Module Interfaces and Concurrency

Arvind
Computer Science and Artificial Intelligence Laboratory
M.I.T.
Design Alternatives: Latency and Throughput

- **Combinational (C)**
- **Pipeline (P)**
- **Folded: Reuse a block (F)**

Suppose the latency of each:
- $f_i$ is 5ns
- FIFO is 1ns
- mux is 1ns
- register is 1ns

Latency: Total time to process 1 element

Throughput: Rate at which new elements can be processed

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ns)</td>
<td>$1+3\times5+1=17$</td>
<td>$1+3\times(5+1)=19$</td>
<td>$1+3\times(1+5+1)+1=23$</td>
</tr>
<tr>
<td>Throughput (1/ns)</td>
<td>$1/17$</td>
<td>$1/(5+1) = 1/6$</td>
<td>$1/23$</td>
</tr>
<tr>
<td>Clock period</td>
<td>16</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
A pipelined system

When can stage1 rule fire?
- inQ has an element
- fifo1 has space

Can these 3 rules execute concurrently?
- Yes, but it must be possible to do enq and deq in a fifo simultaneously

Can stage1 and stage3 execute concurrently?
- Yes, even if enq and deq cannot be done simultaneously in a fifo
Multirule Systems

- Most systems we have seen so far had multiple rules but only one rule was ready to execute at any given time (pair-wise mutually exclusive rules)

- Consider a system where multiple rules can be ready to execute at a given time
  - When can two such rules be executed together?
  - What does the synthesized hardware look like for concurrent execution of rules?
One-rule-at-a-time semantics of Bluespec

Repeatedly:
- Select any rule that is ready to execute
- Compute the state updates
- Make the state updates

Any legal behavior of a Bluespec program can be explained by observing the state updates obtained by applying one rule at a time

However, for performance we execute multiple rules concurrently whenever possible
Concurrent execution of rules

- Two rules can execute concurrently, if concurrent execution would not cause a double-write error, and
- The final state can be obtained by executing rules one-at-a-time in some sequential order
- Can these rules execute concurrently without violating the one-rule-at-a-time-semantics?

Example 1

```
rule ra;
  x <= x+1;
endrule
rule rb;
  y <= y+2;
endrule
```

Example 2

```
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule
```

Example 3

```
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= y+2;
endrule
```
Concurrent Execution

Example 1

\[
\begin{align*}
\text{rule } ra; \\
x^{t+1} &\leq x^t + 1; \\
\text{endrule} \\
\text{rule } rb; \\
y^{t+1} &\leq y^t + 2; \\
\text{endrule}
\end{align*}
\]

Final value of \((x,y)\) given the initial values \((0,0)\)

Concurrent Execution

Ex 1  (1,2)  Ex 2  (1,2)  Ex 3  (1,2)

\[\text{ra} < \text{rb}\]

\[\text{rb} < \text{ra}\]
## Executing ra before rb (ra < rb)

### Example 1

```plaintext
rule ra;
  x^{t+1} <= x^t + 1;
endrule
rule rb;
  y^{t+2} <= y^{t+1} + 2;
endrule
```

### Example 2

```plaintext
rule ra;
  x^{t+1} <= y^t + 1;
endrule
rule rb;
  y^{t+2} <= x^{t+1} + 2;
endrule
```

### Example 3

```plaintext
rule ra;
  x^{t+1} <= y^t + 1;
endrule
rule rb;
  y^{t+2} <= y^{t+1} + 2;
endrule
```

### Final value of \((x,y)\) given the initial values \((0,0)\)

<table>
<thead>
<tr>
<th>Concurrent Execution</th>
<th>Ex 1</th>
<th>Ex 2</th>
<th>Ex 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra &lt; rb</td>
<td>(1,2)</td>
<td>(1,2)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>rb &lt; ra</td>
<td>(1,2)</td>
<td>(1,3)</td>
<td>(1,2)</td>
</tr>
</tbody>
</table>

For any \(x\), if there is no \(x^{t+1}\) defined, then \(x^{t+1} = x^t\)
Executing rb before ra (rb < ra)

<table>
<thead>
<tr>
<th>Concurrent Execution</th>
<th>Ex 1</th>
<th>Ex 2</th>
<th>Ex 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra &lt; rb</td>
<td>(1,2)</td>
<td>(1,2)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>rb &lt; ra</td>
<td>(1,2)</td>
<td>(3,2)</td>
<td>(3,2)</td>
</tr>
</tbody>
</table>

Final value of (x,y) given the initial values (0,0)
Can these rules execute concurrently? (without violating the one-rule-at-a-time-semantics)

Example 1

\[
\text{rule ra; x <= x+1; endrule}
\]

\[
\text{rule rb; y <= y+2; endrule}
\]

