A High-Performance Microarchitecture with Hardware-Programmable Functional Units

(missing Harvard shield)

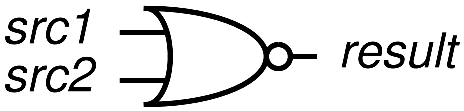
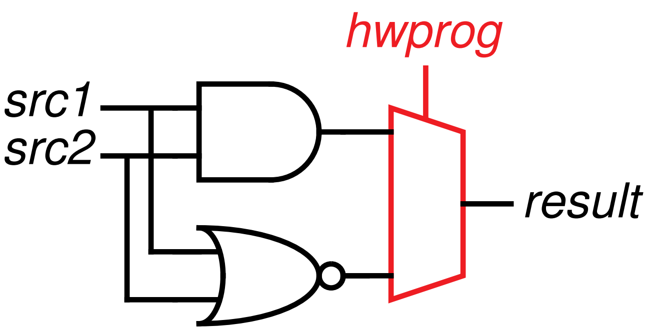
Rahul Razdan
Digital Equipment Corporation

Michael D. Smith
Harvard University

Why Programmable Logic?

- **Bring benefits of application-specific computing to general-purpose systems**
  - 25x speedup for long int multiplication
  - over 100x speedup for logic emulation

- **Tailor hardware to match problem**

	Datapath functional unit	*Operation to perform c = a & b*
Fixed FU		op t1= a,a *# invert a* op t2= b,b *# invert b* op t3= t1,t2 *# AND(a,b)*
H/W-prog FU		ld_hwprog *# setup AND* op c= a,b *# AND(a,b)*

Typical Approaches

- **Full FPGA design**

  e.g., VLIW with hardware-reprogrammable FUs
  (Iseli and Sanchez, 1993)

  + extremely flexibility
  + large potential performance benefit
  – low clock rates for common operations, e.g. 32-bit add
  – hand-optimized

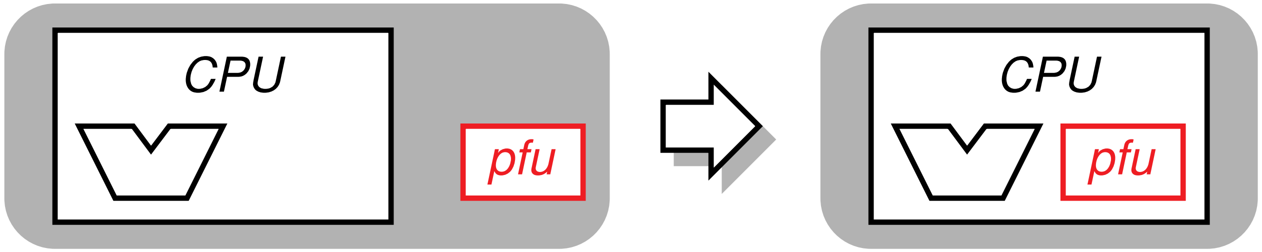
- **FPGA coprocessor or I/O-bus board**

  e.g., XILINX chips on TurboChannel board (Bertin et al., 1989)

  + augmented workstation performance
  – high communication overhead
  – hand-optimized or limited automation

Our Approach

- **Tightly integrate hardware-programmable resources into existing high-performance datapath**



  - augment, not replace, datapath functionality
  - lower overhead to use h/w-programmable FU (PFU)
  - wider applicability at expense of “peak” perf. benefit

- **Automatically and dynamically configure the PFU**
  - automatically manage “problems” with PFU
  - dynamically load PFU to support reuse

