

Constructive Computer Architecture: Pipelining combinational circuits

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-1

Content

- ◆ Elastic versus Inelastic pipelines
- ◆ The role of FIFOs
- ◆ Concurrency issues
- ◆ Ephemeral History Registers (EHRs)

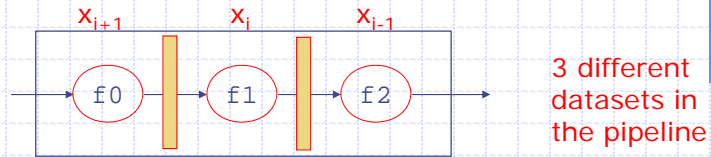
- ◆ BSV Concepts
 - The Maybe Type
 - EHR

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-2

Pipelining Combinational Functions



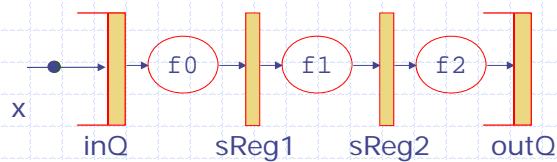
- ◆ Lot of area and long combinational delay
- ◆ Folded or multi-cycle version can save area and reduce the combinational delay but throughput per clock cycle gets worse
- ◆ Pipelining: a method to increase the circuit throughput by evaluating multiple inputs

September 19, 2016

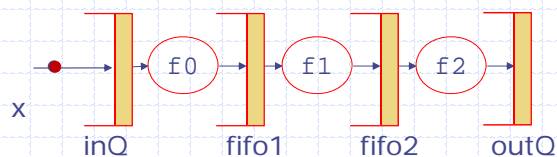
<http://csg.csail.mit.edu/6.175>

L06-3

Inelastic vs Elastic pipeline



Inelastic: all pipeline stages move synchronously



Elastic: A pipeline stage can process data if its input FIFO is not empty and output FIFO is not Full

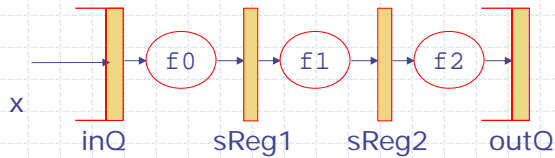
Most complex processor pipelines are a combination of the two styles

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-4

Inelastic pipeline



```

rule sync-pipeline;
  if(inQ.notEmpty &&
     outQ.notFull)
  begin inQ.deq;
    sReg1 <= f0(inQ.first);
    sReg2 <= f1(sReg1);
    outQ.enq(f2(sReg2))
  end
endrule

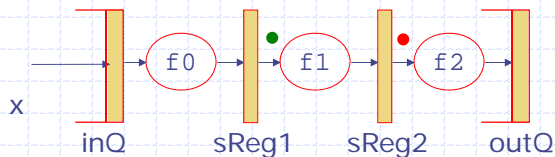
```

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-5

Pipeline bubbles



```

rule sync-pipeline;
  if(inQ.notEmpty &&
     outQ.notFull)
  begin inQ.deq;
    sReg1 <= f0(inQ.first);
    sReg2 <= f1(sReg1);
    outQ.enq(f2(sReg2))
  end
endrule

```

- Red and Green tokens must move even if there is nothing in inQ !
- Also if there is no token in $sReg_2$ then nothing should be enqueued in the $outQ$

Valid bits or
the Maybe type

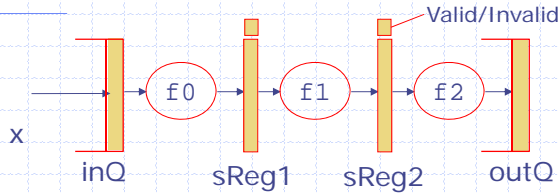
Modify the rule to deal with these conditions

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-6

Explicit encoding of Valid/Invalid data



```
typedef union tagged {void Valid; void Invalid;
} Validbit deriving (Eq, Bits);
```

```
rule sync-pipeline;
if(outQ.notFull || sReg2v != Valid)
if (inQ.notEmpty)
begin sReg1 <= f0(inQ.first); inQ.deq;
sReg1v <= Valid end
else sReg1v <= Invalid;
sReg2 <= f1(sReg1); sReg2v <= sReg1v;
if (sReg2v == Valid) outQ.enq(f2(sReg2))
endrule
```

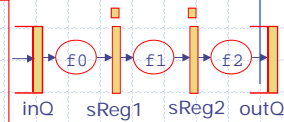
September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-7

