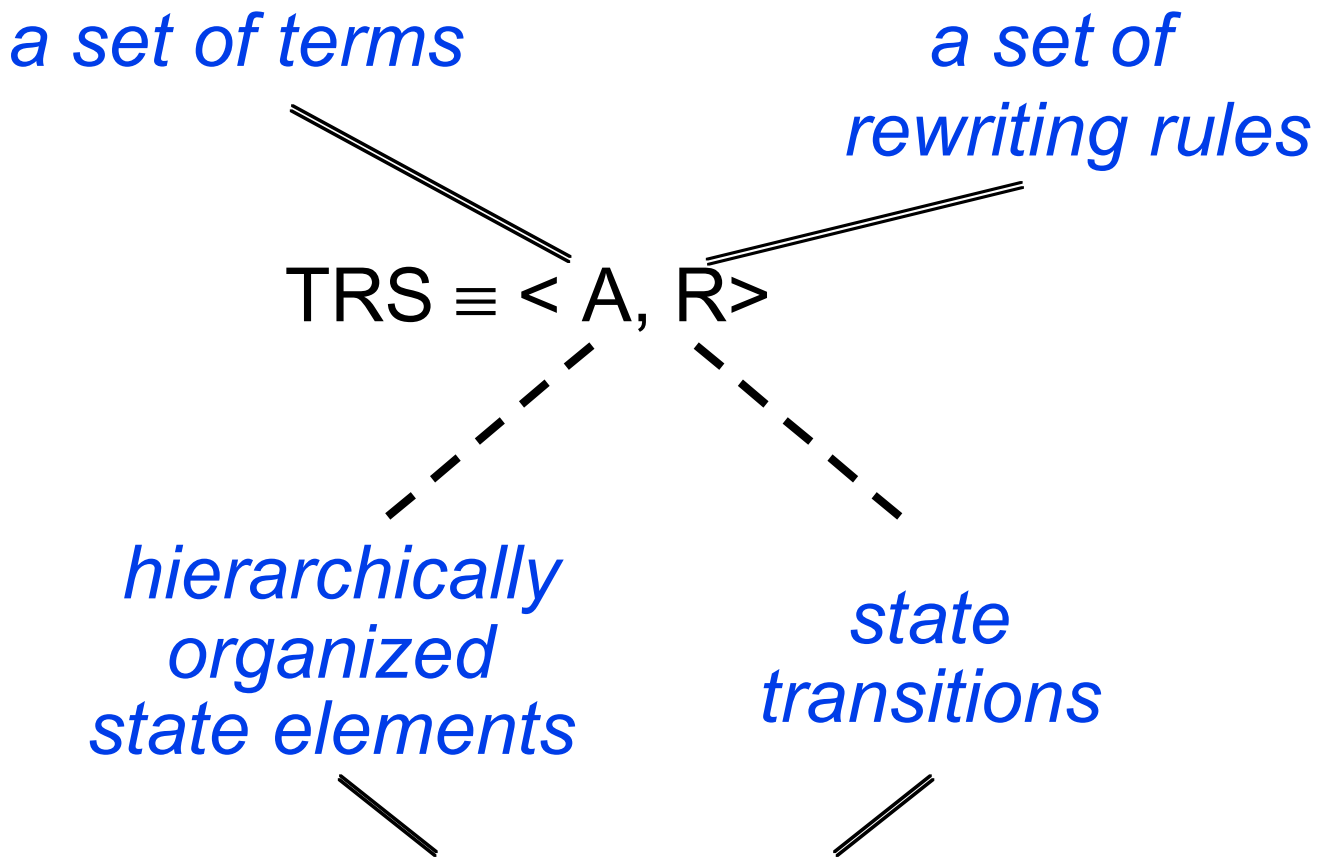
ADD rd, rs, rt

GPR[rd] ← GPR[rs] + GPR[rt]

PC ← PC + 4

TRS as a Hardware Description Language

Term Rewriting System



An operation centric view of the world

TRS Execution Semantics

Given a set of rules and an initial term s

While (some rules are applicable to s)