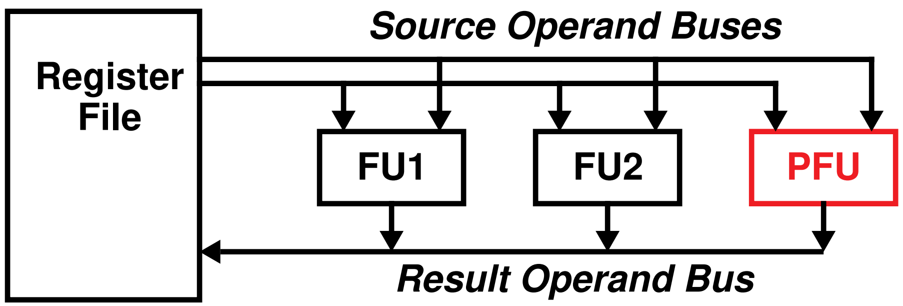
Outline

- PFU design and microarchitecture
- ISA extensions
- Compilation system and techniques
- Experimental results
- Summary

PRISC-1 Microarchitecture

- **Wide range of design choices possible**

- **Constraints on current microarchitecture**
  - single combinational PFU in single-issue pipeline

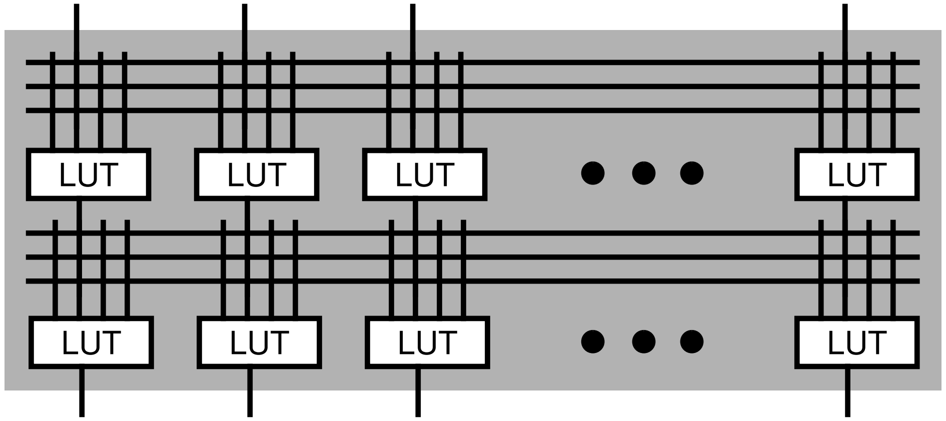


  - looks like another ALU from user-level
    - 2 source operands, 1 result
    - single cycle latency
  - relatively small amount of hardware
  - limited impact on cycle time

Design of a Combinational PFU

- **Similar to FPGA logic based on SRAM-fuse technology**
- **However, combinational design requires only**
  - logic evaluation units
  - interconnect resources

Inputs from operand buses



Outputs to result bus

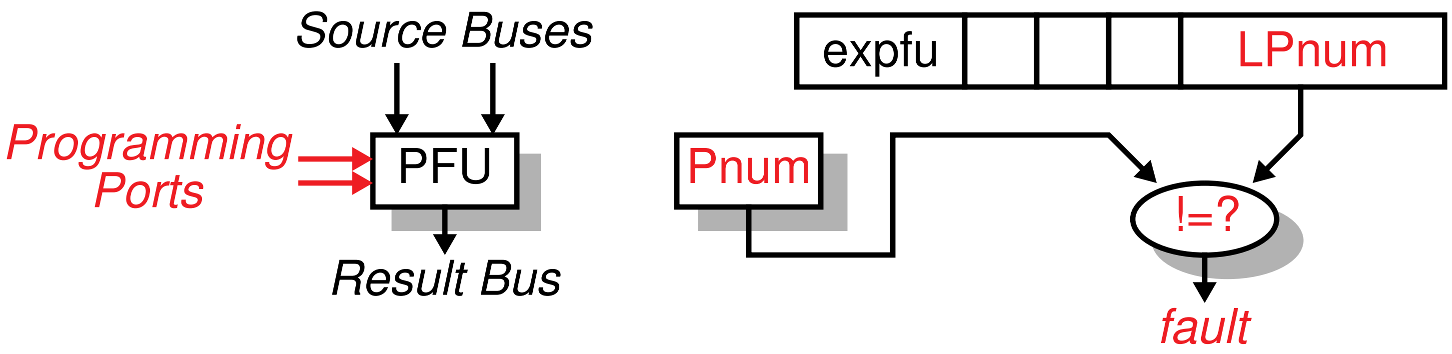
- **Current design**
  - contains three layers
  - is symmetric, i.e. same hardware support for each bit position
  - requires ~30K transistors for a 32-bit datapath (< 1KB SRAM)