When does the state change?

```
rule sync-pipeline;
if(outQ.notFull || sReg2v != Valid)
if (inQ.notEmpty)
begin sReg1 <= f0(inQ.first); inQ.deq;
sReg1v <= Valid end
else sReg1v <= Invalid;
sReg2 <= f1(sReg1); sReg2v <= sReg1v;
if (sReg2v == Valid) outQ.enq(f2(sReg2))
endrule
```



inQ	sReg1v	sReg2v	outQ
NE	V	V	NF
NE	V	V	F
NE	V	I	NF
NE	V	I	F
NE	I	V	NF
NE	I	V	F
NE	I	I	NF
NE	I	I	F

yes

No

inQ	sReg1v	sReg2v	outQ
E	V	V	NF
E	V	V	F
E	V	I	NF
E	V	I	F
E	I	V	NF
E	I	V	F
E	I	I	NF
E	I	I	F

NE = Not Empty; NF = Not Full

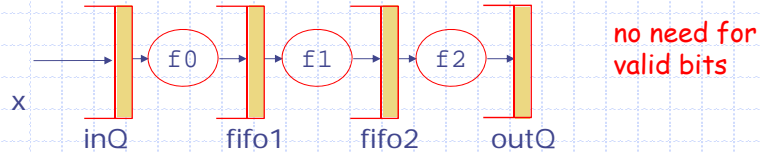
September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-8

Elastic pipeline

Use FIFOs instead of pipeline registers



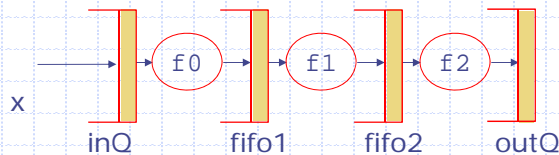
```

rule elasticPipeline;
  if(inQ.notEmpty && fifo1.notFull)
    begin fifo1.enq(f0(inQ.first); inQ.deq end;
  if(fifo1.notEmpty && fifo2.notFull)
    begin fifo2.enq(f1(fifo1.first); fifo1.deq end;
  if(fifo2.notEmpty && outQ.notFull)
    begin outQ.enq(f2(fifo2.first); fifo2.deq) end;
endrule

```

- ◆ When does the state change?
- ◆ Can tokens be left in the pipeline?

State Change conditions for the rule



inQ	fifo1	fifo2	outQ	act1	act2	act3
NE	NE,NF	NE,NF	NF	Yes	Yes	Yes
NE	NE,NF	NE,NF	F	Yes	Yes	No
NE	NE,NF	NE,F	NF	Yes	No	Yes
NE	NE,NF	NE,F	F	Yes	No	No
...				...		

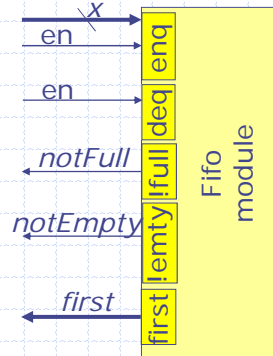
- ◆ The execution of this rule assumes that enq and deq methods of the FIFOs can be executed concurrently
- ◆ What if they cannot?

One-Element FIFO Implementation

```

module mkCFFifo (Fifo#(1, t));
  Reg#(t) d <- mkRegU();
  Reg#(Bool) v <- mkReg(False);
  method Bool notFull;
    return !v;
  endmethod
  method Bool notEmpty;
    return v;
  endmethod
  method Action enq(t x);
    v <= True; d <= x;
  endmethod
  method Action deq;
    v <= False;
  endmethod
  method t first;
    return d;
  endmethod
endmodule

```



1. What if enq and deq were executed together?
2. Can notEmpty and notFull be true simultaneously?

September 19, 2016

<http://csg.csail.mit.edu/6.175>

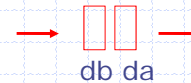
L06-11

Two-Element FIFO

```

module mkCFFifo (Fifo#(2, t));
  Reg#(t) da <- mkRegU();
  Reg#(Bool) va <- mkReg(False);
  Reg#(t) db <- mkRegU();
  Reg#(Bool) vb <- mkReg(False);
  method Bool notFull; return !vb; endmethod
  method Bool notEmpty; return va; endmethod
  method Action enq(t x);
    if (va) begin db <= x; vb <= True; end
    else begin da <= x; va <= True; end
  endmethod
  method Action deq;
    if (vb) begin da <= db; vb <= False; end
    else begin va <= False; end
  endmethod
  method t first; return da; endmethod
endmodule

```



Assume, if there is only one element in the FIFO it resides in da

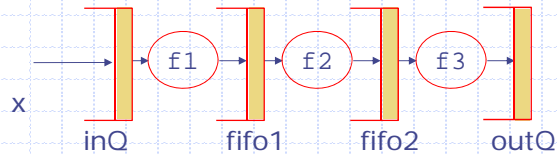
notEmpty and notFull can be true simultaneously