{

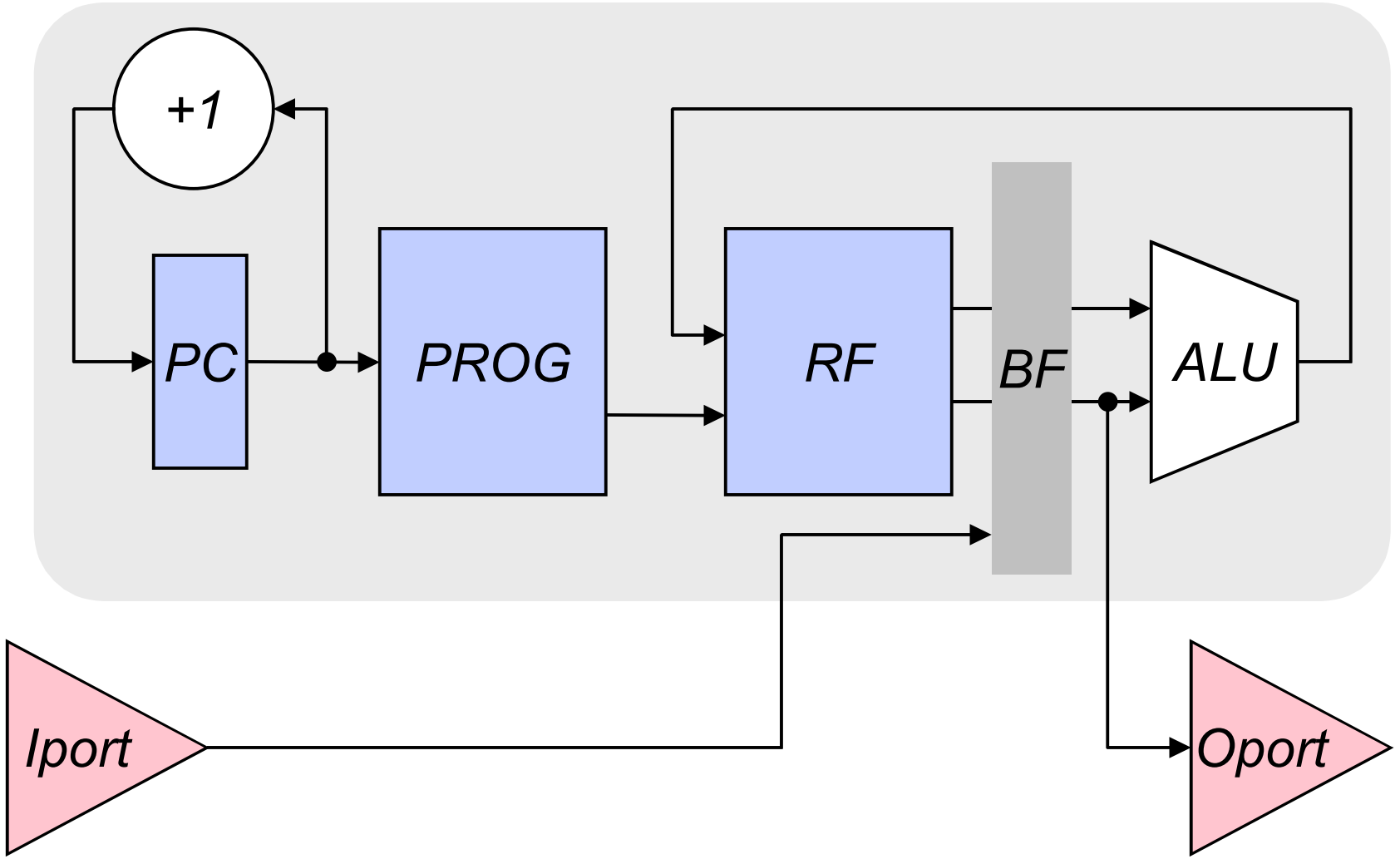
- choose an applicable rule

  (non-deterministic)

- apply the rule atomically to s

}

Architectural Description



AX Architectural Description

Type SYS = Sys(PROC, IPORT, OPORT)

Type PROC = Proc(PC, RF, PROG, BF)

Type PC = Bit[16]

Type RF = Array[RNAME] VAL

Type RNAME= Reg0 || Reg1 || Reg2 || . . .

Type VAL = Bit[16]

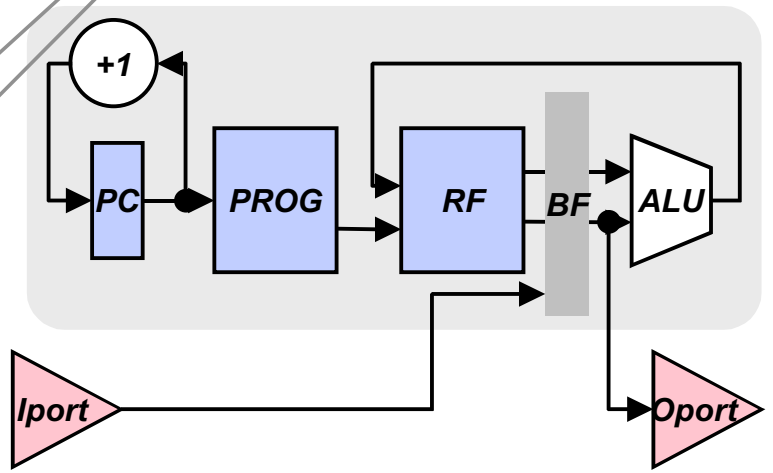
Type PROG = Array[PC] INST

Type BF = Fifo INST_D

Type IPORT = Iport VAL

Type OPORT= Oport VAL

Abstract Datatypes



AX Instruction Set

```
Type INST   =   Loadi (RD, VAL)
            ||  Loadpc (RD)
            ||  Add (RD, R1, R2)
            ||  Sub (RD, R1, R2)
            ||  . . .
            ||  Bz (RA,RC)
            ||  MovToO (R1)
            ||  MovFromI (RD)
```

Decoded instructions

Type INST_D = Add_d (RD, V1, V2) || ...

