ECE 327/627
Digital Hardware Systems

Lecture 8: Pipelining II

Andrew Boutros

andrew.boutros@uwaterloo.ca

Some Logistics

- Lab 2 deadline is this Friday
  - Worth **8%** of the ECE 327 course grade
  - Worth **4%** of the ECE 627 course grade
  - Deadline on **Friday, Feb 6 @ 11:59 pm**

In the Previous Lecture …

- What is pipelining?
- Pipelining terminology:
  - Throughput (ops/sec) & Latency (sec)
  - Critical Path Delay & Clock Period
- The effect of pipelining …
  - Critical path delay (and therefore clock period) → decreases
  - Frequency → increases
  - Throughput → increases
  - Latency → depends (more cycles but each cycle is shorter)
- Pipelining example using polynomial circuit (code & tradeoffs)
- Wire delays becoming dominant in modern processes
  - We can pipeline wires that traverse a long distance on the chip
- Handling bubbles in pipelined circuits → in/out valid tags

Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$

Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$

- t is time → $t-1$ is 1 cycle before t, $t-2$ is 2 cycles before t

Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$

- t is time → $t-1$ is 1 cycle before t, $t-2$ is 2 cycles before t

- The values of x are streamed in cycle-by-cycle

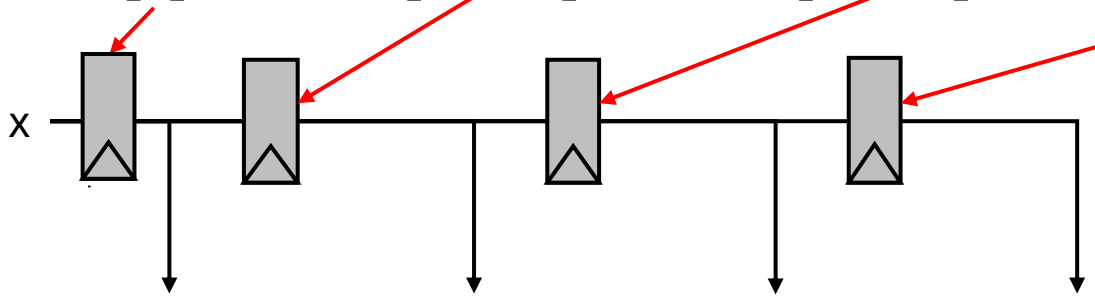
Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$
- t is time → $t-1$ is 1 cycle before t, $t-2$ is 2 cycles before t
- The values of x are streamed in cycle-by-cycle
- The w values are constant filter coefficients → they do not change during computation

Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

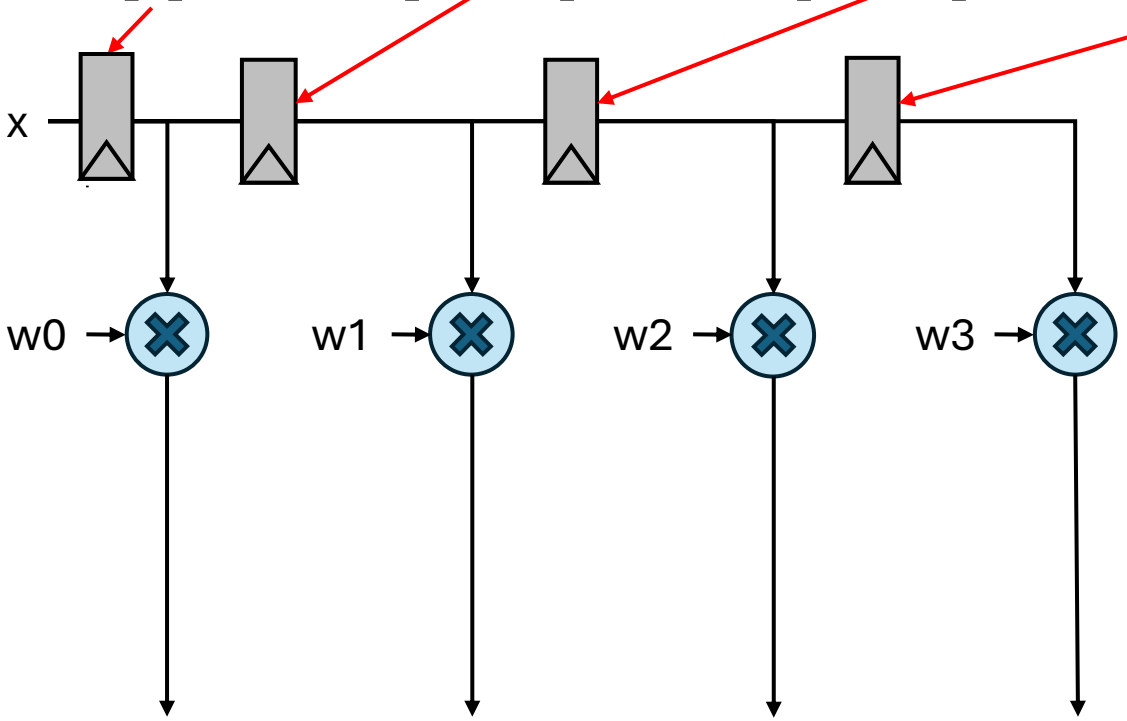
$$y[t] = x[t]\ w0 + x[t-1]\ w1 + x[t-2]\ w2 + x[t-3]\ w3$$



Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

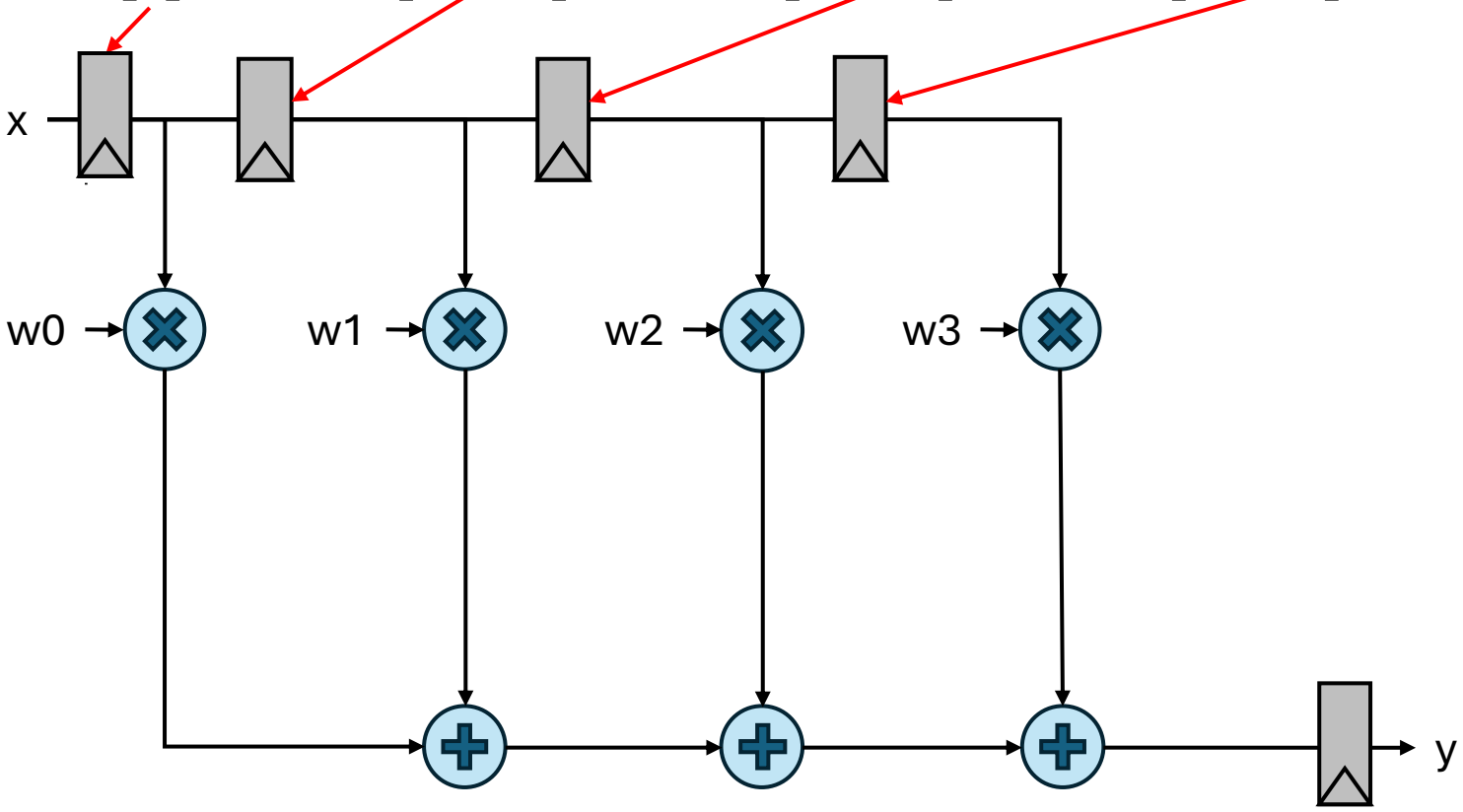
$$y[t] = x[t]\,w0 + x[t-1]\,w1 + x[t-2]\,w2 + x[t-3]\,w3$$



Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

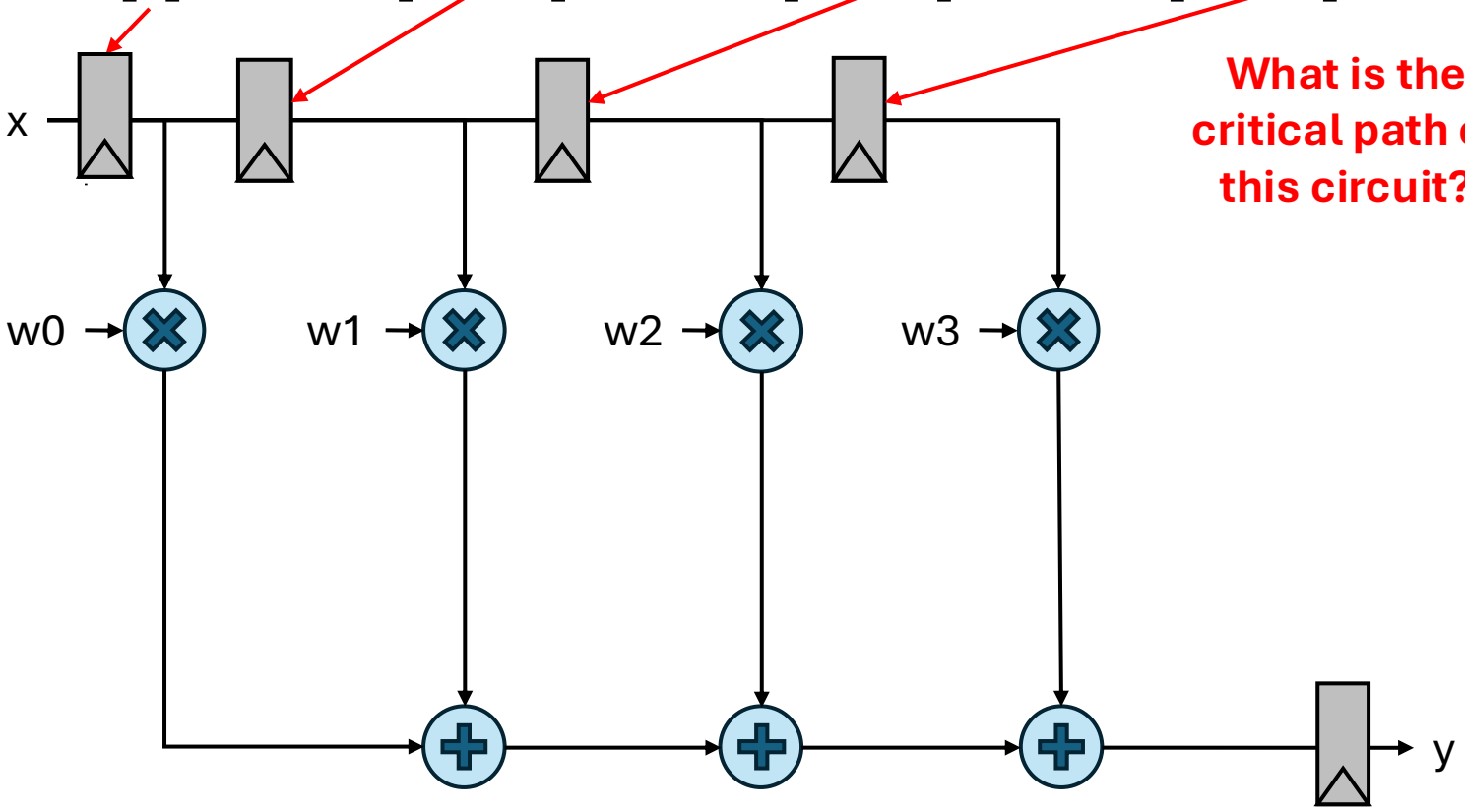
$$y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$$



Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

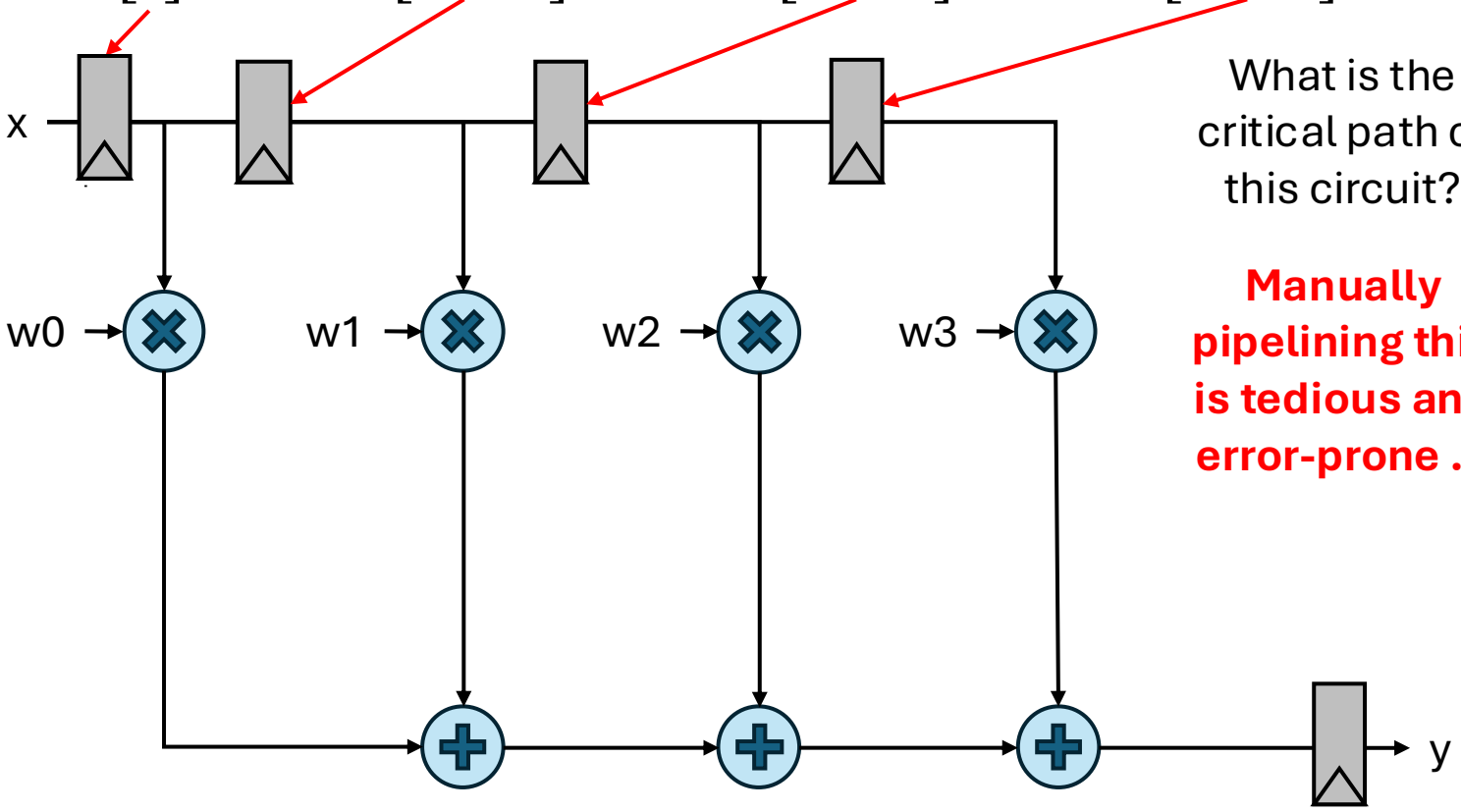
$$y[t] = x[t]\,w0 + x[t-1]\,w1 + x[t-2]\,w2 + x[t-3]\,w3$$



Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

$$y[t] = x[t]\,w0 + x[t-1]\,w1 + x[t-2]\,w2 + x[t-3]\,w3$$



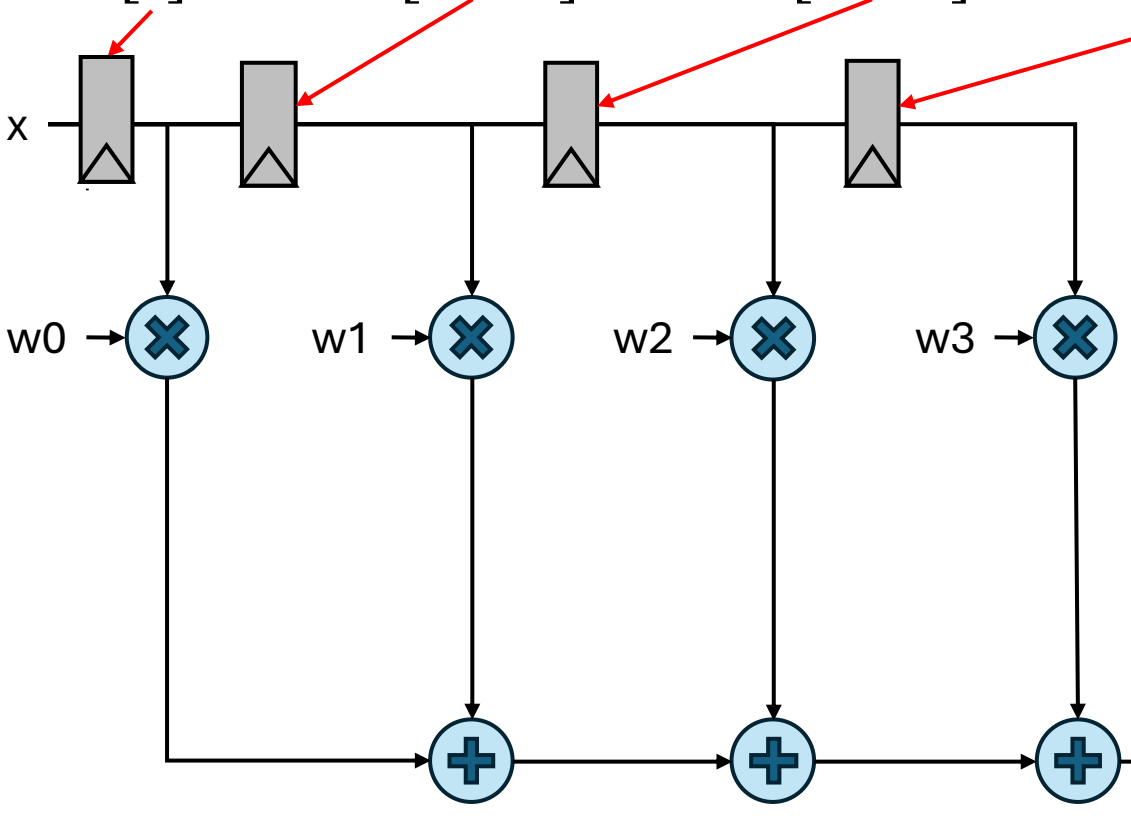
What is the critical path of this circuit?

Manually pipelining this is tedious and error-prone …

Example: Finite Impulse Response (FIR) Filter

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

$$y[t] = x[t]\,w0 + x[t-1]\,w1 + x[t-2]\,w2 + x[t-3]\,w3$$



What is the critical path of this circuit?

Manually pipelining this is tedious and error-prone …

Some simple rules can help!

Retiming

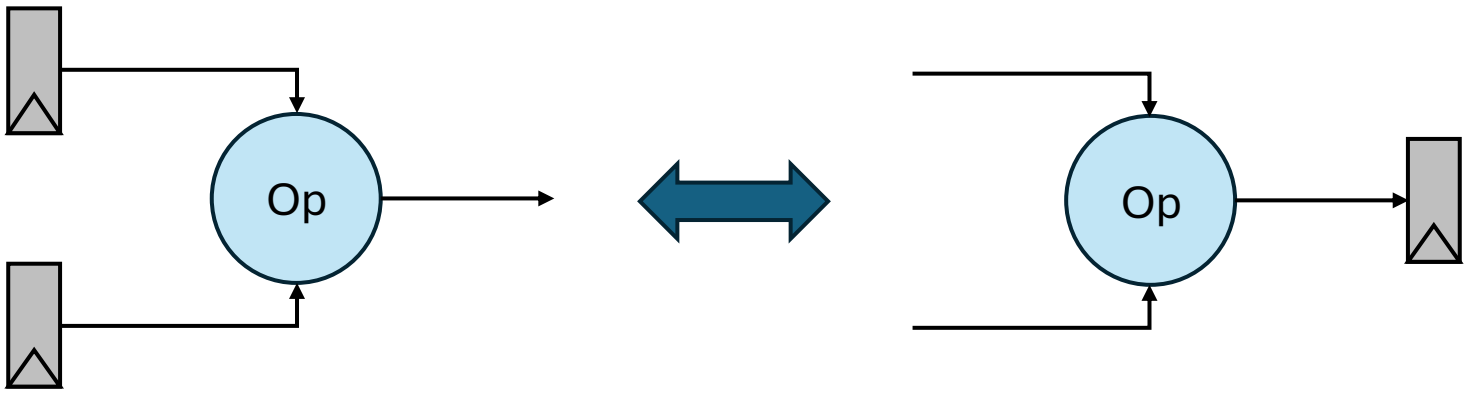
- Retiming is the process of moving registers around with the goal of optimizing clock frequency and/or number of pipeline registers such that:
  - When monitoring the inputs and outputs of the circuit, the circuit behavior is preserved

Retiming

- Retiming is the process of moving registers around with the goal of optimizing clock frequency and/or number of pipeline registers such that:
  - When monitoring the inputs and outputs of the circuit, the circuit behavior is preserved

- Some simple rules:

  1. Registers on the inputs of an operation can be moved to its outputs and vice-versa

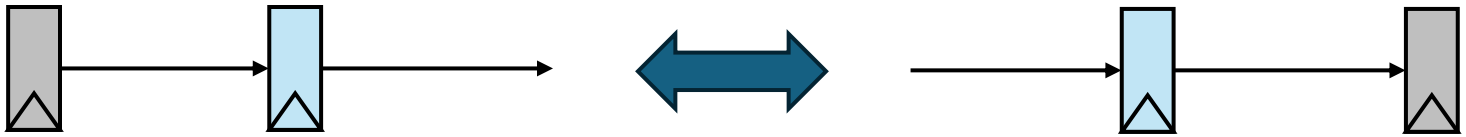


Retiming

- Retiming is the process of moving registers around with the goal of optimizing clock frequency and/or number of pipeline registers such that:
  - When monitoring the inputs and outputs of the circuit, the circuit behavior is preserved

- Some simple rules:

  1. Registers on the inputs of an operation can be moved to its outputs and vice-versa (an operation itself can be a register!)

