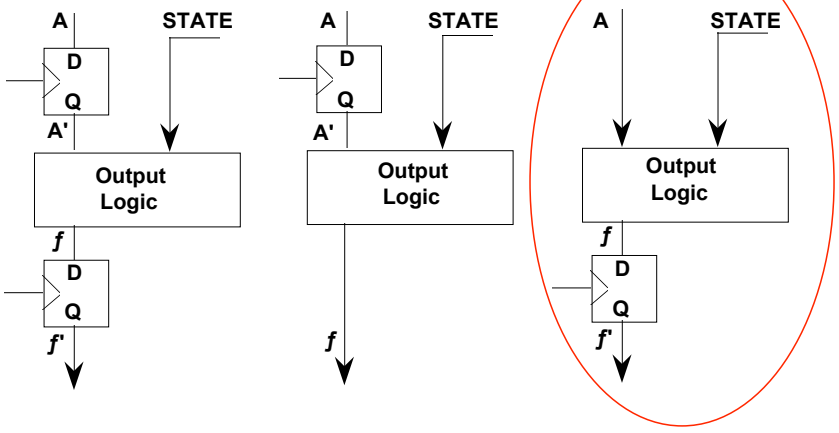
State Machine Timing

- Retiming
  - Slosh logic between registers to balance latencies and improve clock timings
  - Accelerate or retard cycle in which outputs are asserted
- Parallelism
  - Doing more than one thing at a time
- Pipelining
  - Splitting computations into overlapped, smaller time steps

Recall: Synchronous Mealy Machine Discussion

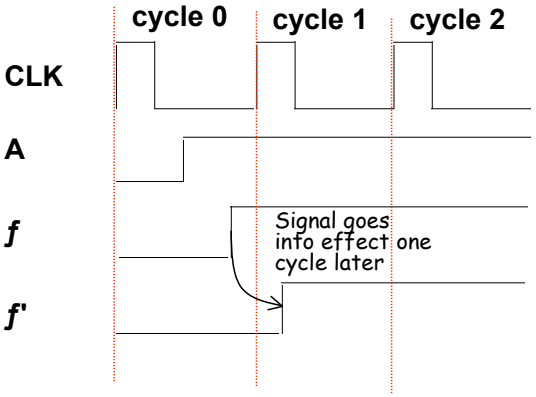
- Placement of flipflops before and after the output logic changes the timing of when the output signals are asserted …

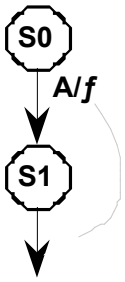
Synchronizer Circuitry at Inputs and Outputs



Recall: Synchronous Mealy Machine with Synchronizers Following Outputs

Case III: Synchronized Outputs





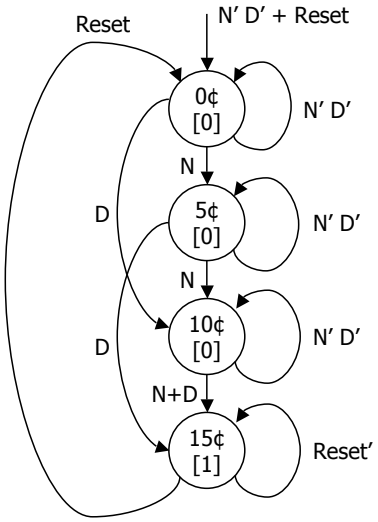
A asserted during Cycle 0, f' asserted in next cycle