ISA Extensions

- **Added a single user-level instruction which operates PFU**

6	5	5	5	11
expfu	rs	rt	rd	LPnum

  - LPnum indicates operation to perform
  - hardware check for correct programming

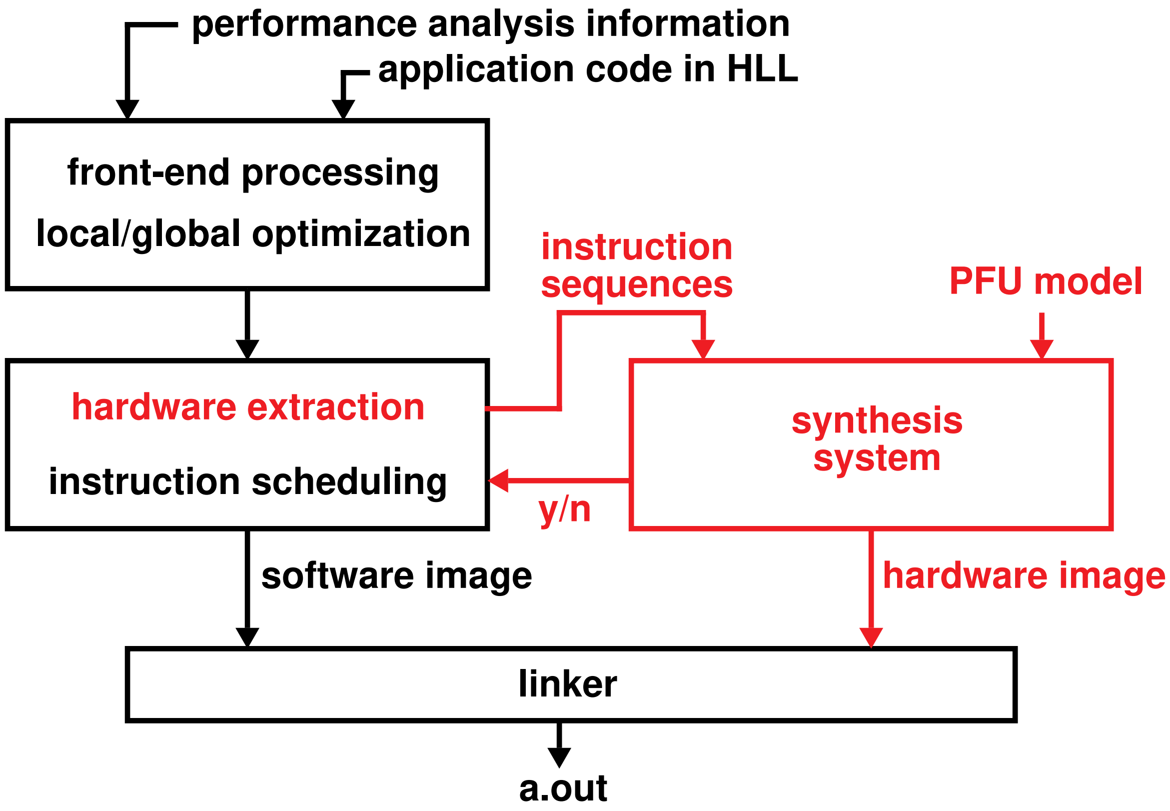


  - if mismatch, TLB-like fault

- **Wide range of PFU programming options**
  - tradeoff programming latency for hardware complexity
  - simulations use high-latency alternative—500 cycles to program

Compilation System

- Overall architecture



- Automation through coupling of compilation and logic synthesis techniques

Function Width Analysis (FWA)

- **Key observations for effective hardware extraction**
  - different instructions have different hardware costs

32-bit ADD **>** ***more complex than*** *8-bit* ADD **>** ***more complex than*** *32-bit* OR

