Implementing Pipelines

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

- We will focus on some modules (FIFO, Register File, Scoreboard) needed to implement pipelines.
- Introduce a new type of register called EHR to express more concurrency and Bypasses in Bluespec.
Inelastic pipeline
control flow

This rule can fire only if
- inQ has an element
- outQ has space

Atomicity: Either all or none of the state elements inQ, outQ, sReg1 and sReg2 will be updated

```
rule sync-pipeline;
  inQ.deq;
  sReg1 <= f0(inQ.first);
  sReg2 <= f1(sReg1);
  outQ.enq(f2(sReg2));
endrule
```
Inelastic pipeline
starting and stopping

\[ \text{rule sync-pipeline;} \]
\[ \text{inQ.deq;} \]
\[ \text{sReg1} \leftarrow f0(\text{inQ.first}); \]
\[ \text{sReg2} \leftarrow f1(\text{sReg1}); \]
\[ \text{outQ.enq(f2(sReg2));} \]

\text{endrule}

- Red and Green tokens must move even if there is nothing in inQ!
- Also if there is no token in sReg2 then nothing should be enqueued in outQ

Modify the rule to deal with these conditions by introducing a valid bit for each pipeline register
Inelastic Pipeline with proper control

\[
\text{rule sync-pipeline;}
\]

\[
\text{if (inQnotEmpty)}
\]

\[
\text{begin sReg1 }\leftarrow f0(\text{inQ.first()}); \text{inQ.deq();}
\]

\[
\text{sReg1v }\leftarrow \text{Valid end}
\]

\[
\text{else sReg1v }\leftarrow \text{Invalid;}
\]

\[
\text{sReg2 }\leftarrow f1(\text{sReg1}); \text{sReg2v }\leftarrow \text{sReg1v;}
\]

\[
\text{if (sReg2v }\leftarrow \text{Valid) outQ.enq(f2(sReg2));}
\]

\[
\text{endrule}
\]

sReg1 and sReg2 with valid bits are beginning to look like one-element fifos!
These rules are easy to write but introduce a new concern:

- These rules can execute concurrently, only if fifos allow concurrent enq and deq
- In our one-element fifo design, enq and deq were mutually exclusive!
One-Element FIFO
from L11-L12

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

Can we make a fifo where enq and deq can be done concurrently?

ME = mutually exclusive
How about a Two-Element FIFO?

- Initially, both va and vb are false
- First enq will store the data in da and mark va true
- An enq can be done as long as vb is false;
- A deq can be done as long as va is true;
- Assume, if there is only one element in the FIFO, it resides in da
Two-Element FIFO

module mkCFFifo (Fifo#(2, t));
    // instantiate da, va, db, vb
    rule canonicalize if (vb && !va);
        da <= db;
        va <= True;
        vb <= False;
    endrule
    method Action enq(t x) if (!vb);
        begin db <= x; vb <= True; end
    endmethod
    method Action deq if (va);
        va <= False;
    endmethod
    method t first if (va);
        return da;
    endmethod
endmodule

Both enq and deq can execute concurrently but both are mutually exclusive with canonicalize. Canonicalize rule introduces a dead cycle after an enq/deq.

<table>
<thead>
<tr>
<th>enq</th>
<th>deq</th>
<th>first</th>
<th>cano</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>CF</td>
<td>CF</td>
<td>ME</td>
</tr>
<tr>
<td>CF</td>
<td>C</td>
<td>&gt;</td>
<td>ME</td>
</tr>
<tr>
<td>CF</td>
<td>&lt;</td>
<td>CF</td>
<td>ME</td>
</tr>
<tr>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>C</td>
</tr>
</tbody>
</table>
In Bluespec one thinks of bypassing in terms of reducing the number of cycles it takes to execute two conflicting rules or methods.

For example, design a FIFO, where a rule can perform an `enq` on a full FIFO provided another rule performs a `deq` simultaneously.

- Requires signaling from `deq` to `enq`.