RD, RA, etc. are RNAME's. V1, V2, etc. are values

AX Processor Model: Fetch Rules

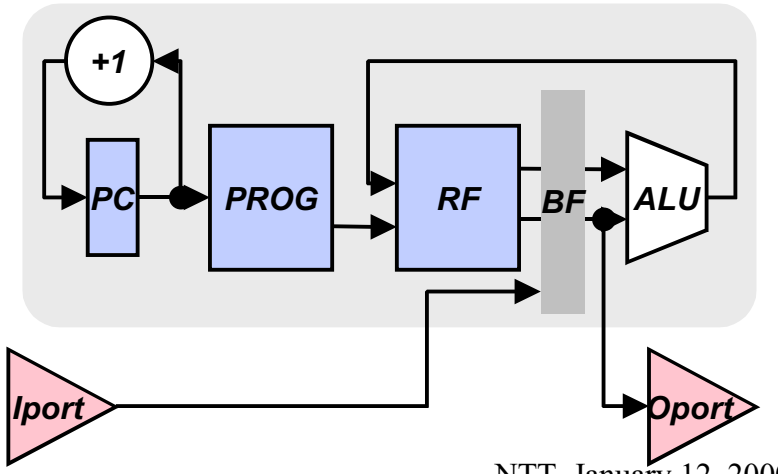
Fetch Add Rule

Proc(pc, rf, prog, bf)

if $r_1 \notin$ target(bf) $\wedge$ $r_2 \notin$ target(bf)

where Add(r, r_1, r_2)=prog[pc]

$\Rightarrow$ Proc(pc+1, rf, prog, enq(bf,Add_d(r,rf[r_1],rf[r_2])))



AX Processor Model: Execute Rules

Proc(pc, rf, prog, bf) *if* $r_1 \notin$ target(bf) $\wedge$ $r_2 \notin$ target(bf)

where Add(r, r_1, r_2)=prog[pc]

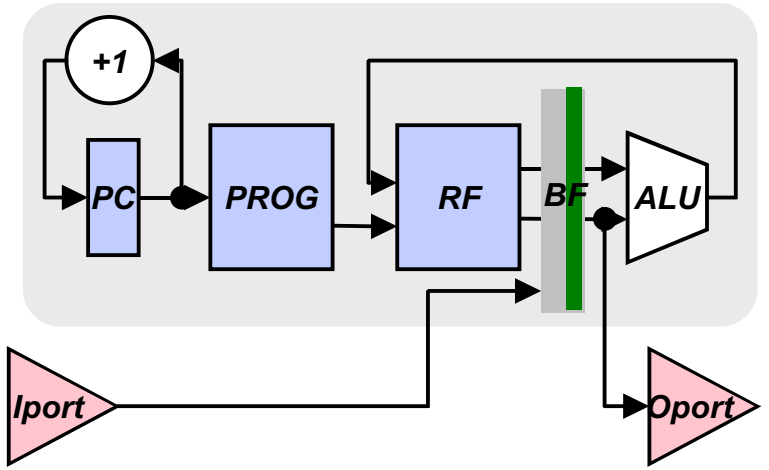
$\Rightarrow$ Proc(pc+1, rf, prog, enq(bf,Add_d(r,rf[r_1],rf[r_2])))

Proc(pc, rf, prog, bf)

where Add_d(r, v_1, v_2)=first(bf)

$\Rightarrow$ Proc(pc, rf[r:=v_1+v_2], prog, deq(bf))

"Execute Add"