  - PFU is less efficient than custom FU

- **Function Width Analysis (FWA)**
  - estimates the hardware cost of software operations
  - limits space searched by hardware extractor
  - is an iterative dataflow algorithm based on ternary algebra

- **Enables wide array of data flow and control flow optimizations**

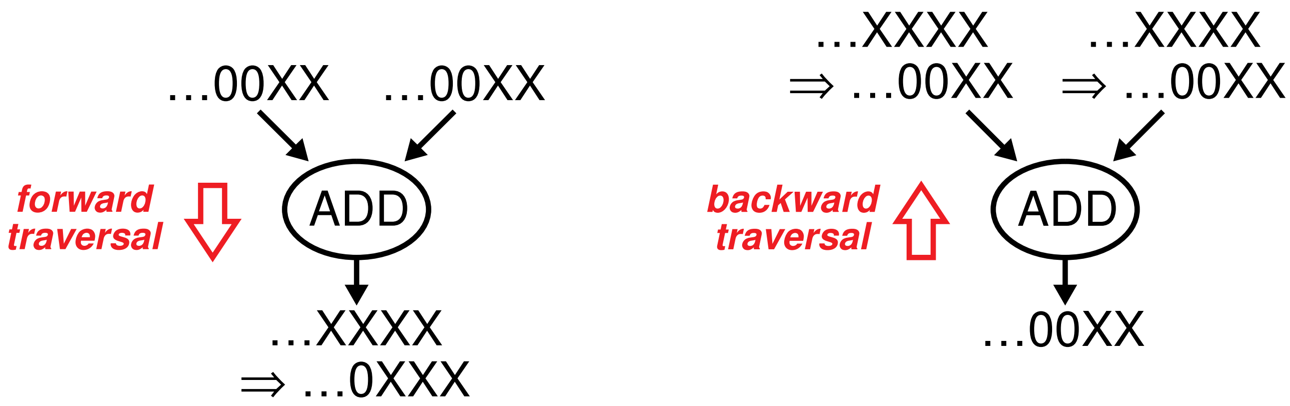
Basic Algorithm for FWA

Initialize instr output = XX…X; /* i.e. each op affects all 32-bits */

While (changes) {

Do forward traversal trying to fix output bits;

Do backward traversal trying to fix input bits;



}

Mark short and logically simple instructions;

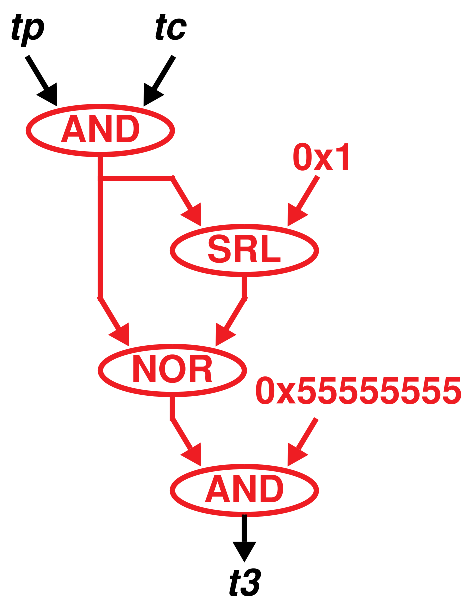
Hardware Extraction Techniques (1)

- **Data flow optimizations**
  - convert sequences of logical, shift, and test operations
  - recognize table lookups: shift, OR, add, and load
- **Example: PFU-expression (in cofactor.c of espresso)**

Original Code

```
#define DISJOINT 0x55555555
    lui   t5, 0x5555
    ori   t5, t5, 0x5555
    …
x = p[w] & c[w];
    lw    tp, 0(pw)
    lw    tc, 0(cw)
    and   tx, tp, tc
if (~(x|x>>1) & DISJOINT) goto false;
    srl   t1, tx, 1
    nor   t2, tx, t1
    and   t3, t2, t5
    bne   t3, zero, false
    …
```

PFU Logic



Hardware Extraction Techniques (1)