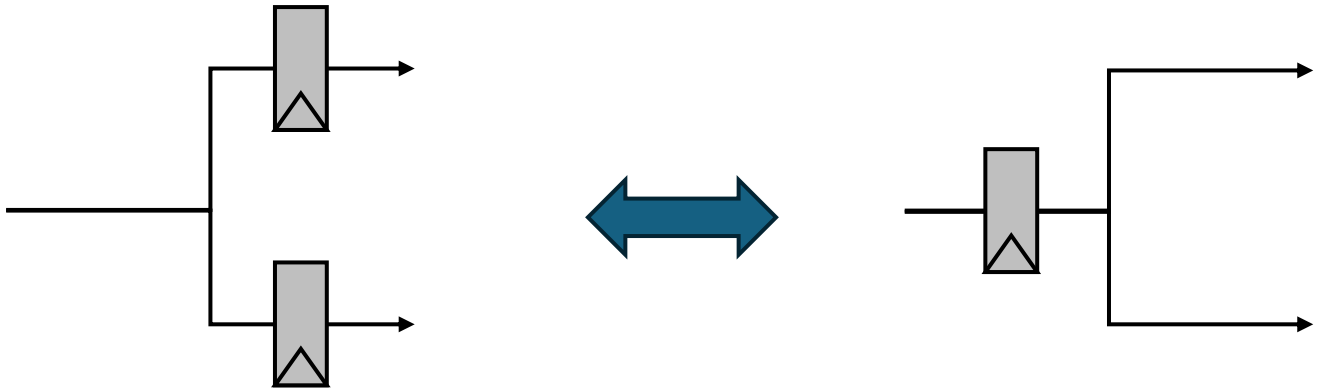


Retiming

- Retiming is the process of moving registers around with the goal of optimizing clock frequency and/or number of pipeline registers such that:
  - When monitoring the inputs and outputs of the circuit, the circuit behavior is preserved

- Some simple rules:

  1. Registers on the inputs of an operation can be moved to its outputs and vice-versa (an operation itself can be a register! Or fanout!)

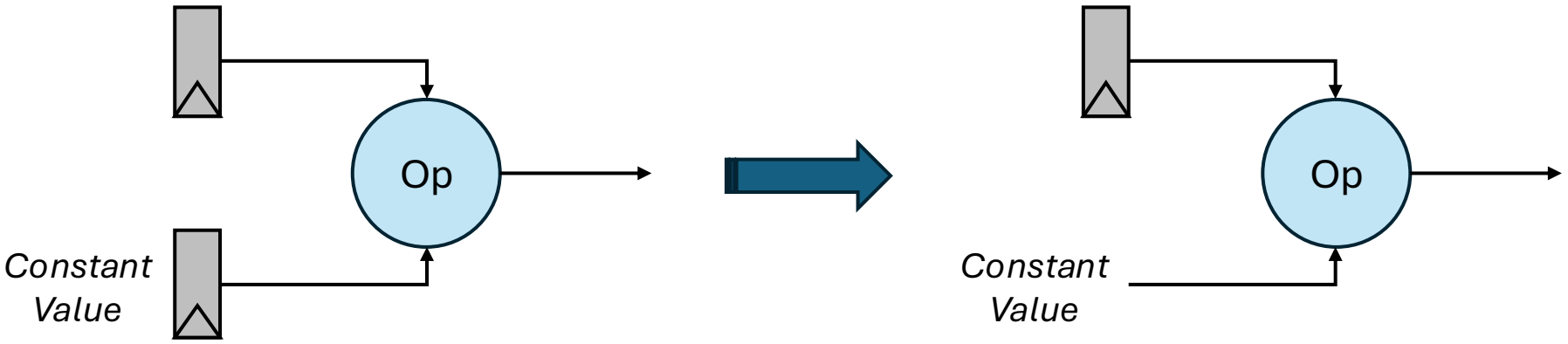


Retiming

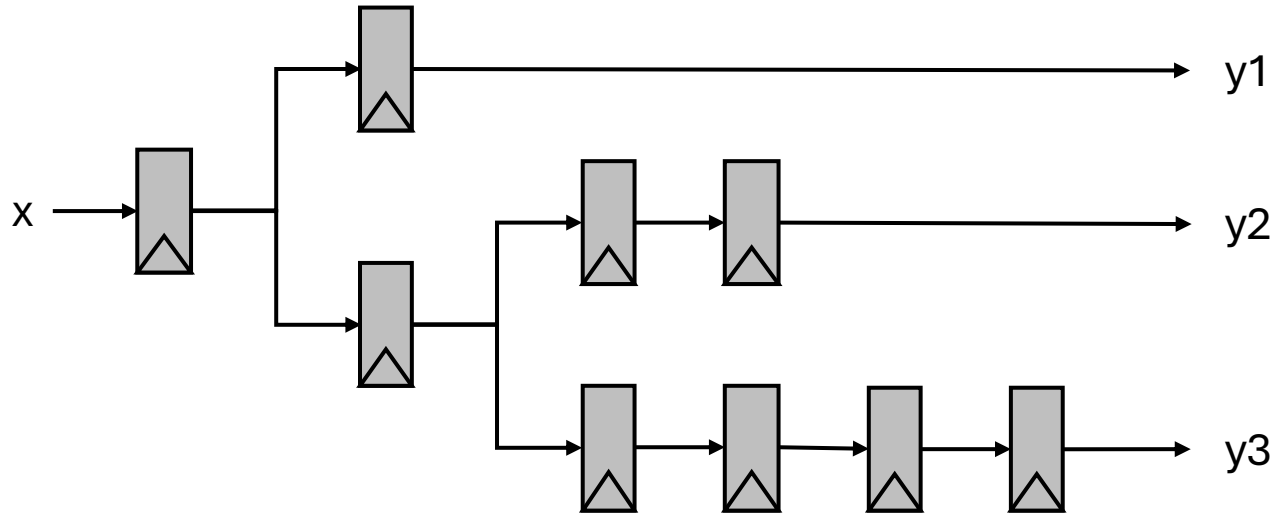
- Retiming is the process of moving registers around with the goal of optimizing clock frequency and/or number of pipeline registers such that:
  - When monitoring the inputs and outputs of the circuit, the circuit behavior is preserved

- Some simple rules:

  2. If an input is a constant, that input doesn't need to be registered after a retiming move

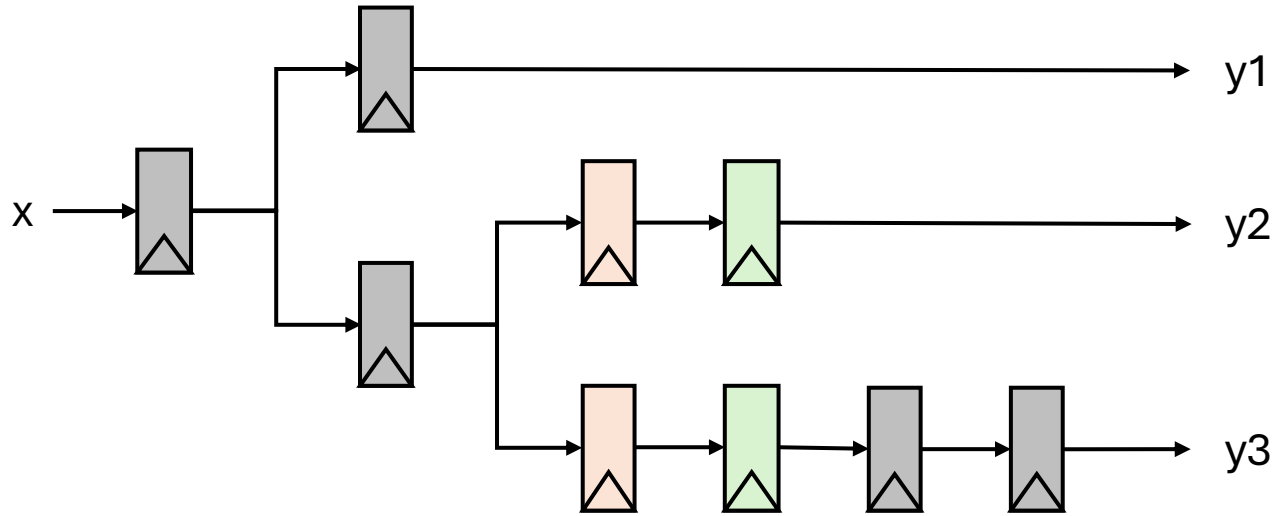


Register Fanout Example



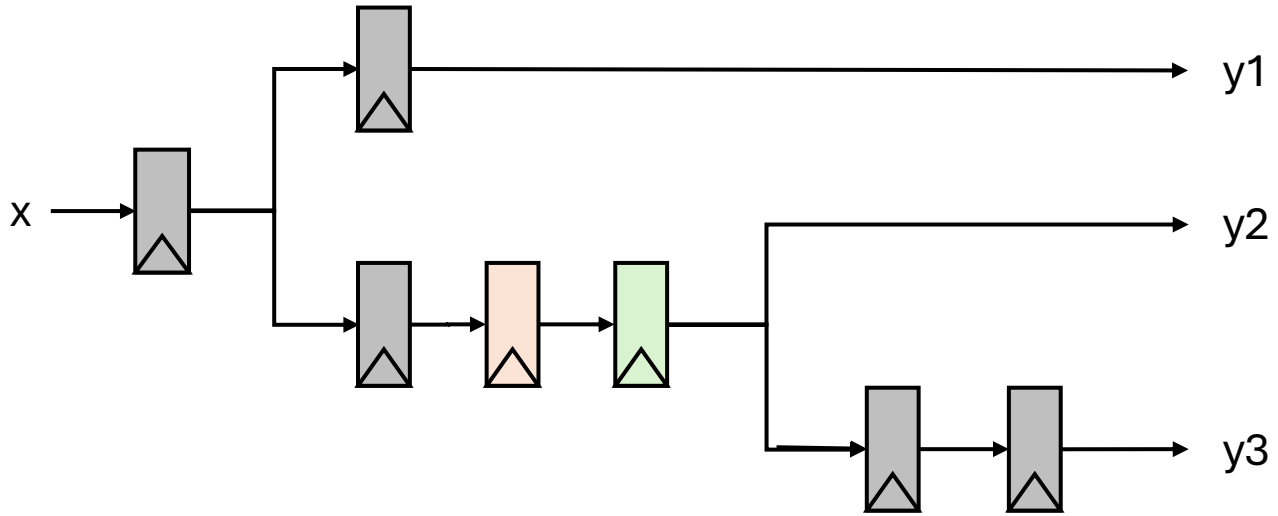
- Latency from x to y1 = 2
- Latency from x to y2 = 4
- Latency from x to y3 = 6
- No. of registers = 9

Register Fanout Example



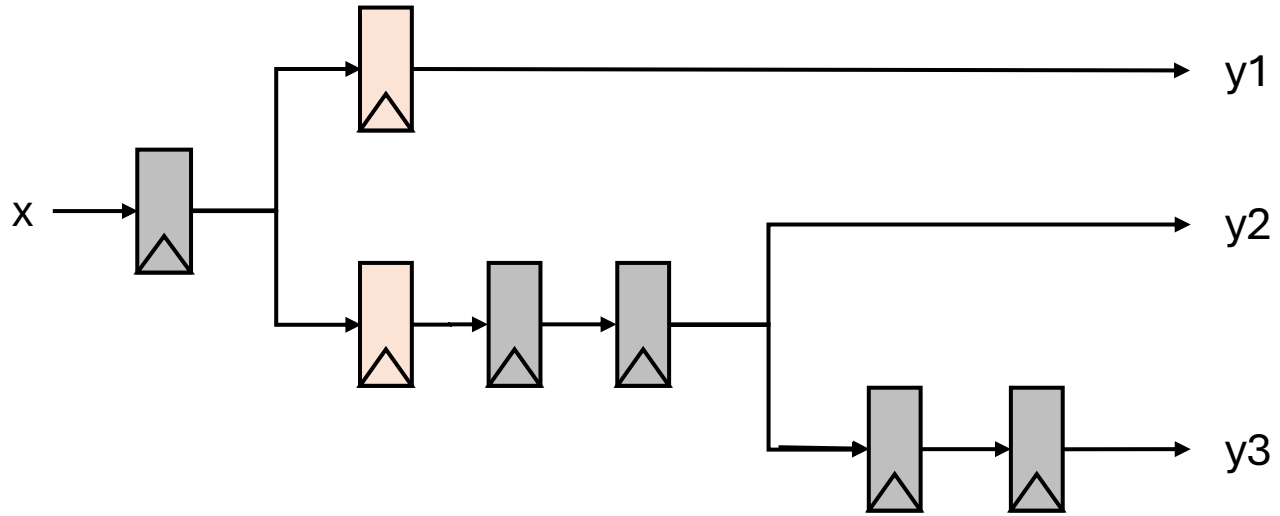
- Latency from x to y1 = 2
- Latency from x to y2 = 4
- Latency from x to y3 = 6
- No. of registers = 9