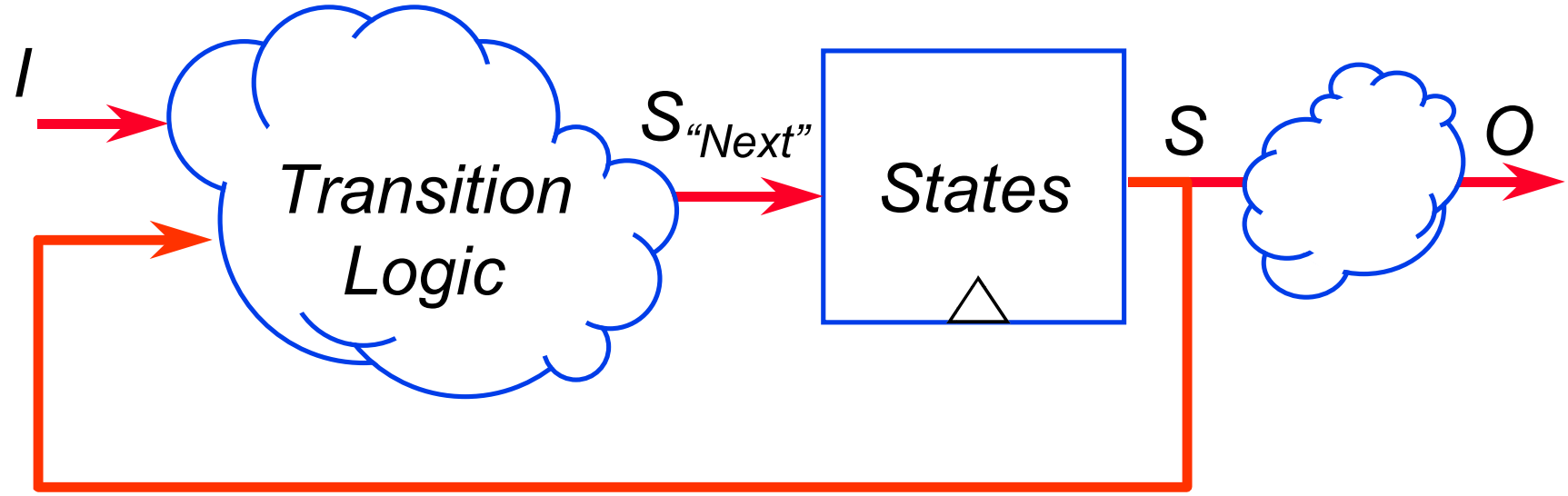


TRS as an HDL

- Clean, expressive, precise and concise
  - speculative & superscalar microarchitectures [IEEE Micro, June '99]
  - memory models & cache coherence protocols [ISCA99, ICS99]
- Supports parallel and non-deterministic specifications
- The correctness of a TRS can be verified against a reference TRS specification
- Some pipelining can be done automatically as a source-to-source transformation on TRS's
- Superscalar versions of TRS's can be derived mechanically from pipelined TRS's.

Synthesis from TRS's

From TRS to Synchronous FSM

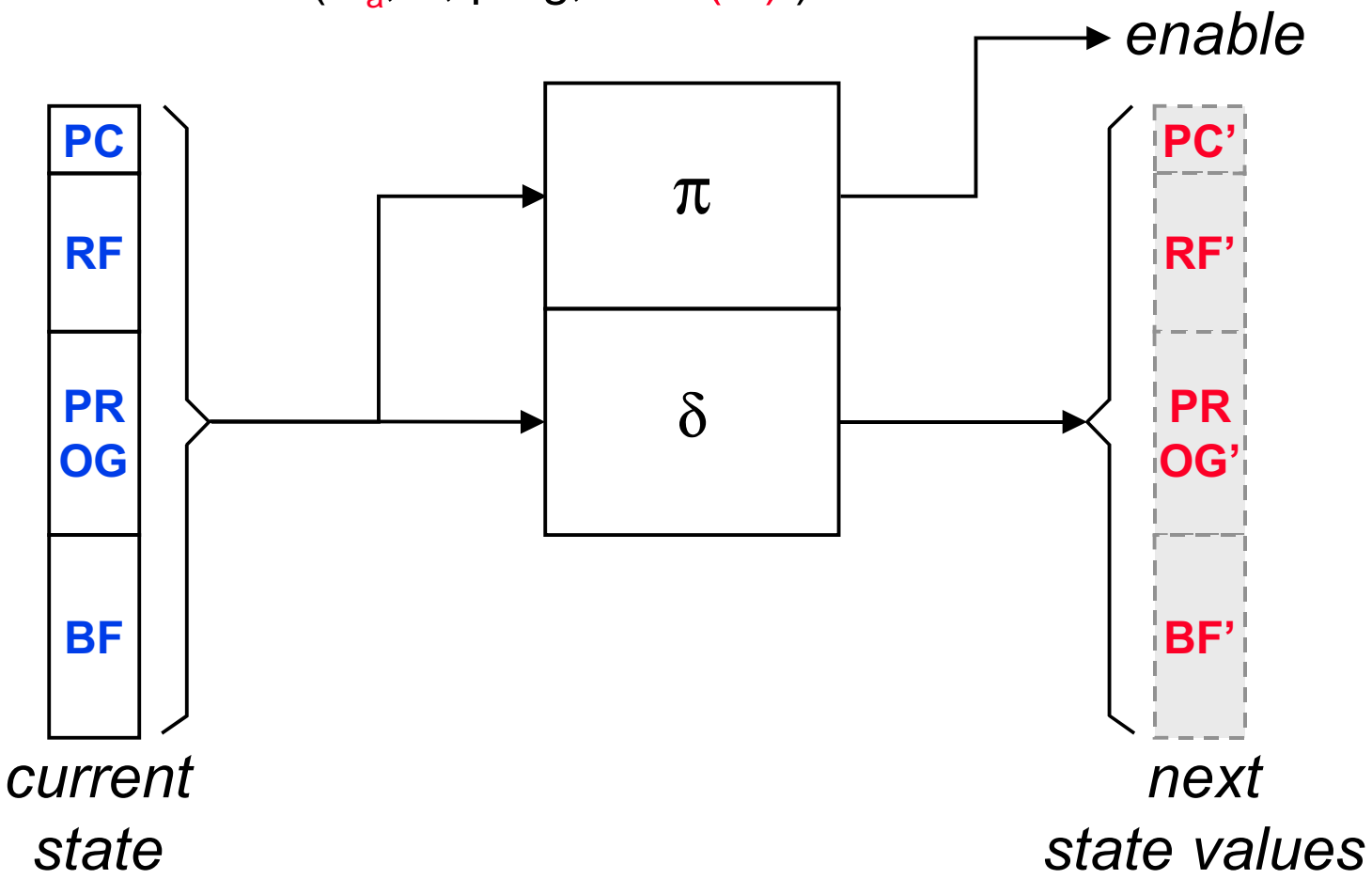


- Extract state elements (registers) from the type declaration
- Extract state transition logic from the rules

