

FIFO and Concurrency Issues

Nirav Dave

Computer Science & Artificial Intelligence Lab
Massachusetts Institute of Technology

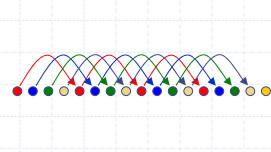
November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-1

Longest Prefix Match for IP lookup: 3 possible implementation architectures

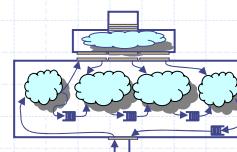
Rigid pipeline



Inefficient memory usage but simple design

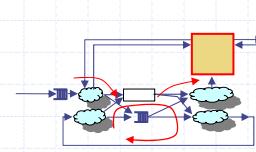
Designer's Ranking:

Linear pipeline



Efficient memory usage through memory port replicator

Circular pipeline



Efficient memory with most complex control

Which is "best"?

Arvind, Nikhil, Rosenband & Dave ICCAD 2004

L18-2

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%

Bluespec results can match carefully coded Verilog

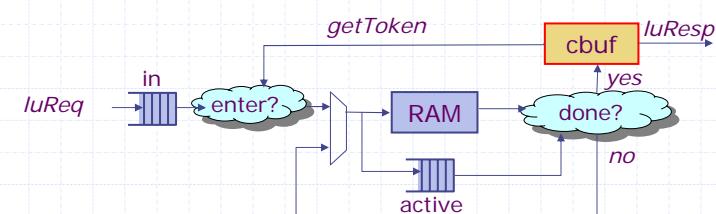
Compiler Argument: Within Reason

- Micro-architecture has a dramatic impact on performance

V = Verilog; BSV = Bluespec System Verilog

L18-3

Circular pipeline



Completion buffer

- gives out tokens to control the entry into the circular pipeline
- ensures that departures take place in order even if lookups complete out-of-order

Circular Pipeline Code

```

rule enter (True);
    Token t <- cbuf.getToken();
    IP ip = inQ.first();
    ram.req(ip[31:16]);
    fifo.enq(tuple2(ip[15:0], t));
    inQ.deq();
endrule

```

When can these rules fire?

```

rule done (True);
    TableEntry p <- ram.resp();
    match {.rip, .t} = fifo.first();
    if (isLeaf(p)) cbuf.done(t, p);
    else begin
        fifo.enq(rip << 8, t);
        ram.req(p+signExtend(rip[15:8]));
    end
    fifo.deq();
endrule

```

November 14, 2006

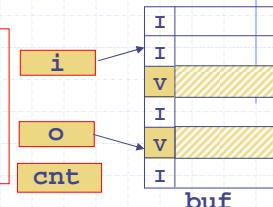
L18-5

Completion buffer

```

interface CBuffer#(type any_T);
    method ActionValue#(Token) getToken();
    method Action done(Token t, any_T d);
    method ActionValue#(any_T) getResult();
endinterface

```



```

module mkCBuffer (CBuffer#(any_T))
    provisos (Bits#(any_T,sz));
    RegFile#(Token, Maybe#(any_T)) buf <- mkRegFileFull();
    Reg#(Token) i <- mkReg(0); //input index
    Reg#(Token) o <- mkReg(0); //output index
    Reg#(Token) cnt <- mkReg(0); //number of filled slots
    ...

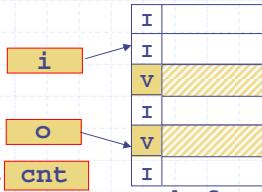
```

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-6

Completion buffer



```
... // state elements buf, i, o, n .. cnt
method ActionValue#(any_T) getToken()
    if (cnt <= maxToken);
        cnt <= cnt + 1; i <= i + 1;
        buf.upd(i, Invalid);
    return i;
endmethod
method Action done(Token t, any_T data);
    return buf.upd(t, Valid data);
endmethod
method ActionValue#(any_T) get() if (cnt > 0) &&
    (buf.sub(o) matches tagged (Valid .x));
    o <= o + 1;
    cnt <= cnt - 1;
    return x;
endmethod
```

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-7

Sharing Methods

- ◆ Both “done” and “recirc” make a memory request and enqueue into the fifo. How do we handle this sharing?
- ◆ Software approach: Make copies for each rule
 - How do we keep atomicity?
 - More analysis!
 - ◆ Does this scale?
- ◆ Hardware approach: Only one of them gets to go. They conflict
 - Which one get to go?
source annotations