Register Fanout Example



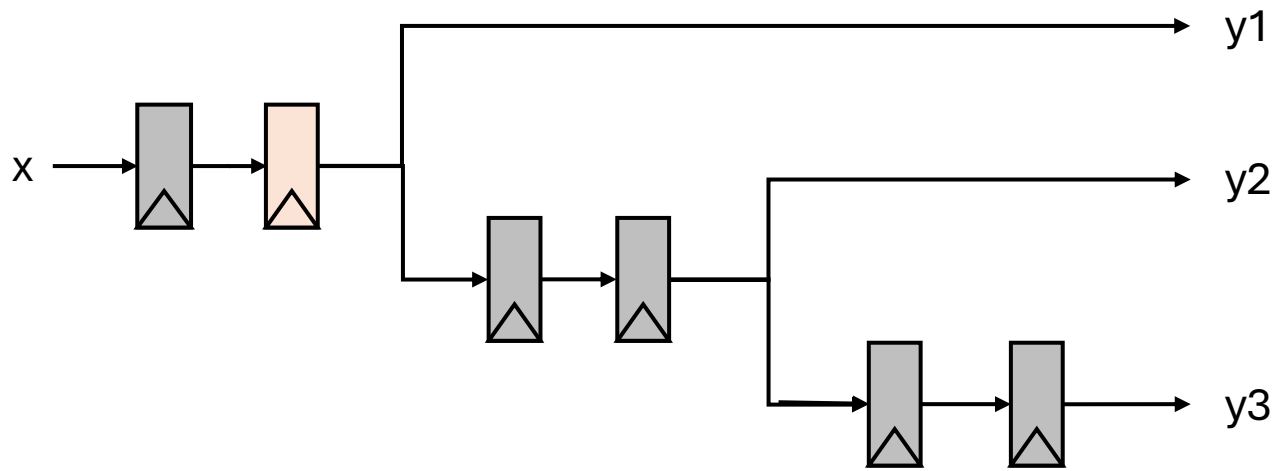
- Latency from x to y1 = 2
- Latency from x to y2 = 4
- Latency from x to y3 = 6
- No. of registers = **7**

Register Fanout Example



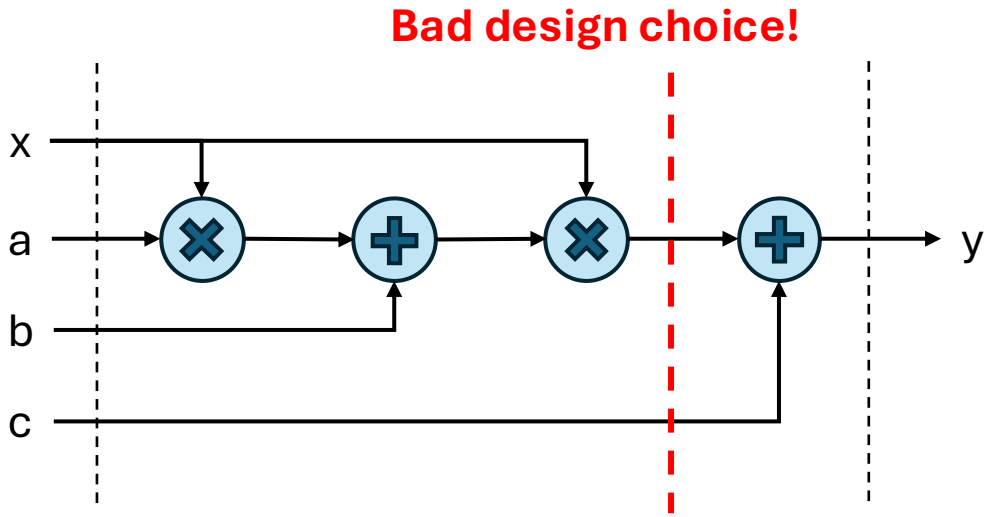
- Latency from x to y1 = 2
- Latency from x to y2 = 4
- Latency from x to y3 = 6
- No. of registers = 7

Register Fanout Example



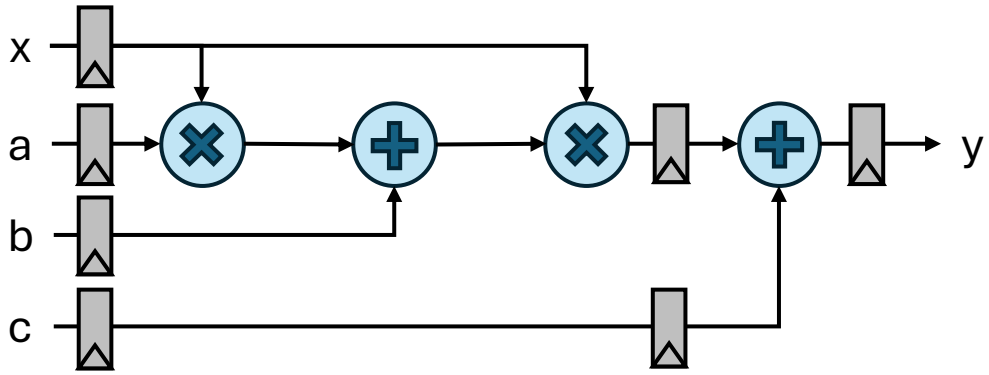
- Latency from x to y1 = 2
- Latency from x to y2 = 4
- Latency from x to y3 = 6
- No. of registers = **6**

Retiming Polynomial Circuit



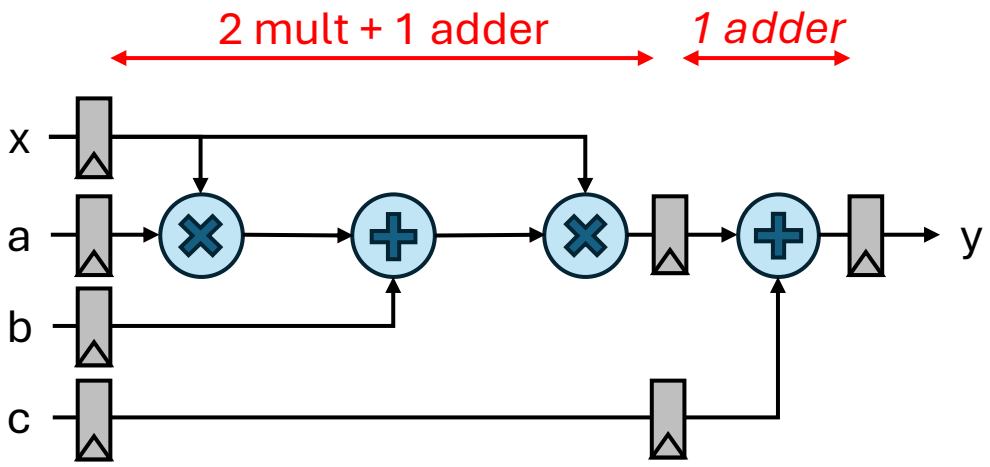
- Retiming can help recover from bad pipelining decisions by moving registers around using simple rules

Retiming Polynomial Circuit



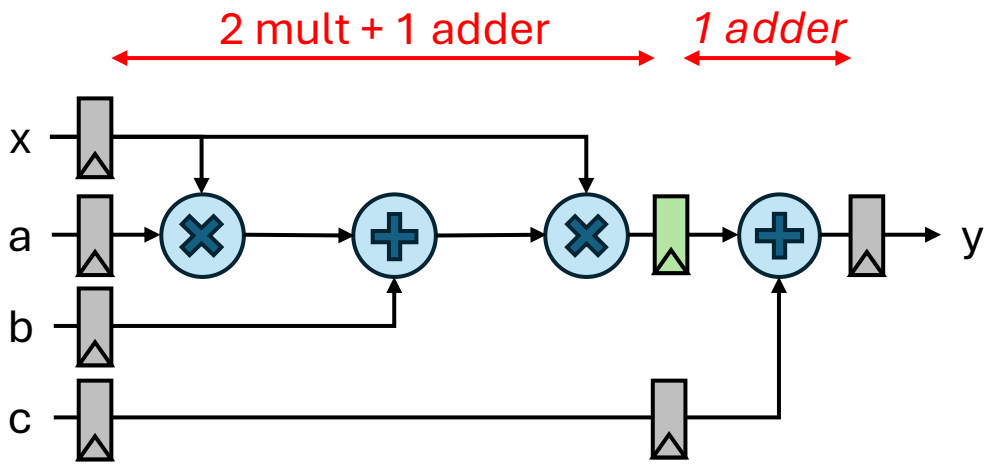
- Retiming can help recover from bad pipelining decisions by moving registers around using simple rules

Retiming Polynomial Circuit



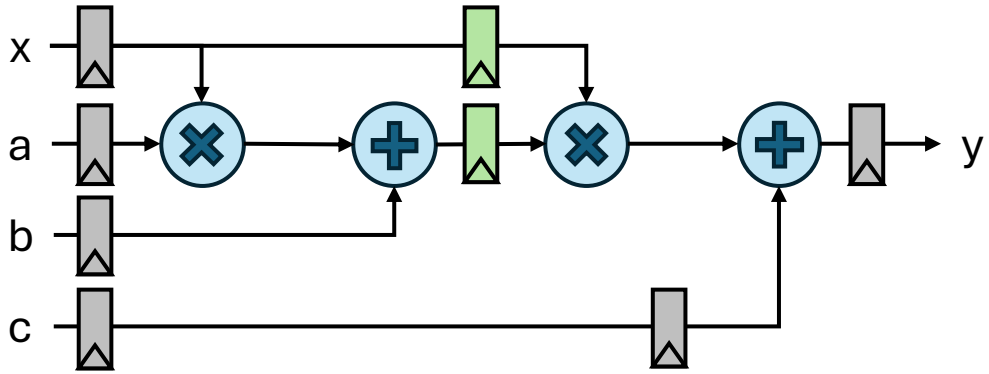
- Retiming can help recover from bad pipelining decisions by moving registers around using simple rules

Retiming Polynomial Circuit



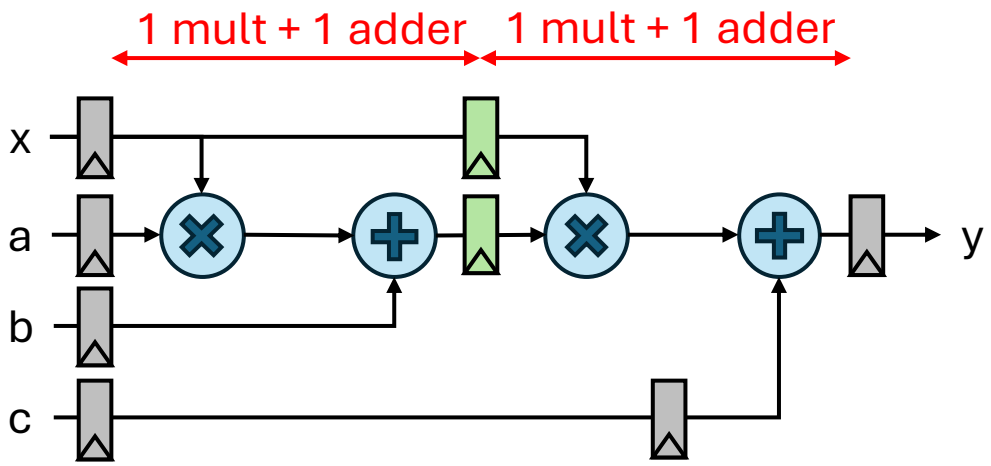
- Retiming can help recover from bad pipelining decisions by moving registers around using simple rules

Retiming Polynomial Circuit



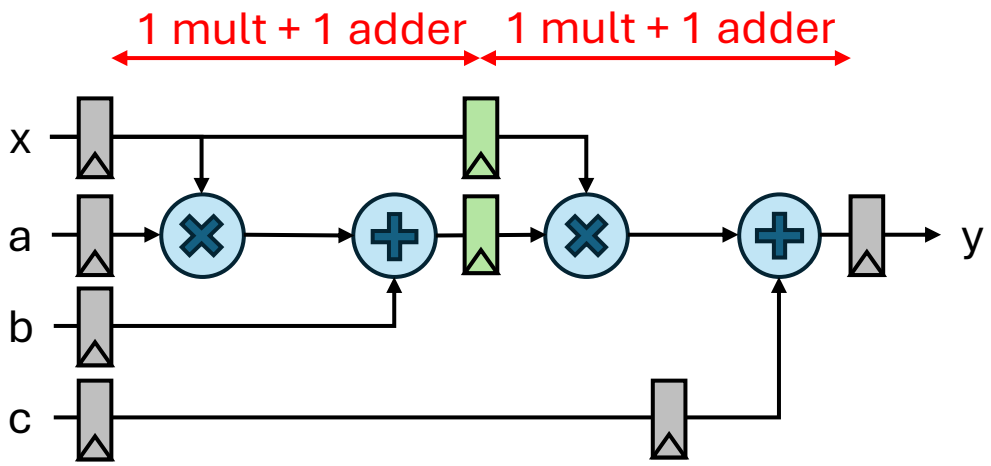
- Retiming can help recover from bad pipelining decisions by moving registers around using simple rules