Rule: As a State Transformer

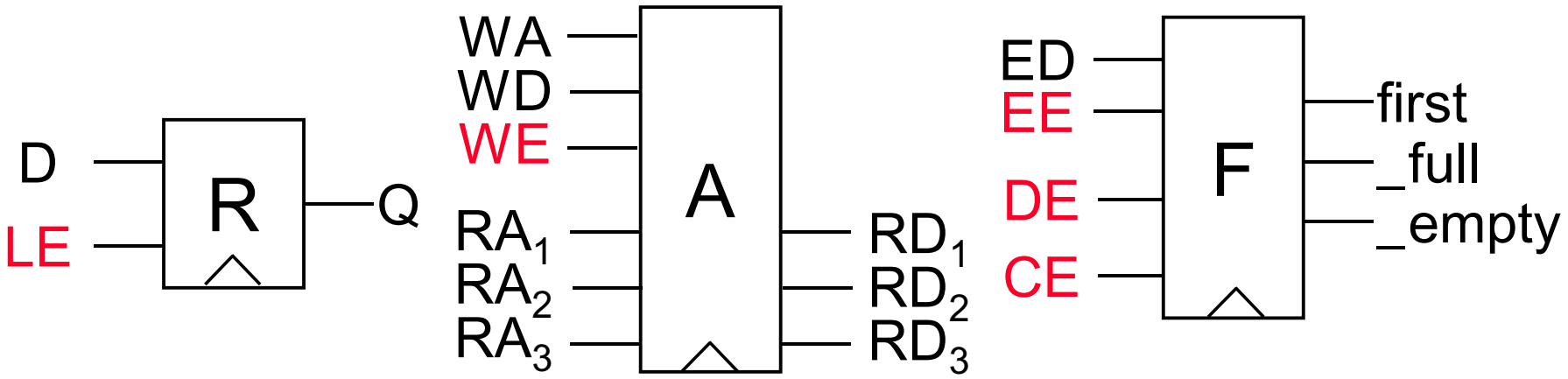
Proc(pc, rf, prog, bf) *where* $Bz_d(v_a, 0)$ = first(bf)

$\Rightarrow$ Proc(v_a, rf, prog, clear(bf))