- **Example: PFU-table-lookup (from logic simulation domain)**

Truth Table

A	B	NAND
00	XX	00
01	00	00
01	X1	10
01	1X	10
10	00	00
10	01	10
10	10	01
10	11	11
11	00	00
11	X1	11
11	1X	11

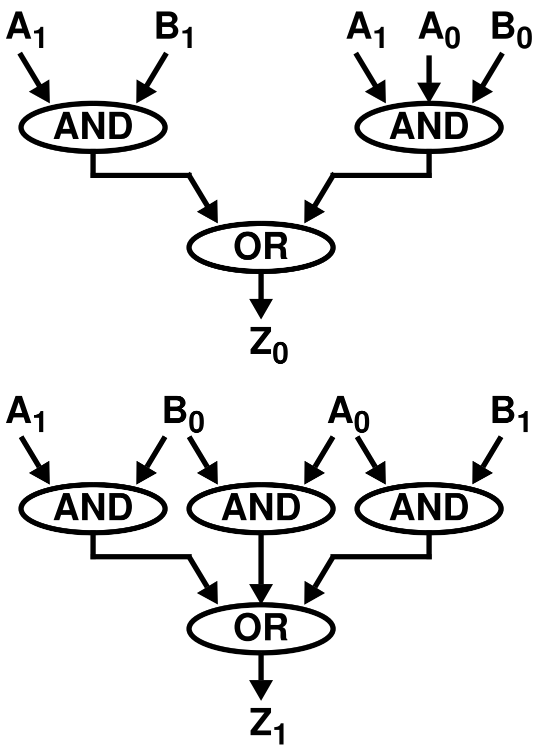
MIPS Code

```
sll   r1, A, 2
or    r1, B, r1
add   r1, T, r1
ldb   Z, 0(r1)
```

Key:

00	*illegal state*
01	*logic zero*
10	*logic one*
11	*logic X*

PFU logic



single EXPFU operation

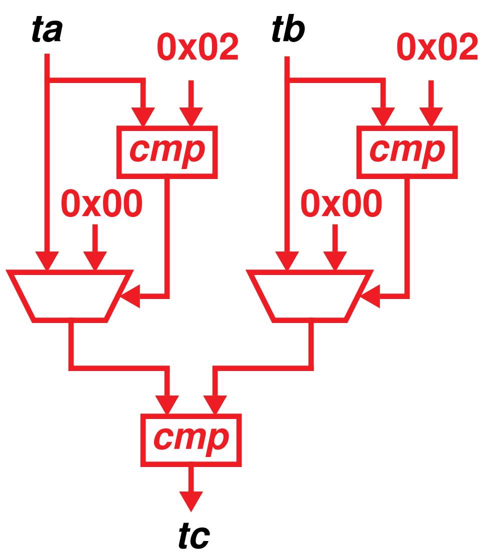
Hardware Extraction Techniques (2)

- **Control flow optimizations**
  - generate selects and predicated execution
  - build switches out of if-then-else trees
- **Example: PFU-predication (in cmppt() of eqntott:pterm_ops.c)**

Original Code

```
    li      t2, 2;
aa = a[0]->ptand[i]; bb = b[0]->ptand[i];
    lh      ta, 0(ap);
    lh      tb, 0(bp);
if (aa == 2) aa = 0;
    bne     ta, t2, next_if
    nop
    li      ta, 0
if (bb == 2) bb = 0;
    bne     tb, t2, next_if
    nop
    li      tb, 0
if (aa != bb) …
    beq     ta, tb, lab ⇒ bne  tc, zero, lab
    . . .
```

PFU Logic



Hardware Extraction Techniques (3)