Retiming Polynomial Circuit



- Retiming can help recover from bad pipelining decisions by moving registers around using simple rules

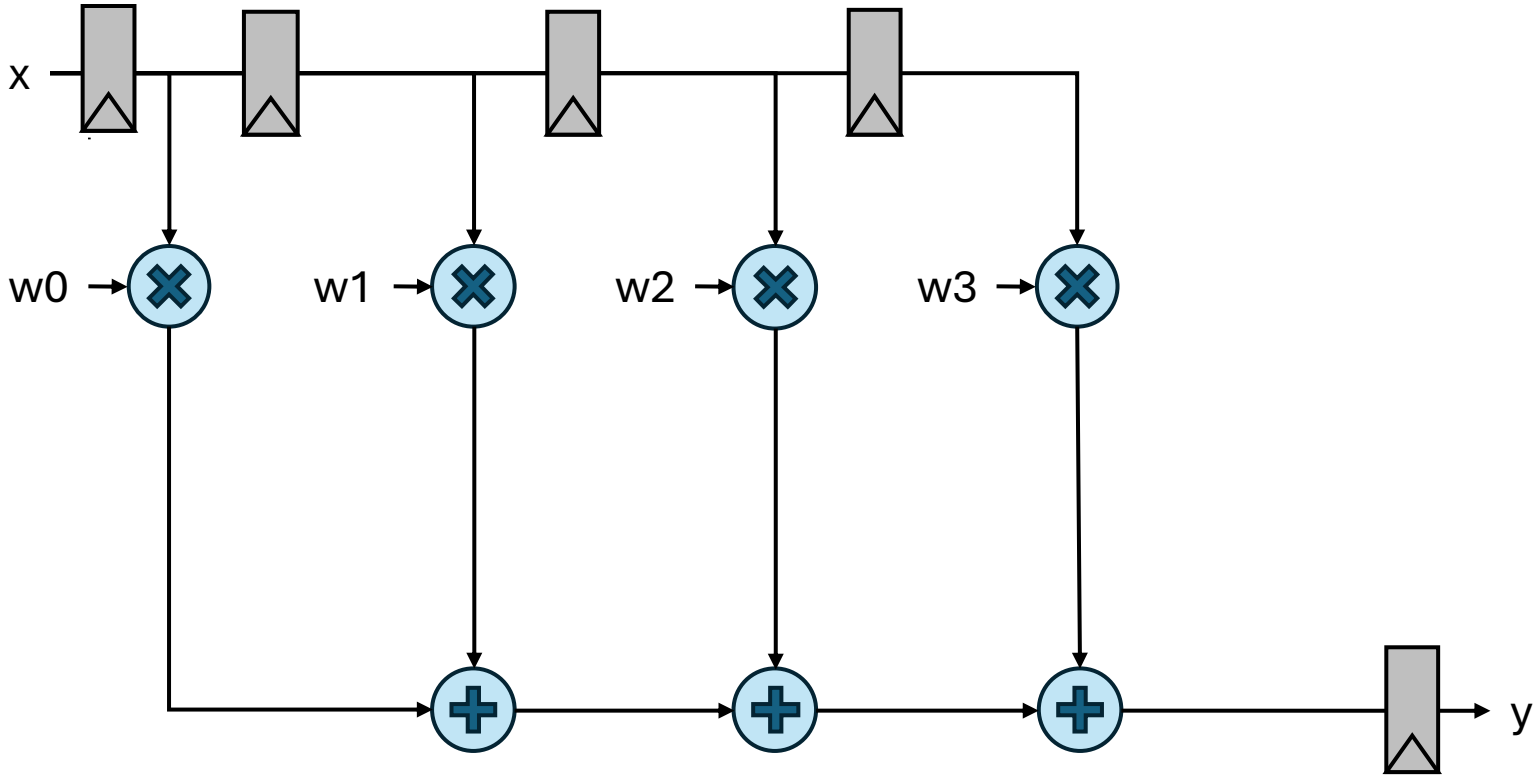
Retiming Polynomial Circuit



- Retiming can help recover from bad pipelining decisions by moving registers around using simple rules
- Throughput is improved because critical path is shorter
- Behavior of the circuit is preserved

Back to the FIR Filter Example …

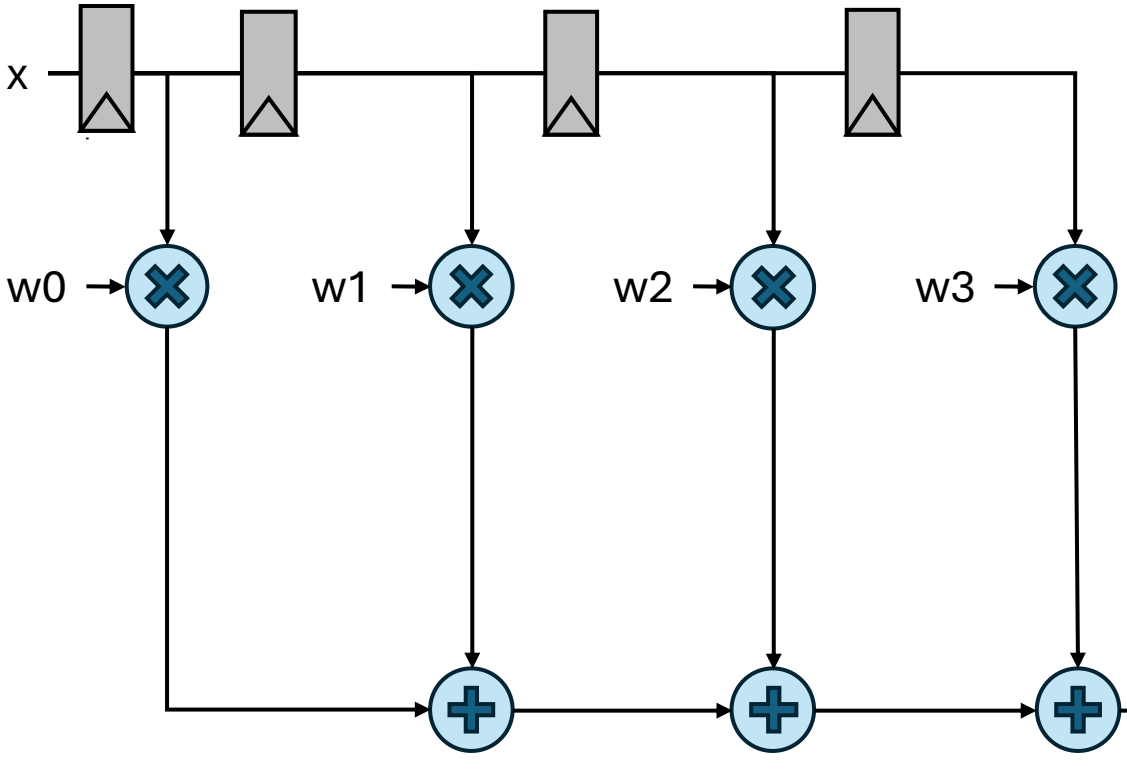
- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

$$y[t] = x[t]\,w0 + x[t-1]\,w1 + x[t-2]\,w2 + x[t-3]\,w3$$

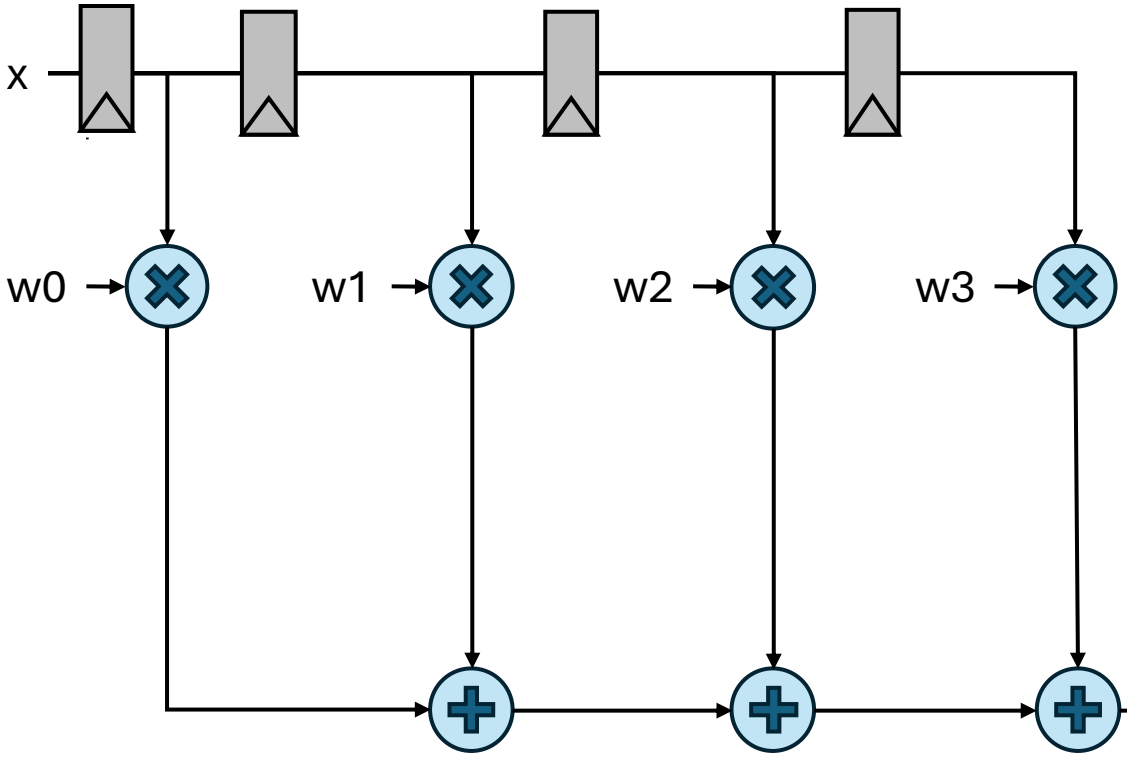


Let's assume we are given the budget of adding 3 pipeline stages to the circuit …

Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:

$$y[t] = x[t]\,w0 + x[t-1]\,w1 + x[t-2]\,w2 + x[t-3]\,w3$$

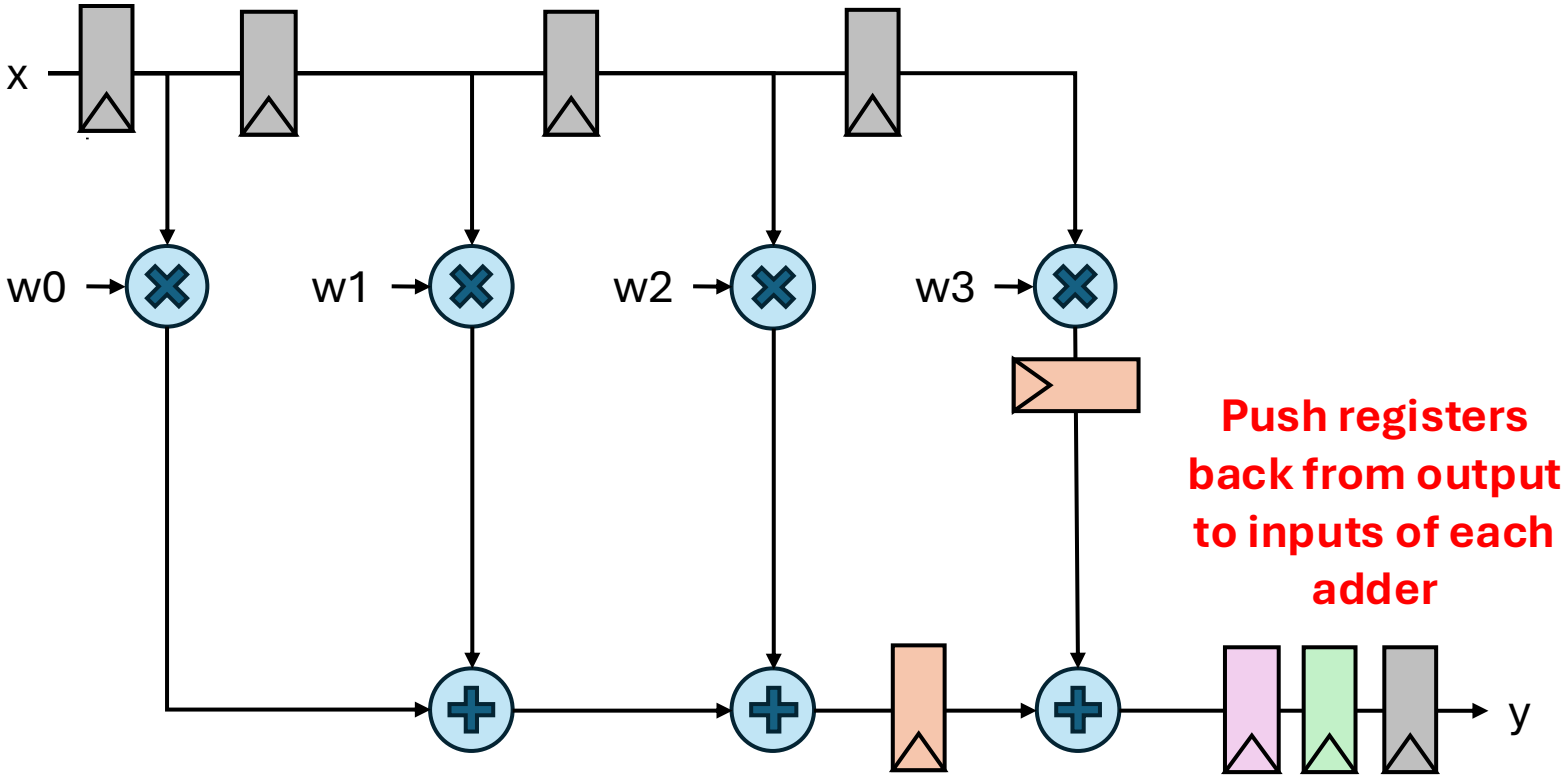


Let's assume we are given the budget of adding 3 pipeline stages to the circuit …

Pipeline by adding registers at the output

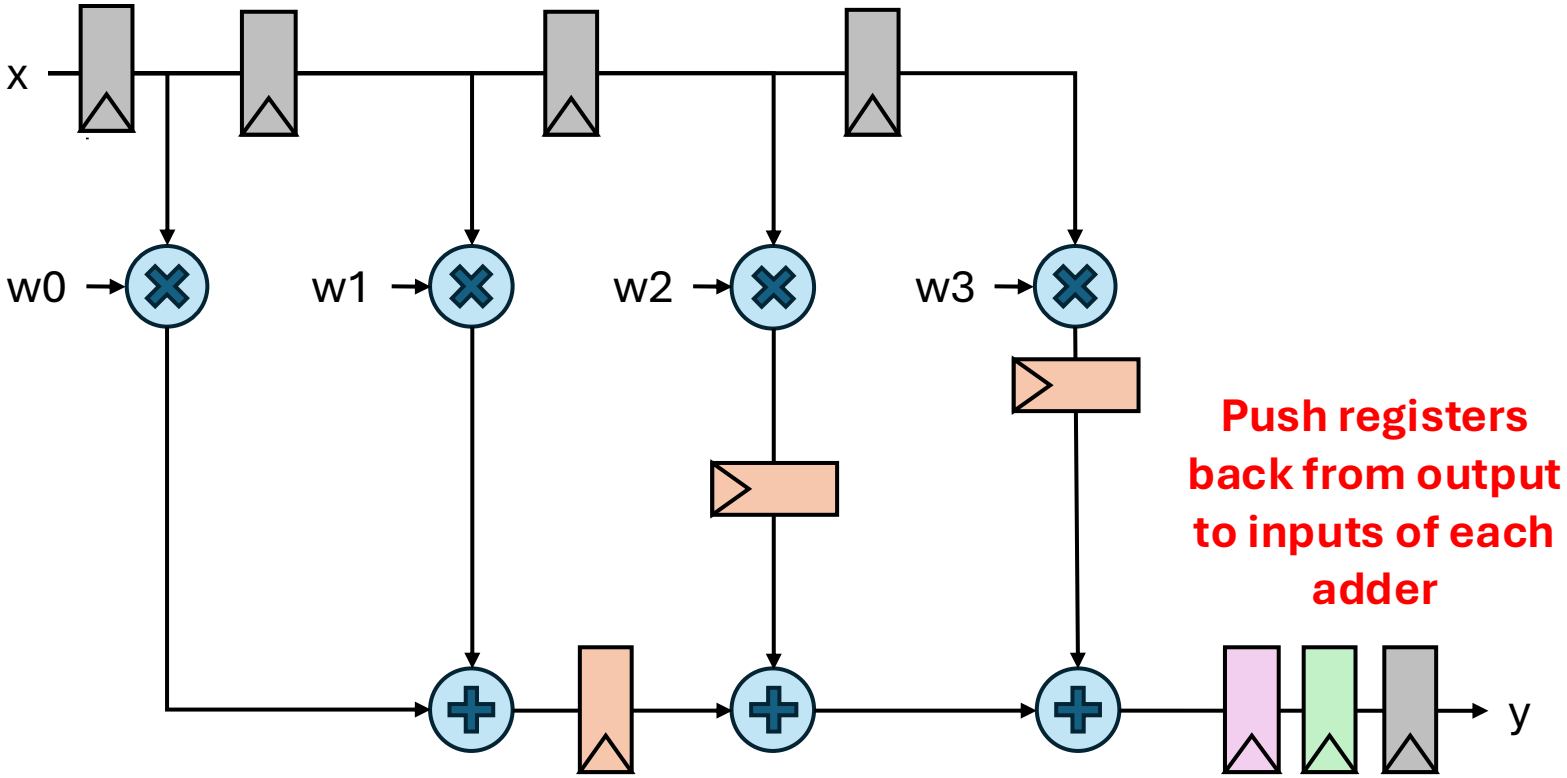
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



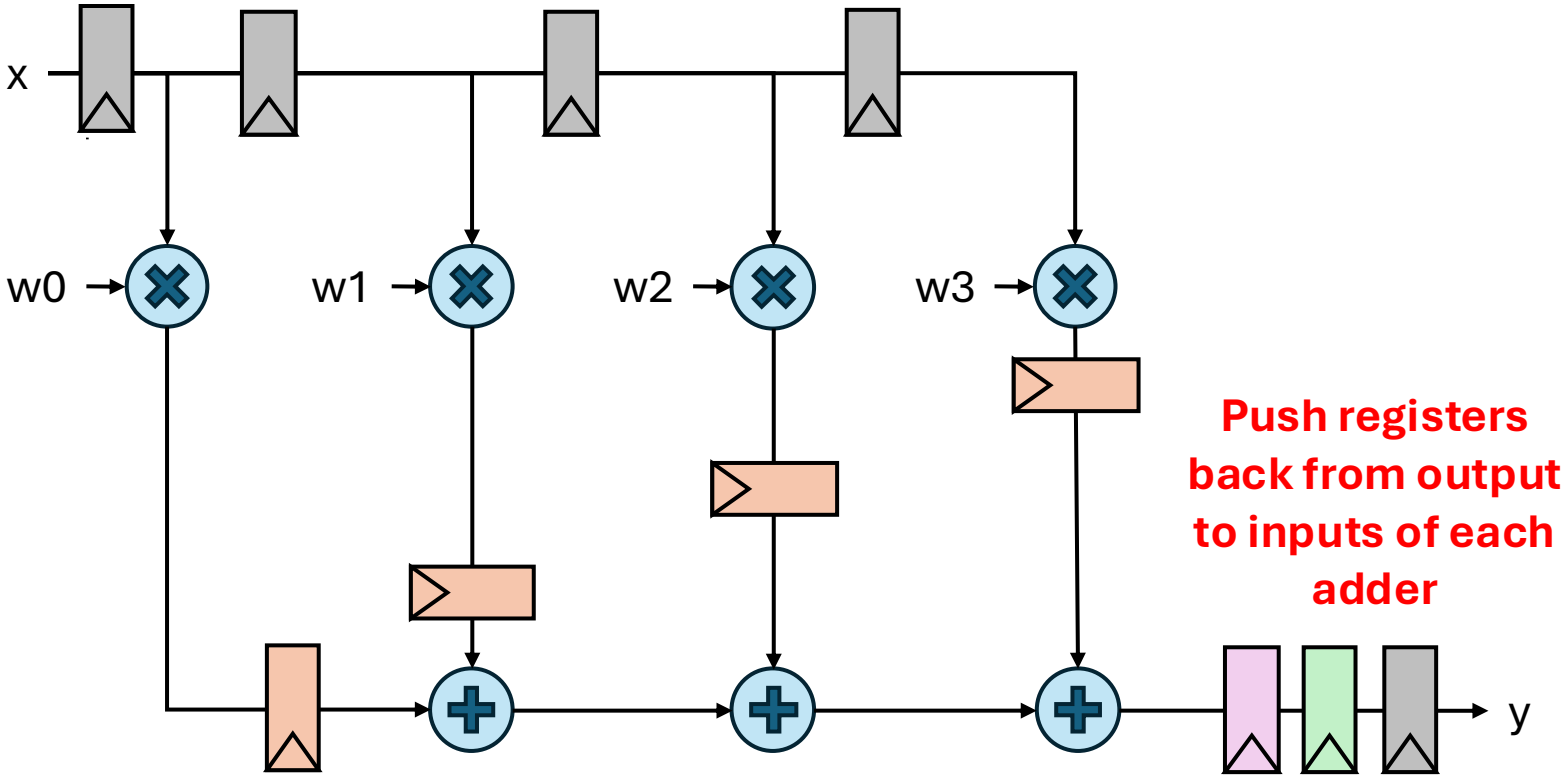
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



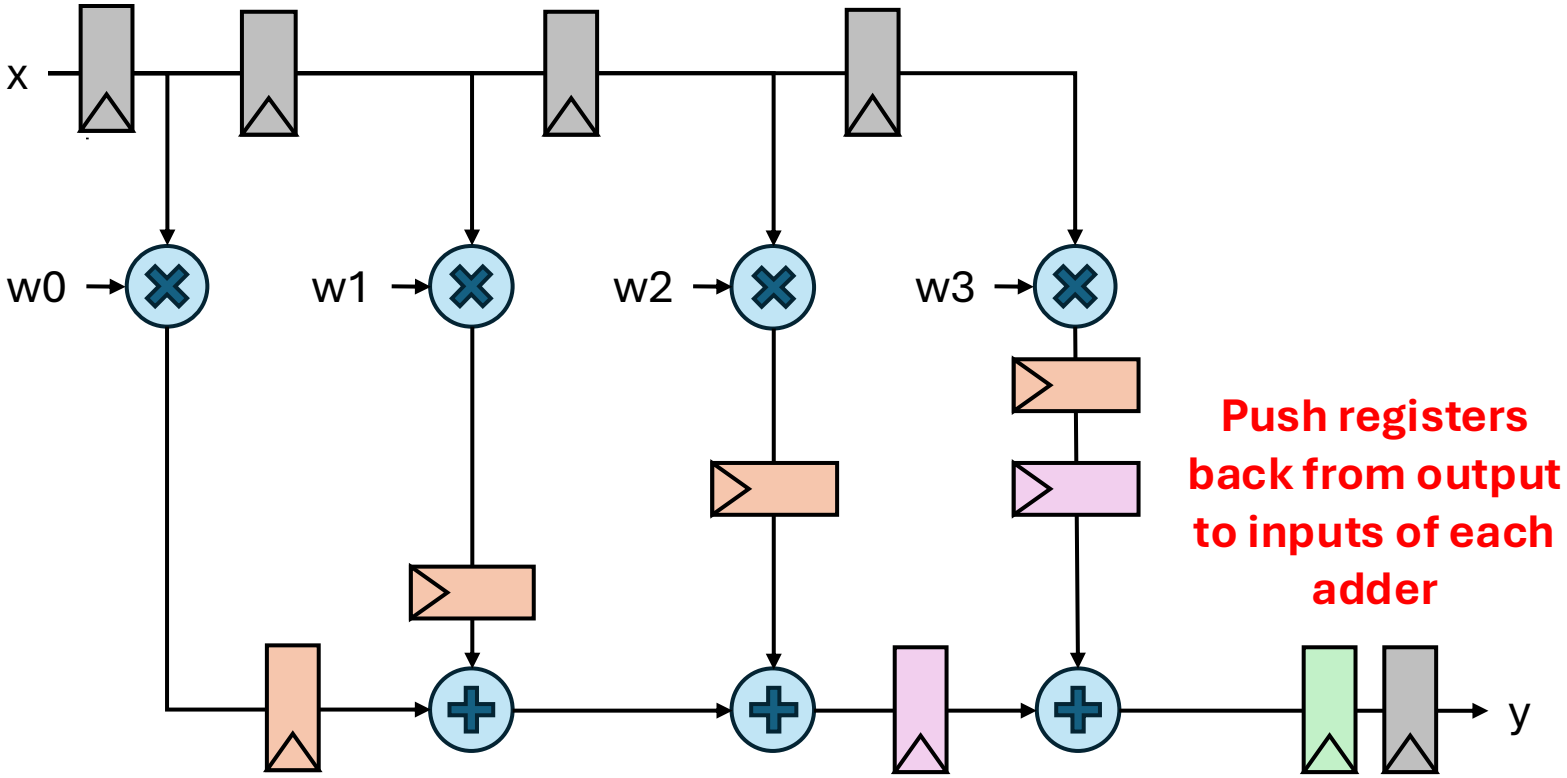
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