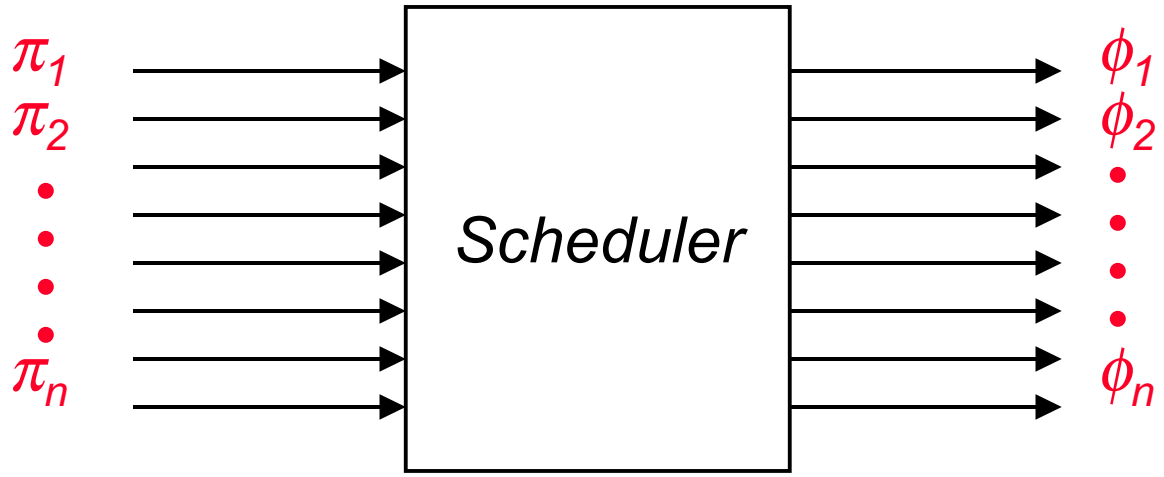
Reference Implementation

- Synchronous state elements



- Single transition per clock cycle

Scheduler

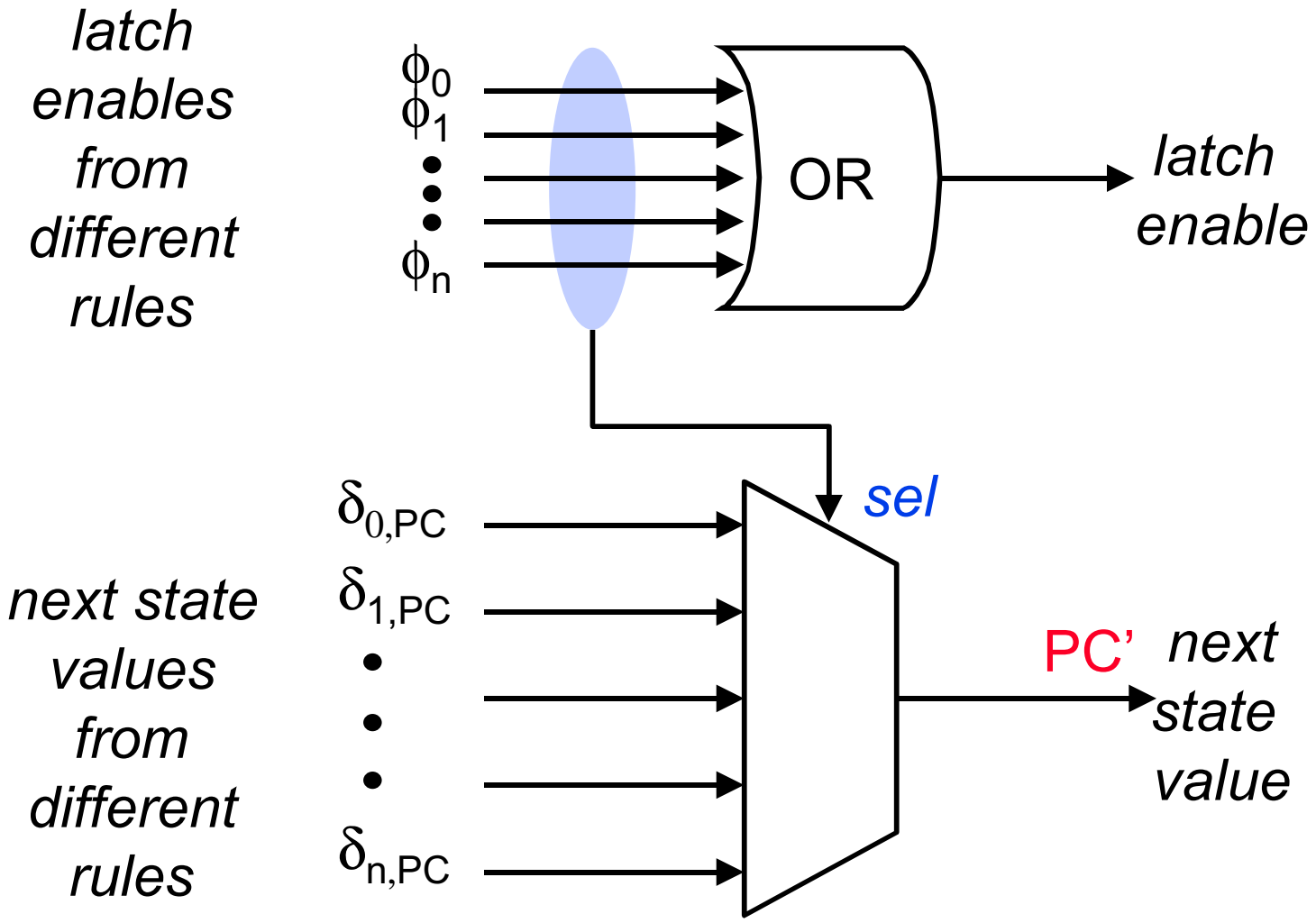


1. $\phi_i \Rightarrow \pi_i$

2. $\pi_1 \vee \pi_2 \vee .... \vee \pi_n \Rightarrow \phi_1 \vee \phi_2 \vee .... \vee \phi_n$