Another example: Transform the rules on the right so that they execute concurrently, and `ra < rb`.

- Requires communicating the value of `x` from `ra` to `rb` in the same cycle.

Not possible in the subset of Bluespec you have seen so far!
Limitations of registers

- Using the register primitive no *communication* can take place in the same clock cycle between
  - two methods or
  - two rules or
  - a rule and a method

EHRs to the rescue ...

Ephemeral History Register (EHR): a primitive element to remedy this problem
Ephemeral History Register (EHR)
Dan Rosenband [MEMOCODE’04]

- $r[1]$ returns:
  - the current state if $w[0]$ is not enabled
  - the value being written if $w[0]$ is enabled
- $w[1]$ has higher priority than $w[0]$
## Conflict Matrix of Primitive modules
### Registers and EHRs

<table>
<thead>
<tr>
<th>Register</th>
<th>reg.r</th>
<th>reg.w</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg.r</td>
<td>CF</td>
<td>&lt;</td>
</tr>
<tr>
<td>reg.w</td>
<td>&gt;</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EHR</th>
<th>EHR.r0</th>
<th>EHR.w0</th>
<th>EHR.r1</th>
<th>EHR.w1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EHR.r0</td>
<td>CF</td>
<td>&lt;</td>
<td>CF</td>
<td>&lt;</td>
</tr>
<tr>
<td>EHR.w0</td>
<td>&gt;</td>
<td>C</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>EHR.r1</td>
<td>CF</td>
<td>&gt;</td>
<td>CF</td>
<td>&lt;</td>
</tr>
<tr>
<td>EHR.w1</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>C</td>
</tr>
</tbody>
</table>
Designing FIFOs using EHRs

- **Pipeline FIFO:** An enq into a full FIFO is permitted provided a deq from the FIFO is done simultaneously (deq < enq)

- **Bypass FIFO:** A deq from an empty FIFO is permitted provided an enq into the FIFO is done simultaneously (enq < deq)

- **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 vise versa

We will derive such FIFOs starting with one or two element FIFO implementations
Making One-Element FIFO into a **Pipeline FIFO**

```verilog
class mkFifo (fifo#1, t);
    reg#(t) d <- mkRegU;
    ehr#(2, bool) v <- mkEhr(False);
endclass

method action enq(t x) (!v[1]);
v[1] <= true; d <= x;
endmethod

method action deq if (v[0]);
v[0] <= false;
endmethod

method t first if (v[0]);
    return d;
endmethod
endmodule
```

### Pipelined FIFO CM

<table>
<thead>
<tr>
<th></th>
<th>enq</th>
<th>deq</th>
<th>first</th>
</tr>
</thead>
<tbody>
<tr>
<td>enq</td>
<td>C</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>deq</td>
<td>&lt;</td>
<td>C</td>
<td>&gt;</td>
</tr>
<tr>
<td>first</td>
<td>&lt;</td>
<td>&lt;</td>
<td>CF</td>
</tr>
</tbody>
</table>

- `enq ‘sees’ deq`  
- `v` has the right value in all cases  
- no double write error
Making One-Element FIFO into a Bypassed FIFO

module mkFifo (Fifo#(1, t));
    Ehr#(2, t) d <- mkEhr(?);
    Ehr#(2, Bool) v <- mkEhr(False);
method Action enq(t x) if (!v);
    v <= True; d[0] <= x;
endmethod
method Action deq if (v[1]);
    v[1] <= False;
endmethod
method t first if (v[1]);
    return d[1];
endmethod
endmodule

<table>
<thead>
<tr>
<th></th>
<th>enq</th>
<th>deq</th>
<th>first</th>
</tr>
</thead>
<tbody>
<tr>
<td>enq</td>
<td>C</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>deq</td>
<td>&gt;</td>
<td>C</td>
<td>&gt;</td>
</tr>
<tr>
<td>first</td>
<td>&gt;</td>
<td>&lt;</td>
<td>CF</td>
</tr>
</tbody>
</table>