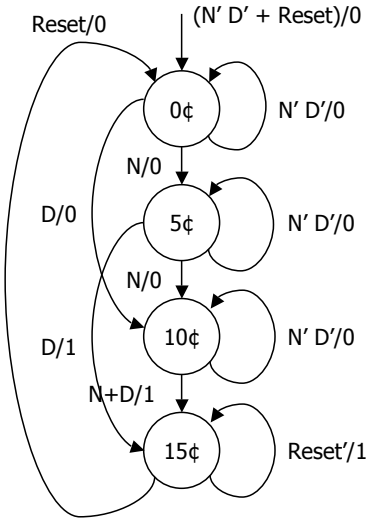
Effect of f delayed one cycle

Vending Machine State Machine

- Moore machine
  - outputs associated with state

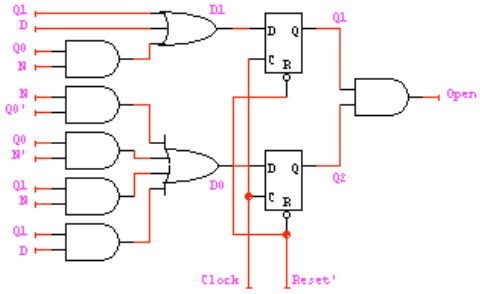
Mealy machine
outputs associated with transitions