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-8

Simple Example

```
rule a(p);
    fifo.enq(f(r0));
    r0 <= r0 + 1;
endrule

rule b(q);
    fifo.enq(f'(r1));
    r1 <= r1 + 1;
endrule
```

➡

```
rule a(p);
    fifo.enq(f(r0));
    r0 <= r0 + 1;
endrule

rule b(q & !p);
    fifo.enq(f'(r1));
    r1 <= r1 + 1;
endrule
```

November 14, 2006

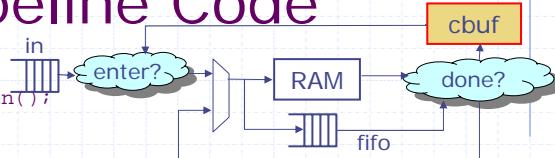
<http://csg.csail.mit.edu/6.827/>

L18-9

Circular Pipeline Code

```
rule enter (True);
    Token t <- cbuf.getToken();
    IP ip = inQ.first();
    ram.req(ip[31:16]);
    fifo.enq(tuple2(ip[15:0], t)); inQ.deq();
endrule

rule done (True);
    TableEntry p <- ram.resp();
    match {.rip, .t} = fifo.first();
    if (isLeaf(p)) cbuf.done(t, p);
    else begin
        fifo.enq(rip << 8, t);
        ram.req(p + signExtend(rip[15:7]));
    end
    fifo.deq();
endrule
```



Can rules enter and
done be applicable
simultaneously?

Which one should go?

What is the
concurrency
expectation for the
fifo?

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-10

One Element FIFO

```
module mkFIFO1 (FIFO#(t));
    Reg#(t)      data  <- mkRegU();
    Reg#(Bool)   full  <- mkReg(False);
    method Action enq(t x) if (!full);
        full <= True;
        data <= x;
    endmethod
    method Action deq() if (full);
        full <= False;
    endmethod
    method t first() if (full);
        return (data);
    endmethod
    method Action clear();
        full <= False;
    endmethod
endmodule
```

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-11

The good news ...

- ◆ It is always possible to transform your design to meet desired concurrency and functionality

How? Good Question!

November 14, 2006

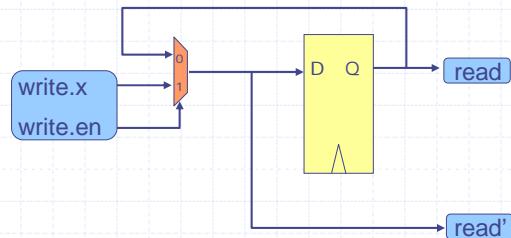
<http://csg.csail.mit.edu/6.827/>

L18-12

Register Interfaces

read < write

write < read ?



read' – returns the current state when write is not enabled
read' – returns the value being written if write is enabled

November 14, 2006

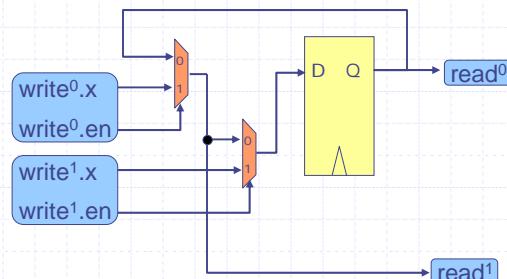
<http://csg.csail.mit.edu/6.827/>

L18-13

Ephemeral History Register (EHR)

[MEMOCODE'04]

read⁰ < write⁰ < read¹ < write¹ < ...



writeⁱ⁺¹ takes precedence over writeⁱ

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-14

Textual Description

```
method r0w0r1w1(w0,x0,w1,x1);
    let r0 = r
    let r1 = w0 ? x0 : r0;
    let r2 = w1 ? x1 : r1;
    if (w0 | w1)
        r <= r2;
endmethod
```

We've merged all
the methods
together – Do the
Rules also get
merged together?

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-15

The One Element FIFO now

```
module mkFIFO1 (FIFO#(t));
    Reg#(t)      data  <- mkRegU();
    Reg#(Bool)   full  <- mkReg(False);
    method ActionValue#(t)
        first_deq$enq(deq,enq,enqx) if (full);

        let full'  = (deq) ? False : full;
        let full'' = (enq) ? True  : full';
        if (enq)
            data <= enqx;
        if (deq | enq)
            full <= full'';
    endmethod
endmodule
```

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-16

Compositions

- ◆ This sort of composition can be generalized to arbitrary rules/methods
- ◆ Can add sequential composition as well

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-17

So now...

- ◆ Any questions?
- ◆ There's a quiz coming up

November 14, 2006

<http://csg.csail.mit.edu/6.827/>

L18-18