- deq 'sees' enq
- v and d have the right values in all cases
- no double write error
module mkCFFifo (Fifo#(2, t));
Ehr#(2, t) da <- mkEhr(?);
Ehr#(2, Bool) va <- mkEhr(False);
Ehr#(2, t) db <- mkEhr(?);
Ehr#(2, Bool) vb <- mkEhr(False);
rule canonicalize (vb[1] && !va[1]);
  vb[1] <= False; endrule

method Action enq(t x) if (!vb[0]);
  db[0] <= x; vb[0] <= True;
endmethod

method Action deq if (va[0]);
  va[0] <= False;
endmethod

method t first if (va[0]);
  return da[0];
endmethod
endmodule

Two-Element FIFO

In any given cycle simultaneous enq and deq are permitted provided the FIFO is neither full nor empty

1. replace all registers by EHRs
2. since enq and deq happen first, assign them ports 0
3. assign canocalize port 1
Normal vs Bypass Register File

module mkRFile(RFile);
    Vector#(32,Reg#(Data)) rfile <- replicateM(mkReg(0));

    method Action wr(RIndx rindx, Data data);
        if(rindx!=0) rfile[rindx] <= data;
    endmethod
    method Data rd1(RIndx rindx) = rfile[rindx];
    method Data rd2(RIndx rindx) = rfile[rindx];
endmodule

{rd1, rd2} < wr

Can we design a bypass register file so that:
wr < {rd1, rd2}
Bypass Register File using EHR

module mkBypassRFile(RFile);
    Vector#(32,Ehr#(2, Data)) rfile <- replicateM(mkEhr(0));

    method Action wr(RIndx rindx, Data data);
        if(rindx!=0) (rfile[rindx])[0] <= data;
    endmethod

    method Data rd1(RIndx rindx) = (rfile[rindx])[1];
    method Data rd2(RIndx rindx) = (rfile[rindx])[1];
endmodule
Using EHRs

- EHRs can be used to design a variety of modules to reduce the conflict between its methods
  - FIFO, RF, Score Board, memory systems
- This way the user of such modules does not have to learn about EHRs unless he/she also wants to design modules with different concurrency properties
- However, modules that use EHRs, e.g., bypass FIFO or pipeline FIFO, can increase the delay of combinational paths and thus, affect the clock period
Material for take home problems and Recitation
module mkBypassRFile(BypassRFile);
    RFile rf <- mkRFile;
    SFifo#(1, RIdxData#(RIndx, Data))
        bypass <- mkBypassSFifo;
rule move;

method Action wr(RIndx rindx, Data data);

method Data rd1(RIndx rindx) =

endmodule

typedef struct {RIndx index; Data data} RIdxData deriving (Bits);

Sfifo = Searchable Fifo

Take Home:
Complete the design
Take home 2: When is this rule enabled?

rule sync-pipeline;
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

<table>
<thead>
<tr>
<th>inQ</th>
<th>sReg1v</th>
<th>sReg2v</th>
<th>outQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE</td>
<td>V</td>
<td>V</td>
<td>NF</td>
</tr>
<tr>
<td>NE</td>
<td>V</td>
<td>V</td>
<td>F</td>
</tr>
<tr>
<td>NE</td>
<td>V</td>
<td>I</td>
<td>NF</td>
</tr>
<tr>
<td>NE</td>
<td>V</td>
<td>I</td>
<td>F</td>
</tr>
<tr>
<td>NE</td>
<td>I</td>
<td>V</td>
<td>NF</td>
</tr>
<tr>
<td>NE</td>
<td>I</td>
<td>V</td>
<td>F</td>
</tr>
<tr>
<td>NE</td>
<td>I</td>
<td>I</td>
<td>NF</td>
</tr>
<tr>
<td>NE</td>
<td>I</td>
<td>I</td>
<td>F</td>
</tr>
</tbody>
</table>

Yes
Concurrency

Take home 3: Can any pipeline stages fire concurrently if the FIFOs do not permit concurrent enq and deq?