3. One-rule-a-time $\Rightarrow$ at most one ϕ_i is true

Combining Logic from Multiple Rules



Performance Considerations

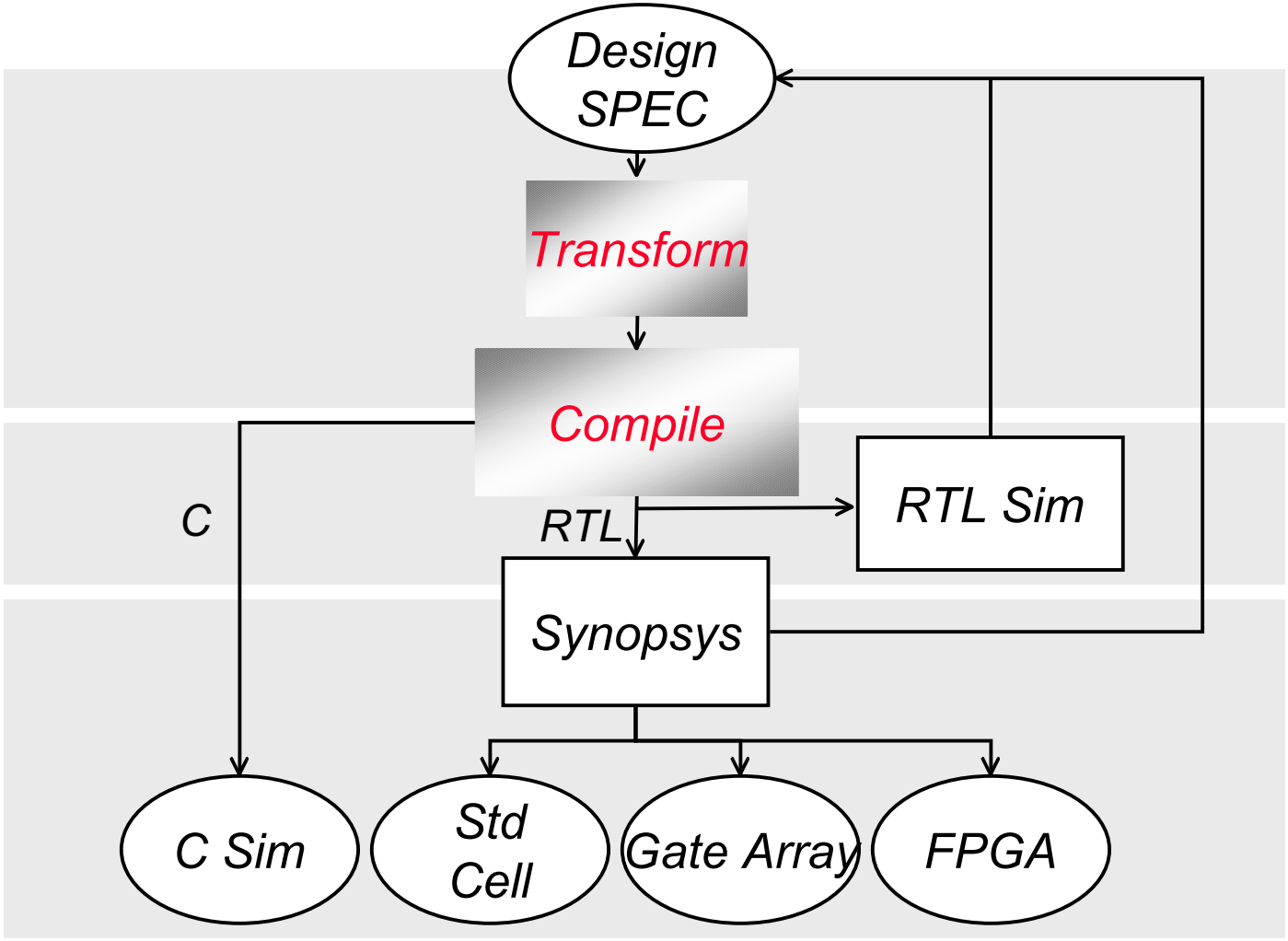
- Concurrent Execution
  - Statically determine which transitions can be safely executed concurrently
  - Generate a scheduler and update logic that allows as many concurrent transitions as possible

Caution: Concurrent firing of two rules can violate one-transition-at-a-time semantics if, for example, firing of one rule disables the other

