State Machine Timing

- **Retiming**
  - Slash logic between registers to balance latencies and improve clock timings
  - Accelerate or retard cycle in which outputs are asserted
- **Pipelining**
  - Splitting computations into overlapped, smaller time steps

Recall: Synchronous Mealy Machine Discussion

- Placement of flipflops before and after the output logic changes the timing of when the output signals are asserted...

Recall: Synchronous Mealy Machine with Synchronizers Following Outputs

Case III: Synchronized Outputs

<table>
<thead>
<tr>
<th>CLK</th>
<th>Cycle 0</th>
<th>Cycle 1</th>
<th>Cycle 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f'</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A asserted during Cycle 0, f' asserted in next cycle
Effect of f delayed one cycle

Vending Machine State Machine

- **Moore machine** outputs associated with state
- **Mealy machine** outputs associated with transitions

State Machine Retiming

- **Moore (Async) Mealy Machine**
  - Vending Machine Example
  - Open asserted only when in state 15
  - Open asserted when last coin inserted leading to state 15

State Machine Retiming

- Retiming the Moore Machine: Faster generation of outputs
- Synchronizing the Mealy Machine: Add a FF, delaying the output
- These two implementations have identical timing behavior
  - Push the AND gate through the State FFs and synchronize with an output FF
  - Like computing open in the prior state and delaying it one state time
State Machine Retiming

- Effect on timing of Open Signal (Moore Case)

NOTE: overlaps with Next State calculation

State Machine Retiming

- Timing behavior is the same, but are the implementations really identical?

Pipelining Principle

- Pipelining review from CS61C

Analog to washing clothes:

step 1: wash (20 minutes)
step 2: dry (20 minutes)
step 3: fold (20 minutes)

60 minutes × 4 loads → 4 hours

<table>
<thead>
<tr>
<th>wash</th>
<th>load1</th>
<th>load2</th>
<th>load3</th>
<th>load4</th>
</tr>
</thead>
<tbody>
<tr>
<td>dry</td>
<td>load1</td>
<td>load2</td>
<td>load3</td>
<td>load4</td>
</tr>
<tr>
<td>fold</td>
<td>load1</td>
<td>load2</td>
<td>load3</td>
<td>load4</td>
</tr>
</tbody>
</table>

20 min

overlapped → 2 hours

Pipelining

- General principle:

\[ T = 8 \text{ ns} \]

\[ T_{\text{setup}} + \text{clk} \rightarrow q = 1 \text{ ns} \]

\[ F = 1/9 \text{ ns} = 111 \text{ MHz} \]

- Cut the Cl block into pieces (stages) and separate with registers:

\[ T = 4 \text{ ns} = 1 \text{ ns} = 3 \text{ ns} = 10 \text{ ns} \]

\[ F = 1/(4 \text{ ns} - 1 \text{ ns}) = 200 \text{ MHz} \]

- Cl block produces a new result every 5 ns instead of every 9 ns

Pipelining

- Wash: load1, load2, load3, load4
- Fold: load1, load2, load3, load4

- Increase number of loads, average time per load approaches 20 minutes
- Latency (time from start to end) for one load = 60 min
- Throughput = 3 loads/hour
- Pipelined throughput = # of pipe stages x un-pipelined throughput.

Limits on Pipelining

- Without FF overhead, throughput improvement proportional to # of stages
- After many stages are added, FF overhead begins to dominate
- FF overhead is the setup and clk to Q times

\[ \frac{\text{throughput}}{\text{ideal}} = \frac{\text{half the clock period in FF overhead}}{\text{FF overhead}} \]

- Other limits to effective pipelining:
  - Clock skew contributes to clock overhead
  - Unequal stages
  - FFs dominate cost
  - Clock distribution power consumption
  - Feedback (depends on loop iteration length)
### Pipelining Example

- \( F(x) = y = ax^2 + bx + c \)
- Computation graph:
  - \( x \) and \( y \) are assumed to be "streams"
  - Divide into 3 (nearly) equal stages.
  - Insert pipeline registers at dashed lines.
  - Can we pipeline basic operators?

### Example: Pipelined Adder

- Possible, but usually not done...
  - (arithmetic units can often be made sufficiently fast without internal pipelining)

### State Machine Retiming Summary

- **Retiming**
  - **Vending Machine Example**
    - Very simple output function in this particular case
    - But if output takes a long time to compute vs. the next state computation time --- can use retiming to "balance" these calculations and reduce the cycle time
  - **Pipelining**
    - Introduce registers to split computation to reduce cycle time and allow parallel computation
    - Trade latency (number of stage delays) for cycle time reduction