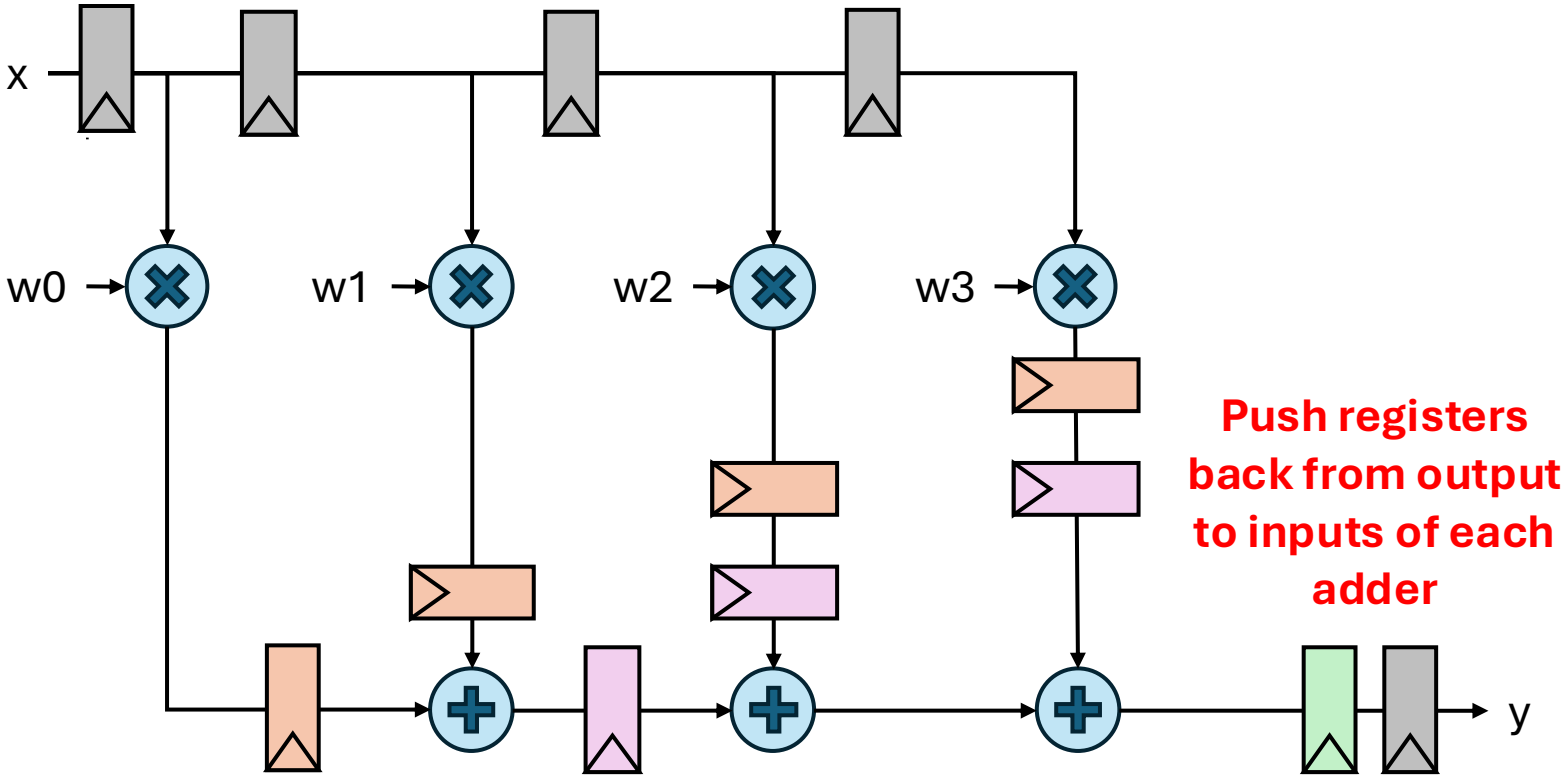
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



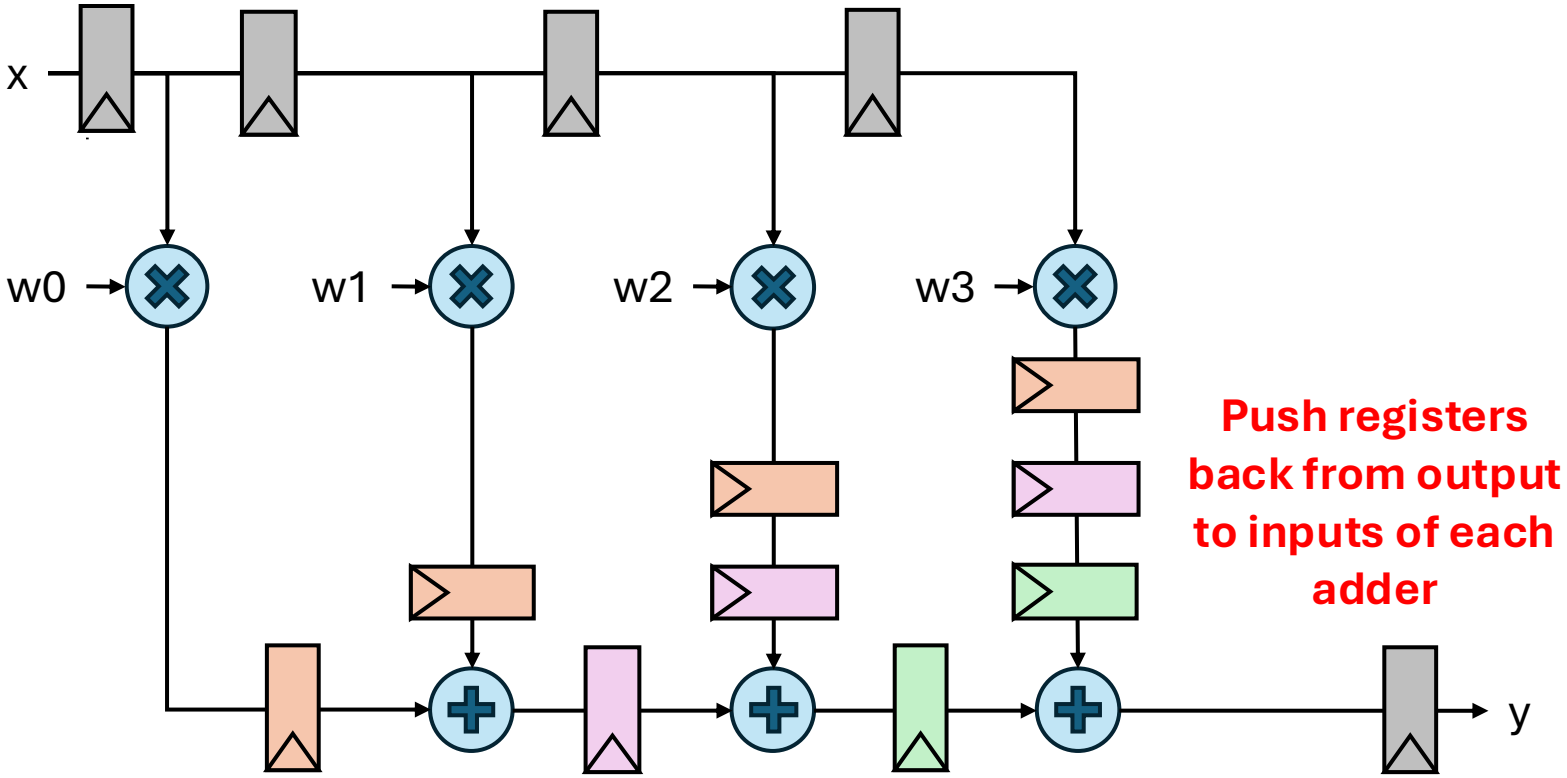
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



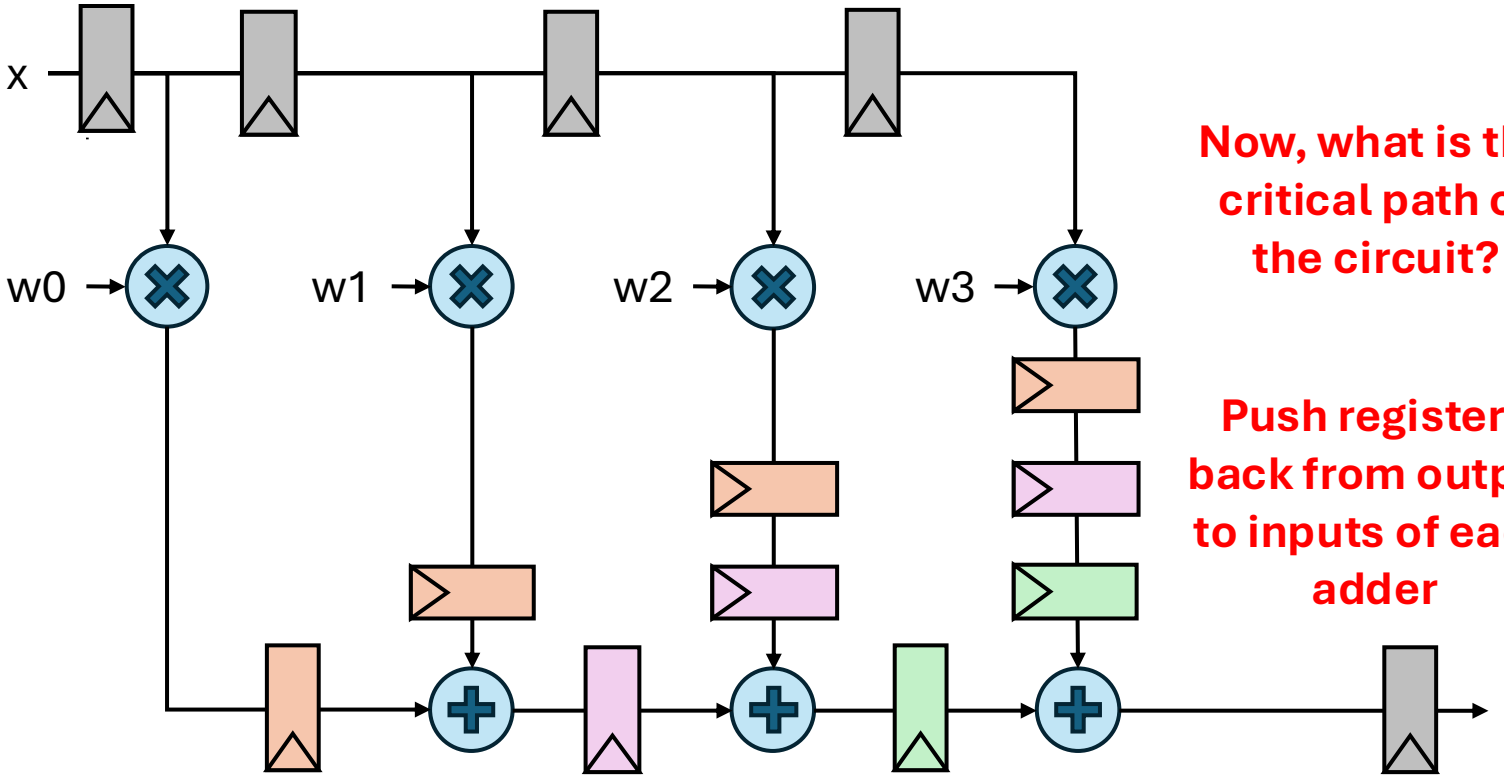
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$