Conflict-free rules

Quality of Synthesis

TRAC Synthesis Flow



Performance: TRS vs. Verilog

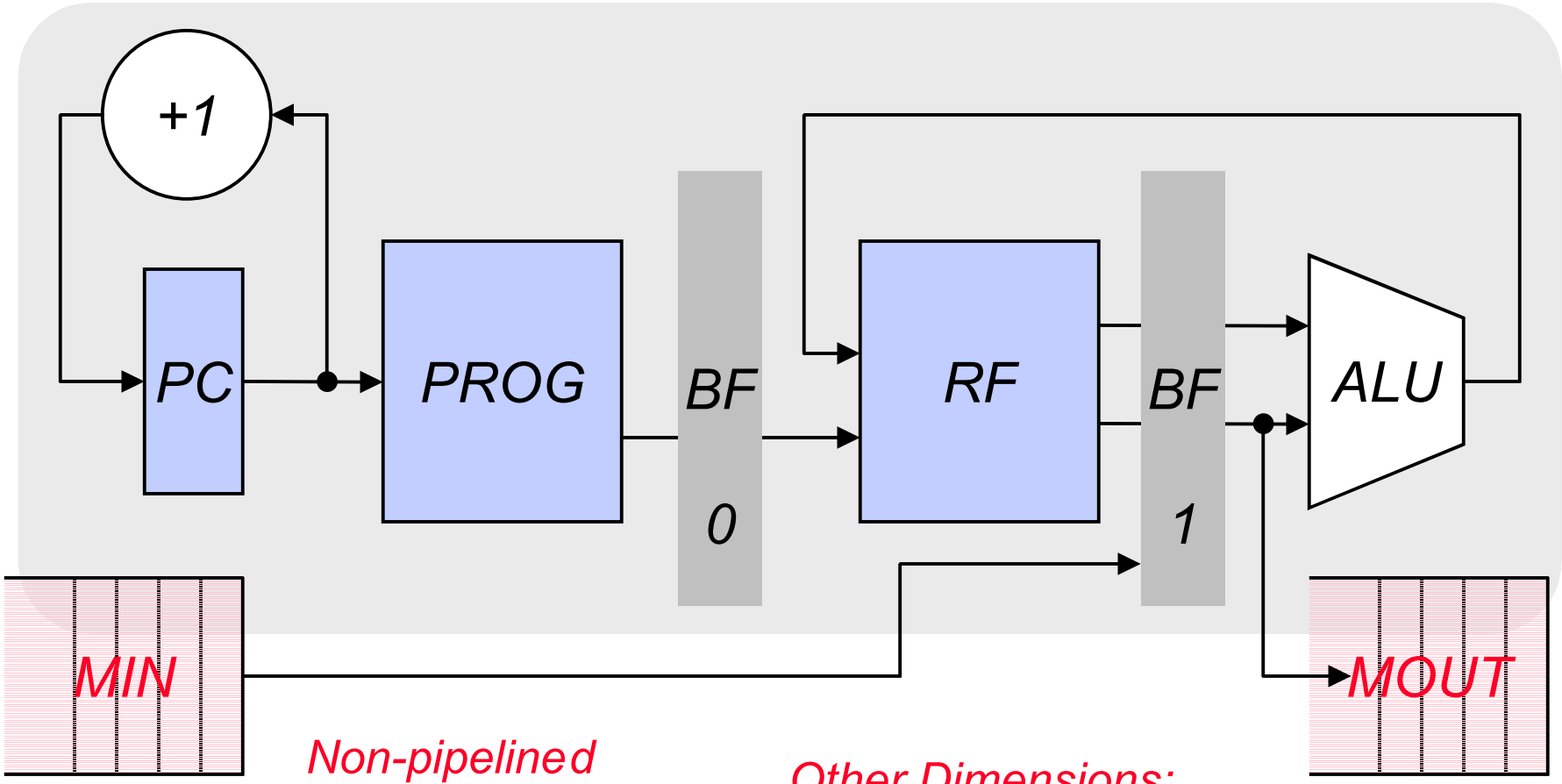
32-bit MIPS Integer Core

	CBA tc6a		LSI 10K	
	Area (cells)	Clock	Area (gates)	Clock
TRS	9521	10ns 100MHz	30756	19.48ns 51MHz
Verilog RTL	8960	11.4ns 88MHz	29483	23.79ns 42MHz

TRS 1 day
Verilog 1 month

Dan Rosenband & James Hoe

Architectural Derivatives



Non-pipelined

2-stage

3-stage

Other Dimensions:
Superscalar, Custom Instructions,
Number of Registers, Word Size ...

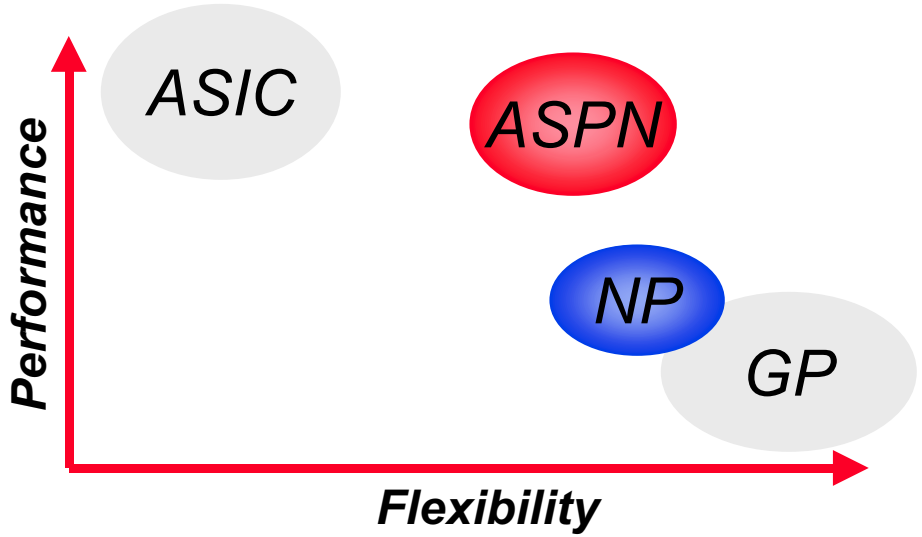
Derivatives and Feedback

- Derivatives of a 32-bit 4-GPR embedded RISC processor
- Synopsys RTL Analyzer reports GTECH area and gate delays *(no wiring or load model)*

	simple	2-stage	3-stage	3-stage,2-way
Delay	30+X	max(18+X,25)	max(6+X,25)	max(8+X,31)
Delay(X=20)	50	38	26	31
Area	4334	5753	6378	9492

unit area=1 NAND unit delay=1 NAND

Application: ASPN Chips



Application-Specific Programmable Network (ASPN) Chips are based on a core architecture and a set of domain-specific building blocks

TRAC allows rapid customization of ASPN designs with ASIC like performance for evolving needs and for different vertical markets within the communication space