Example 2

\[
\text{rule ra; x <= y+1; endrule}
\]

\[
\text{rule rb; y <= x+2; endrule}
\]

Example 3

\[
\text{rule ra; x <= y+1; endrule}
\]

\[
\text{rule rb; y <= y+2; endrule}
\]

Final value of \((x, y)\) given the initial values \((0, 0)\)

Concurrent Execution | Ex 1 | Ex 2 | Ex 3
--- | --- | --- | ---
ra < rb | \((1, 2)\) | \(\neq\) (1,3) | \(\neq\) (1,2)
rb < ra | \(\neq\) (1,2) | Conflicting-Free (CF) | (3,2) ra<rb

March 19, 2019
Conflict Matrix (CM)
BSV compiler generates the pairwise conflict information

Example 1

\begin{align*}
\text{rule } & ra; \\
& x \leq x+1; \\
\text{endrule} \\
\text{rule } & rb; \\
& y \leq y+2; \\
\text{endrule}
\end{align*}

\begin{array}{ccc}
|   | ra | rb | \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>CF</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
<td>C</td>
</tr>
</tbody>
</table>
\end{array}

Example 2

\begin{align*}
\text{rule } & ra; \\
& x \leq y+1; \\
\text{endrule} \\
\text{rule } & rb; \\
& y \leq x+2; \\
\text{endrule}
\end{align*}

\begin{array}{ccc}
|   | ra | rb | \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>
\end{array}

Example 3

\begin{align*}
\text{rule } & ra; \\
& x \leq y+1; \\
\text{endrule} \\
\text{rule } & rb; \\
& y \leq y+2; \\
\text{endrule}
\end{align*}

\begin{array}{ccc}
|   | ra | rb | \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>&lt;</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
<td>C</td>
</tr>
</tbody>
</table>
\end{array}

- C : rules can’t be executed concurrently
- CF: rules can be executed concurrently; the net effect is the same as if ra executed before rb (ra<rb) or (rb<ra)
- ra < rb : rules can be executed concurrently; the net effect is as if ra executed before rb
Using *conflict* (CM) information in hardware synthesis
Suppose we connect \texttt{ra.rdy} to \texttt{ra.en}, and \texttt{rb.rdy} to \texttt{rb.en}
This circuit will execute rules \texttt{ra} and \texttt{rb} concurrently
This circuit is correct only if rules \texttt{ra} and \texttt{rb} do not conflict
But in this example rules \texttt{ra} and \texttt{rb} do conflict!
Need for a scheduler

Example 2

```
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule
```

- Guards (rdy signals) of all rules are fed to a scheduler
- Using the CM, the scheduler lets only non-conflicting rules proceed
  - Scheduler is a pure combinational circuit with a small number of gates
  - A correct but low performance scheduler may schedule only one rule at a time
Example schedulers

Example1

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
</tr>
</tbody>
</table>

Example2

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
</tr>
</tbody>
</table>

Example3

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

The effect ra < rb would be achieved automatically.
The rule scheduler

- Guards \((r_1.rdy \ldots r_n.rdy)\) of many rules may be true simultaneously, and some of them may conflict.
- BSV compiler constructs a combinational scheduler circuit with the following property:

```
for all pairs of rules \(i\) and \(j\), if \(r^i.en\) and \(r^j.en\) are true then the corresponding \(r^i.rdy\) and \(r^j.rdy\) must be true and rules \(i\) and \(j\) must not conflict with each other.
```
Takeaway

- One-rule-at-a-time semantics are important to understand the legal behaviors of a system.
- Efficient hardware for multi-rule system requires that many rules execute in parallel without violating the one-rule-at-time semantics.
- BSV compiler builds a scheduler circuit to execute as many rules as possible concurrently.
- For high-performance designs we have to worry about the CM characteristics of our modules.

More on this topic later in the course.
We are not done yet!

These rules can execute concurrently, only if fifos allow concurrent enq and deq
In our one-element fifo design, enq and deq are mutually exclusive!
We will design better FIFO’s later

<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>ME</td>
<td>ME</td>
</tr>
<tr>
<td>deq</td>
<td>ME</td>
<td>C</td>
<td>&gt;</td>
</tr>
<tr>
<td>first</td>
<td>ME</td>
<td>&lt;</td>
<td>CF</td>
</tr>
</tbody>
</table>

No pipelining
Take-home problem

- Draw a hardware circuit for this design, ignoring the internals of the fifo design

**Hint**
- Draw the guard for each rule
- Assume a enable signal for each rule
- Connect the rule ready and enable signals to a scheduler

```
rule stage1;
  fifo.enq(f0(inQ.first));
  inQ.deq;
endrule
rule stage2;
  outQ.enq(f1(fifo.first));
  fifo.deq;
endrule
```