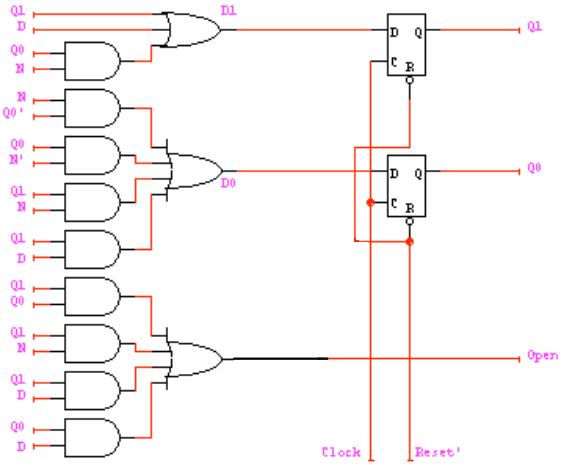


State Machine Retiming

- Moore vs. (Async) Mealy Machine
  - Vending Machine Example



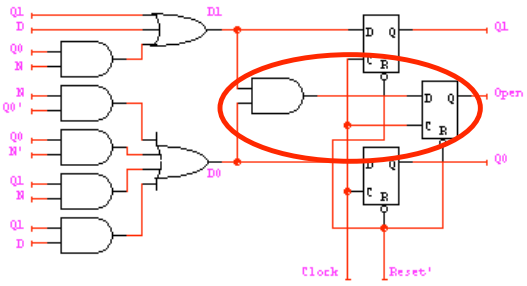
Open asserted only when in state 15



Open asserted when last coin inserted leading to state 15

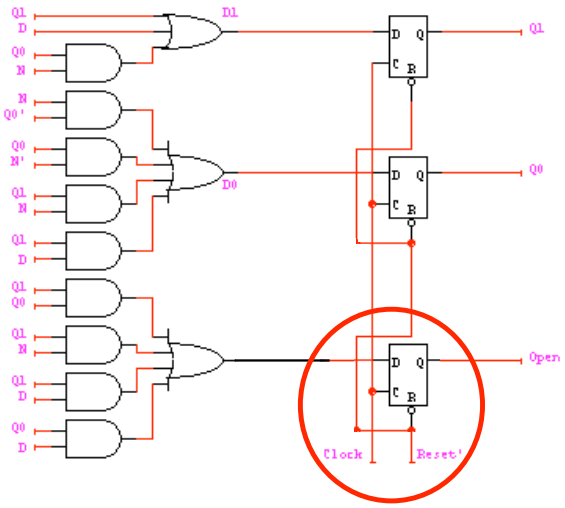
State Machine Retiming

- 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



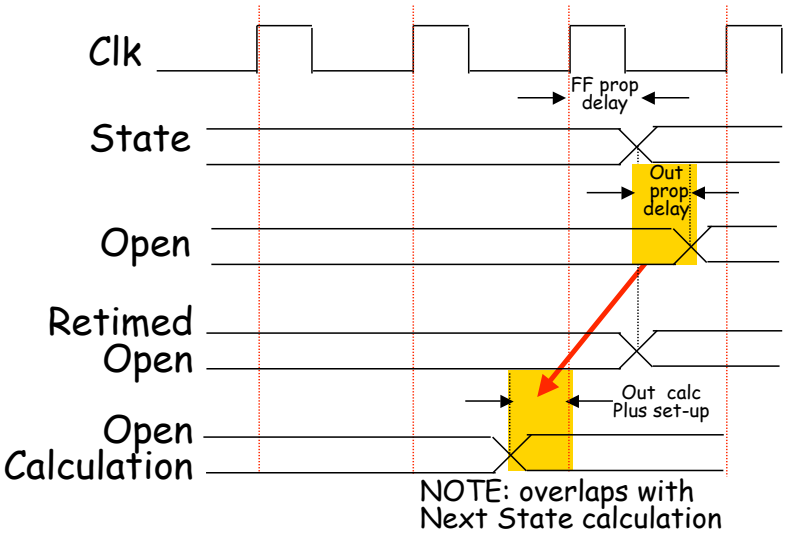
Push the AND gate through the State FFs and synchronize with an output FF

Like computing open in the prior state and delaying it one state time



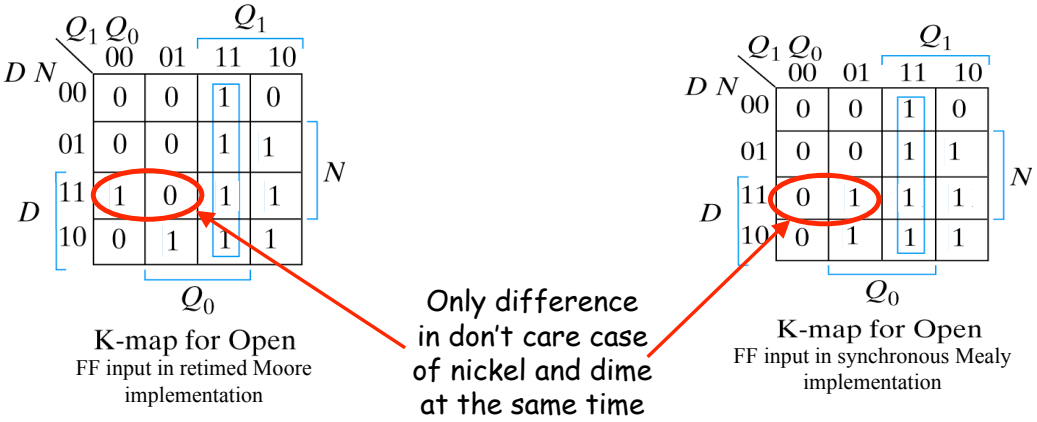
State Machine Retiming

- Effect on timing of Open Signal (Moore Case)



State Machine Retiming

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



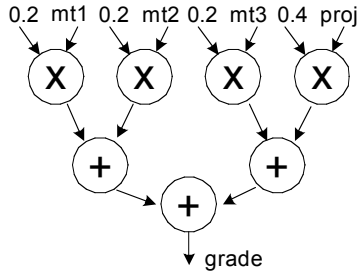
Parallelism

Doing more than one thing at a time: optimization in h/w often involves using parallelism to trade between cost and performance

- Example, Student final grade calculation:

```
read mt1, mt2, mt3, project;
grade = 0.2 x mt1 + 0.2 x mt2
              + 0.2 x mt3 + 0.4 x project;
write grade;
```

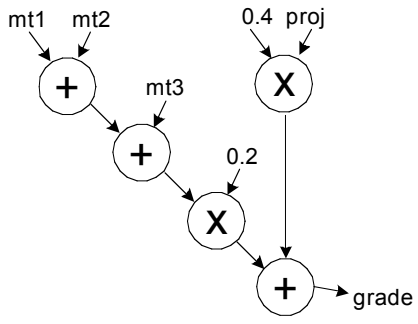
- High performance hardware implementation:



As many operations as possible are done in parallel

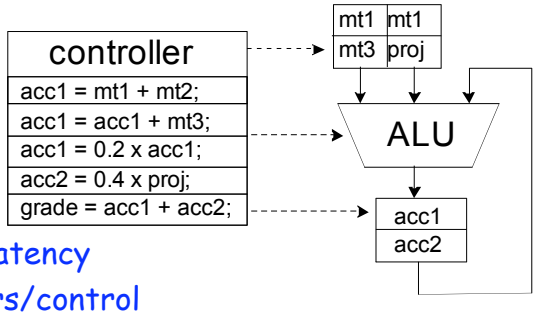
Parallelism

- Is there a lower cost hardware implementation? Different tree organization?
- Can factor out multiply by 0.2:

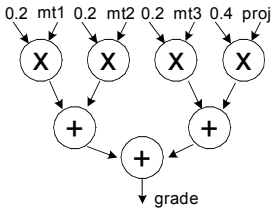


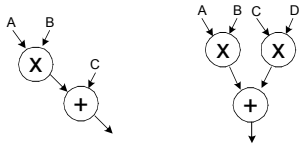
- How about sharing operators (multipliers and adders)?

Time Multiplexing



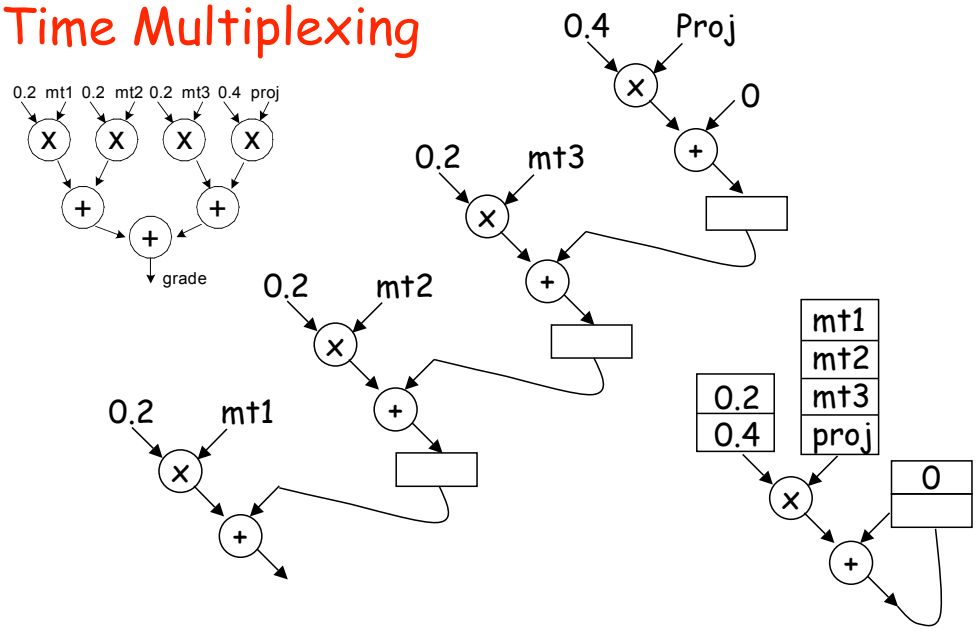
- Reuse single ALU for all adds and multiplies
  - Lower hardware cost, longer latency
  - BUT must add muxes/registers/control
- Consider the combinational hardware circuit diagram as an *abstract computation-graph*:





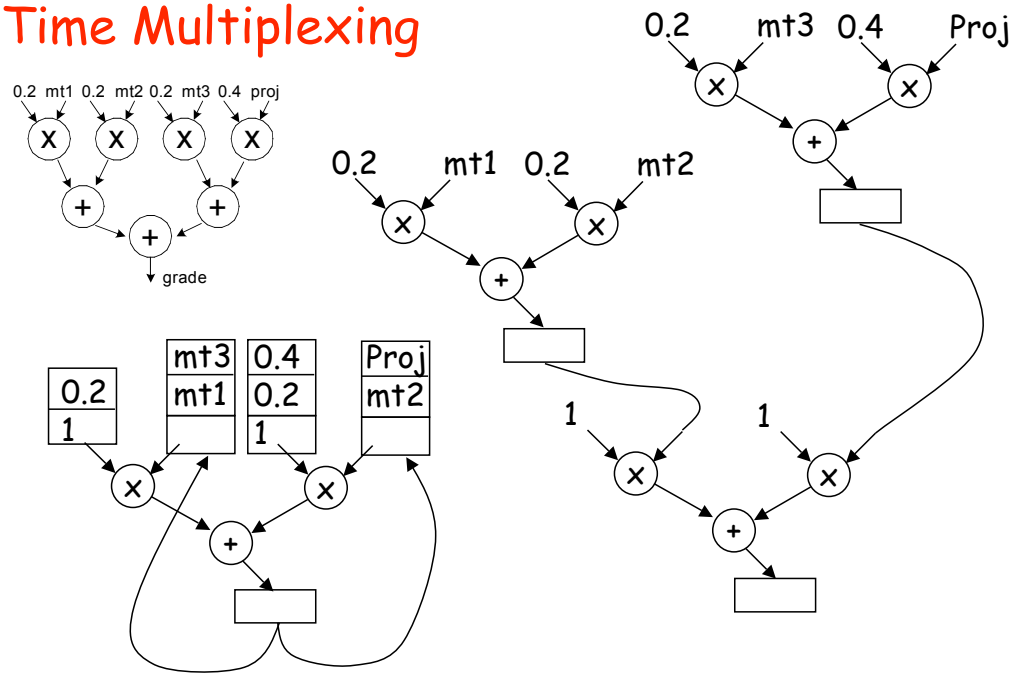
Alternative building blocks

Time Multiplexing



Time-multiplexing "covers" the computation graph by performing the action of each node one at a time

Time Multiplexing



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 x 4 loads ⇒ 4 hours

```
wash  | load1 | load2 | load3 | load4 |       |
dry   |       | load1 | load2 | load3 | load4 |
fold  |       |       | load1 | load2 | load3 | load4 |
      | 20 min|       |       |       |       |       |
```

overlapped ⇒ 2 hours

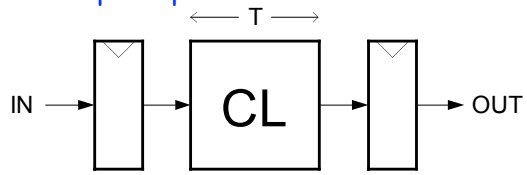
Pipelining

wash	load1	load2	load3	load4		
dry		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

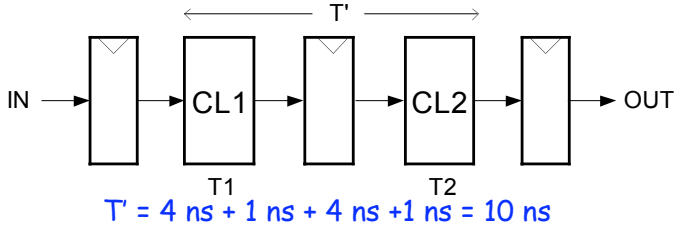
Pipelining

- General principle:



Assume T = 8 ns
T_{FF}(setup +clk→q) = 1 ns
F = 1/9 ns = 111 MHz

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



Assume T1 = T2 = 4 ns

T' = 4 ns + 1 ns + 4 ns +1 ns = 10 ns

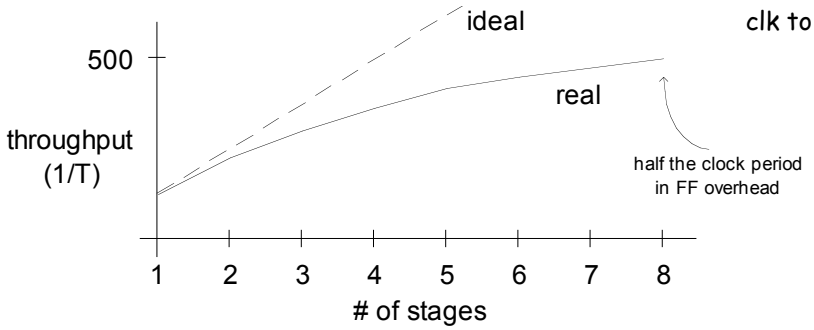
F = 1/(4 ns +1 ns) = 200 MHz

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

Limits on Pipelining

- Without FF overhead, throughput improvement proportional to # of stages
  - After many stages are added. FF overhead begins to dominate:

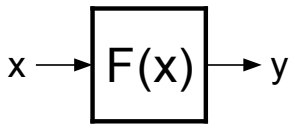
FF "overhead" is the setup and clk to Q times.



- Other limiters to effective pipelining:
  - Clock skew contributes to clock overhead
  - Unequal stages
  - FFs dominate *cost*
  - Clock distribution power consumption
  - *feedback (dependencies between loop iterations)*

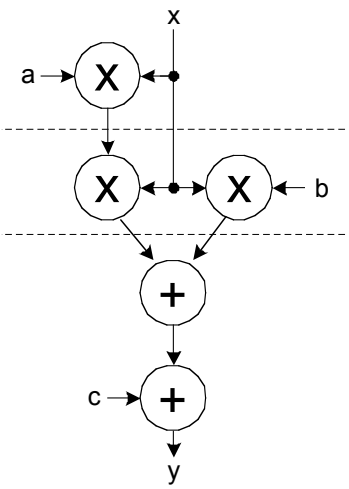
Pipelining Example

- $F(x) = y_i = a\,x_i^2 + b\,x_i + c$



- 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?

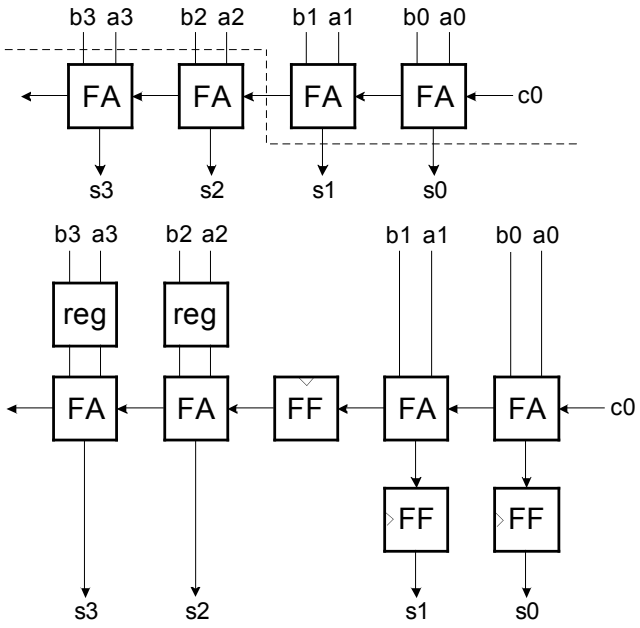
- Computation graph:



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
- Parallelism
  - Tradeoffs in cost and performance
  - Time reuse of hardware to reduce cost but sacrifice performance
- Pipelining
  - Introduce registers to split computation to reduce cycle time and allow parallel computation
  - Trade latency (number of stage delays) for cycle time reduction

Announcements

- Midterm II -- NEXT Thursday, 22 March in CS 150 Laboratory, 2:10 - 3:30+ (*that is one week from today -- tell your friends!)*
  - Closed book, open double sided crib sheet
  - TA review session next week
  - Five or so design-oriented questions covering:
    - State machine word problems
    - Memory systems
    - Datapath design
    - Register transfer
    - Controller implementation
      - Time state
      - Jump Counter
      - Branch Sequencer
      - Horizontal and Vertical Microprogramming
    - Retiming, Parallelism, Pipelining
    - Labs 4, 5, Checkpoint 0, 1