- **Use loop unrolling to pack short vector operations**

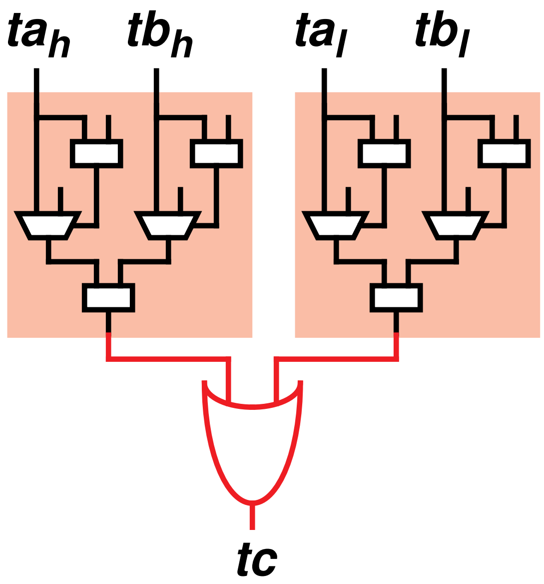
cmppt() Code

```
for (i = 0; i < n; i++) {
    li      i, 0
    ...
L:  lh      ta, 0(ap);
    lh      tb, 0(bp);
    expfu   tc, ta, tb
    bne     tc, zero, exit
    nop
    addiu   i, i, 1
    slt     tx, i, n
    addiu   ap, ap, 2
    bne     tx, zero, L
    addiu   bp, bp, 2
}
    ...
exit:
    ...
```

⇨

```
for (i = 0; i < n; i++) {
    li      i, 0
    ...
L:  lw      ta, 0(ap);
    lw      tb, 0(bp);
    expfu   tc, ta, tb
    bne     tc, zero, exit
    nop
    addiu   i, i, 2
    slt     tx, i, n
    addiu   ap, ap, 4
    bne     tx, zero, L
    addiu   bp, bp, 4
}
    ...
exit:
    [epilog code]
    ...
```

PFU Logic



Other PFU Applications

- **Short set vector operations — e.g. character datatype operations**
  - implement operations over ASCII character set
  - seen a local performance gain of 500% for toupper()

- **Serial operations on packed short set vectors**
  - e.g. test for end of string at a word granularity
  - packing improves efficiency of
    - data transfer
    - memory storage

- **Hashing schemes**

- **… the search continues …**

Proof-of-concept Experiment

- **Implemented fcn-width analysis and h/w-extraction routines**
  - used MIPS executable as input to routines
  - sophistication limited by
    - lack of complete type and variable information
    - lack of code scheduling optimizations
- **Implemented h/w synthesis down to LUT place-and-route**
- **“Added” a PFU to a mythical 200MHz MIPS R2000 datapath**
- **Simulated program execution, results**
  - include PFU programming overhead (500 cycles)
  - do not include cache or multitasking effects

Experimental Results

- **Static PFU optimization instances**

PFU Optimization	compress	eqntott	espresso	gcc	lisp	sc
expression	9	0	48	13	4	12
table-lookup	0	0	0	0	0	0
predication	0	1	0	13	0	0
jump-register	10	0	47	103	0	35
loop-unrolling	0	3	0	4	0	0
TOTAL	19	4	95	133	4	47

- **Overall speedup due to PFU resource**

	compress	eqntott	espresso	gcc	lisp	sc
Speedup	1.15	1.91	1.16	1.10	1.06	1.12

- **Average PFU performance gain > gain from a 1KB increase in an 8KB instruction cache**

Related Work

- **Differs from other automatic ISA work**
  - augment existing ISAs rather than build one from scratch
  - adapt to as-yet-unseen applications

- **Differs from work in writable microcode**
  - optimize at lower level — i.e. gate-level
  - redefine datapath primitives

- **Differs from other work which utilizes programmable logic**
  - most are not focused on a general-purpose solution
  - Athanas and Silverman [1993] similar but they implement
    - board-level coprocessor-like unit
    - synthesis granularity at level of C function

Summary

- **Described a high-performance microarchitecture with a PFU**

- **Coupled compilation and logic synthesis techniques to**
  - seamlessly integrate PFU resources with existing datapaths
  - automatically and dynamically augment the base instruction set
  - develop a cost-effective and general-purpose approach

- **Achieved an average 22% performance improvement**
  - through “clumping” of instructions with low hardware complexity
  - with relatively small hardware budget