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-12

Concurrent method calls



```
rule stage1;  
  if(inQ.notEmpty && fifo1.notFull)  
    begin fifo1.enq(f1(inQ.first)); inQ.deq end;  
  if(fifo1.notEmpty && fifo2.notFull)  
    begin fifo2.enq(f2(fifo1.first)); fifo1.deq end;  
  if(fifo2.notEmpty && outQ.notFull)  
    begin outQ.enq(f3(fifo2.first)); fifo2.deq end;  
endrule
```

- ◆ This rule is illegal if concurrent operations on FIFOs are not permitted

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-13

Limitations of registers

- ◆ Limitations of a language with only the register primitive
 - No communication between rules or between methods or between rules and methods in the same atomic action i.e. clock cycle
 - Can't express a FIFO with concurrent enq and deq

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-14

EHR: Ephemeral History Register

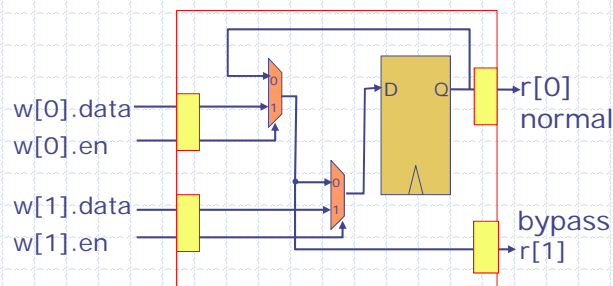
A new primitive element to design modules with concurrent methods

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-15

Ephemeral History Register (EHR) Dan Rosenband [MEMOCODE'04]



r[1] returns:

- the current state if w[0] is not enabled
 - the value being written (w[0].data) if w[0] is enabled
- w[i+1] takes precedence over w[i]

r[0] < w[0]

r[1] < w[1]

w[0] < w[1] < ...

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-16

Designing FIFOs using EHRs

- ◆ *Conflict-Free FIFO*: Both enq and deq are permitted concurrently as long as the FIFO is not-full **and** not-empty
 - The effect of enq is not visible to deq, and vice versa
- ◆ *Pipeline FIFO*: An enq into a full FIFO is permitted provided a deq from the FIFO is done simultaneously
- ◆ *Bypass FIFO*: A deq from an empty FIFO is permitted provided an enq into the FIFO is done simultaneously

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-17

One-Element *Pipelined FIFO*

```
module mkPipelineFifo(Fifo#(1, t)) provisos(Bits#(t, tSz));
  Reg#(t) d <- mkRegU;
  Ehr#(2, Bool) v <- mkEhr(False);

  method Bool notFull = !v[1];
  method Bool notEmpty = v[0];
  method Action enq(t x);
    d <= x;
    v[1] <= True;
  endmethod

  method Action deq;
    v[0] <= False;
  endmethod

  method t first;
    return d;
  endmethod
endmodule
```

Desired behavior

deq < enq
first < deq
first < enq

No double
write error

In any given cycle:

- If the FIFO is not empty then simultaneous enq and deq are permitted;
- Otherwise, only enq is permitted

September 19, 2016

<http://csg.csail.mit.edu/6.175>

L06-18

One-Element Bypass FIFO

```

module mkBypassFifo(Fifo#(1, t)) provisos(Bits#(t, tSz));
  Ehr#(2, t) d <- mkEhr(?);
  Ehr#(2, Bool) v <- mkEhr(False);

  method Bool notFull = !v[0];
  method Bool notEmpty = v[1];
  method Action enq(t x);
    d[0] <= x;
    v[0] <= True;
  endmethod

  method Action deq;
    v[1] <= False;
  endmethod

  method t first;
    return d[1];
  endmethod
endmodule

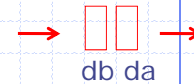
```

Desired behavior
 enq < deq
 first < deq
 enq < first

No double
 write error

In any given cycle:
 - If the FIFO is not full then
 simultaneous enq and deq
 are permitted;
 - Otherwise, only deq is
 permitted

Two-Element Conflict-free FIFO



```

module mkCFFifo(Fifo#(2, t)) provisos(Bits#(t, tSz));
  Ehr#(2, t) da <- mkEhr(?);
  Ehr#(2, Bool) va <- mkEhr(False);
  Ehr#(2, t) db <- mkEhr(?);
  Ehr#(2, Bool) vb <- mkEhr(False);

  rule canonicalize;
    if(vb[1] && !va[1])
      (da[1] <= db[1];
       va[1] <= True; vb[1] <= False) endrule

  method Bool notFull = !vb[0];
  method Bool notEmpty = va[0];
  method Action enq(t x);
    db[0] <= x; vb[0] <= True; endmethod
  method Action deq;
    va[0] <= False; endmethod
  method t first;
    return da[0]; endmethod
endmodule

```

Assume, if there is only
 one element in the FIFO
 it resides in da

Desired behavior
 enq CF deq
 first < deq
 first CF enq

In any given cycle:
 - Simultaneous enq
 and deq are
 permitted only if
 the FIFO is not full
 and not empty