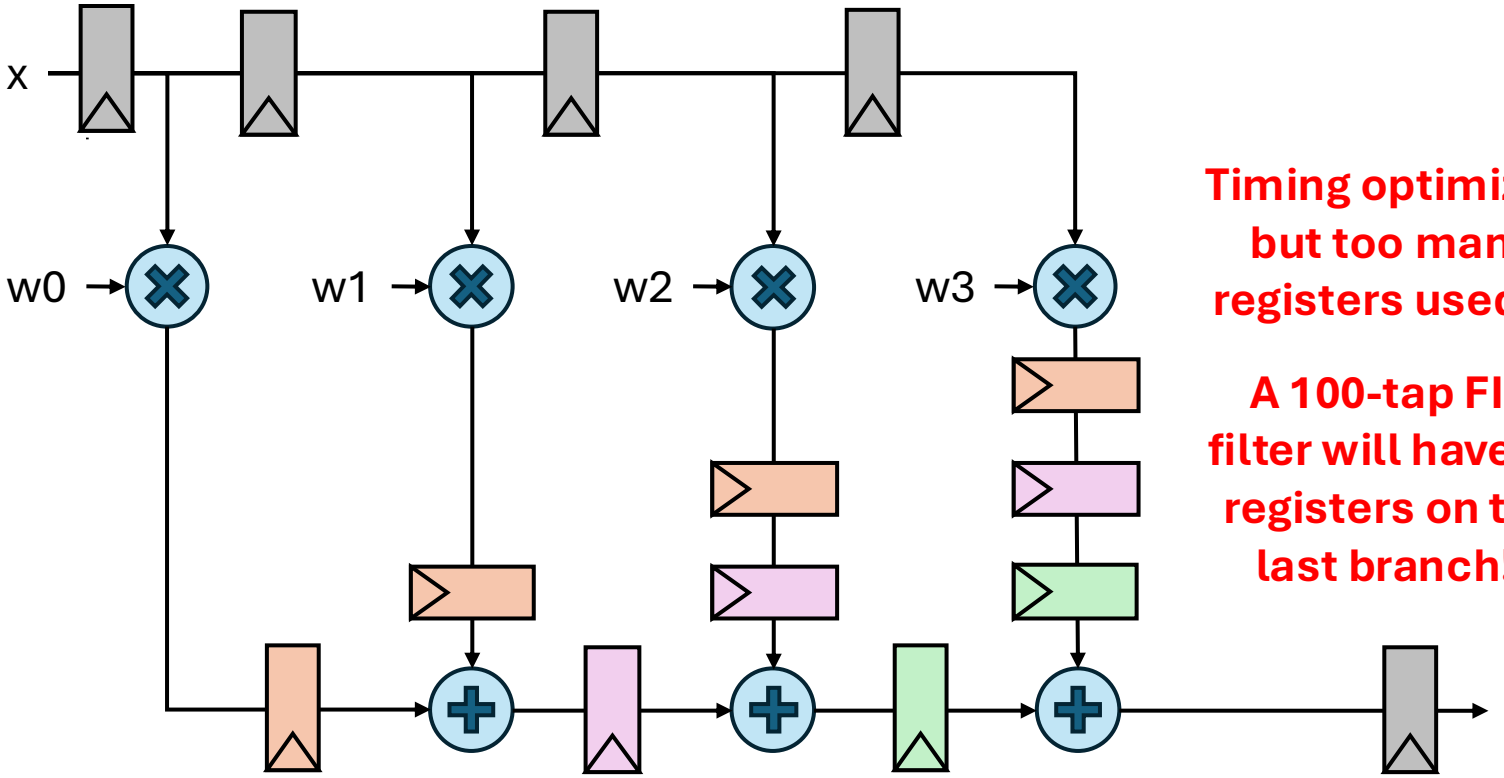


Now, what is the critical path of the circuit?

Push registers back from output to inputs of each adder

Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$

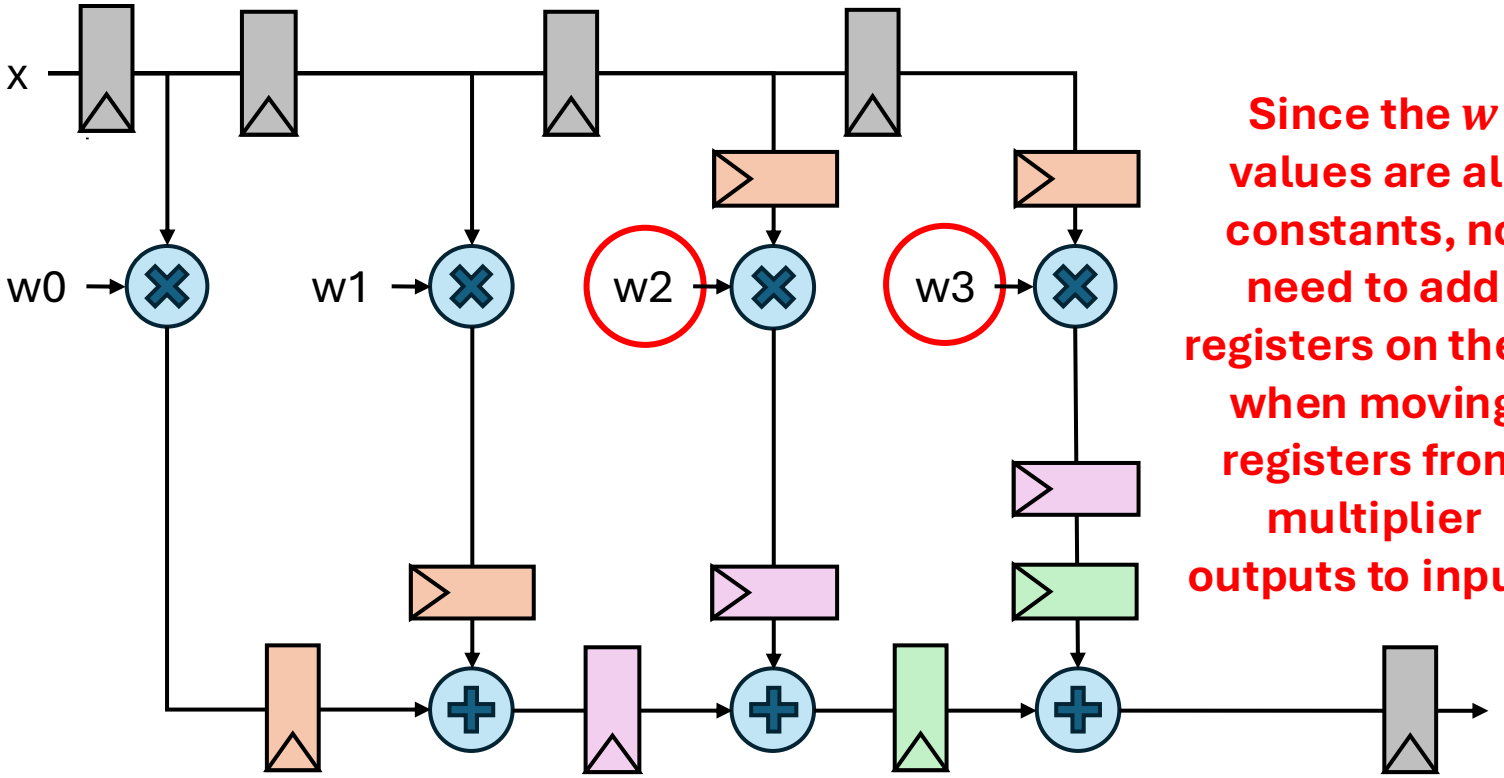


Timing optimized but too many registers used …

A 100-tap FIR filter will have 99 registers on the last branch!!

Back to the FIR Filter Example …

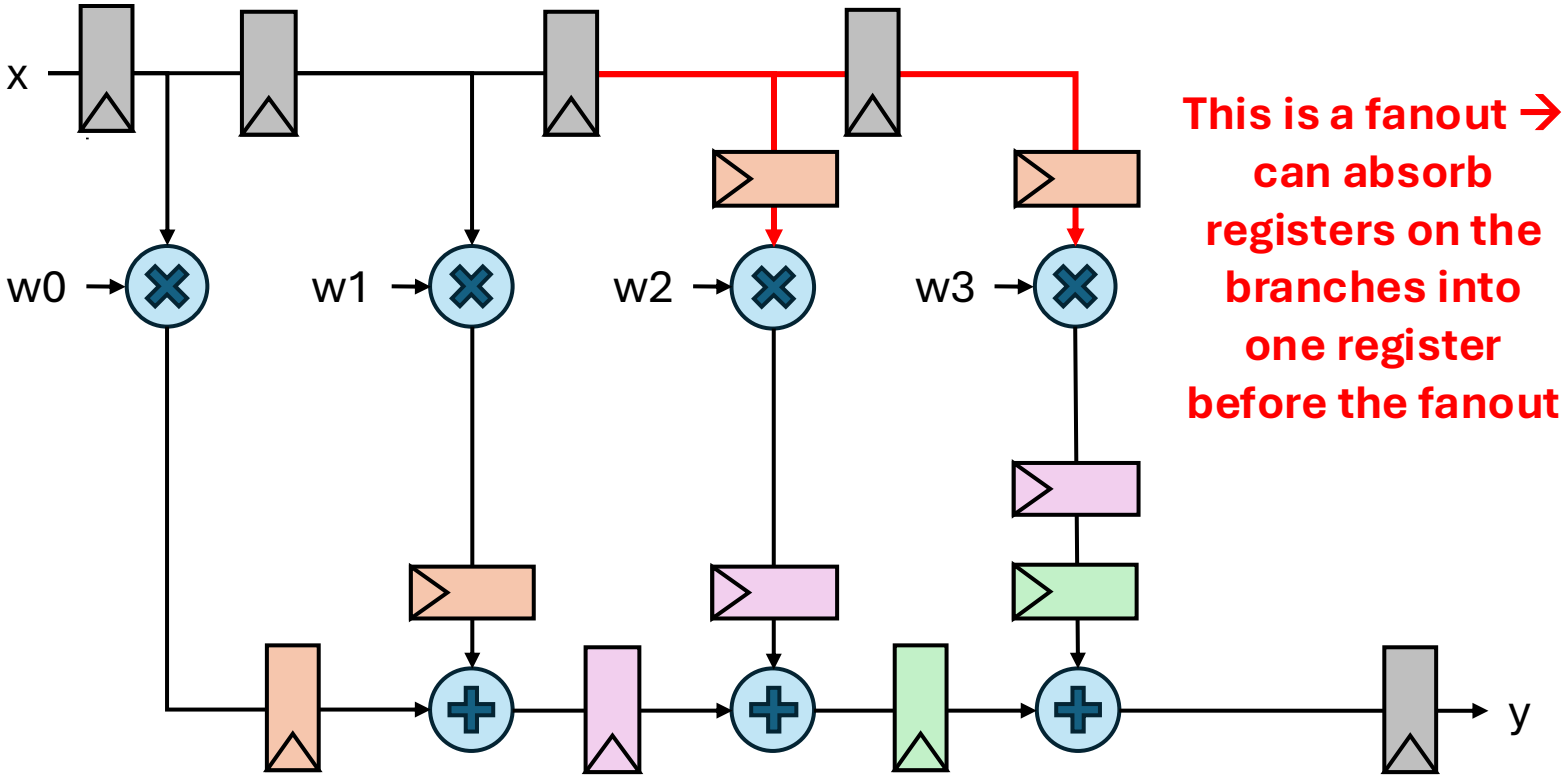
- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



Since the *w* values are all constants, no need to add registers on them when moving registers from multiplier outputs to inputs

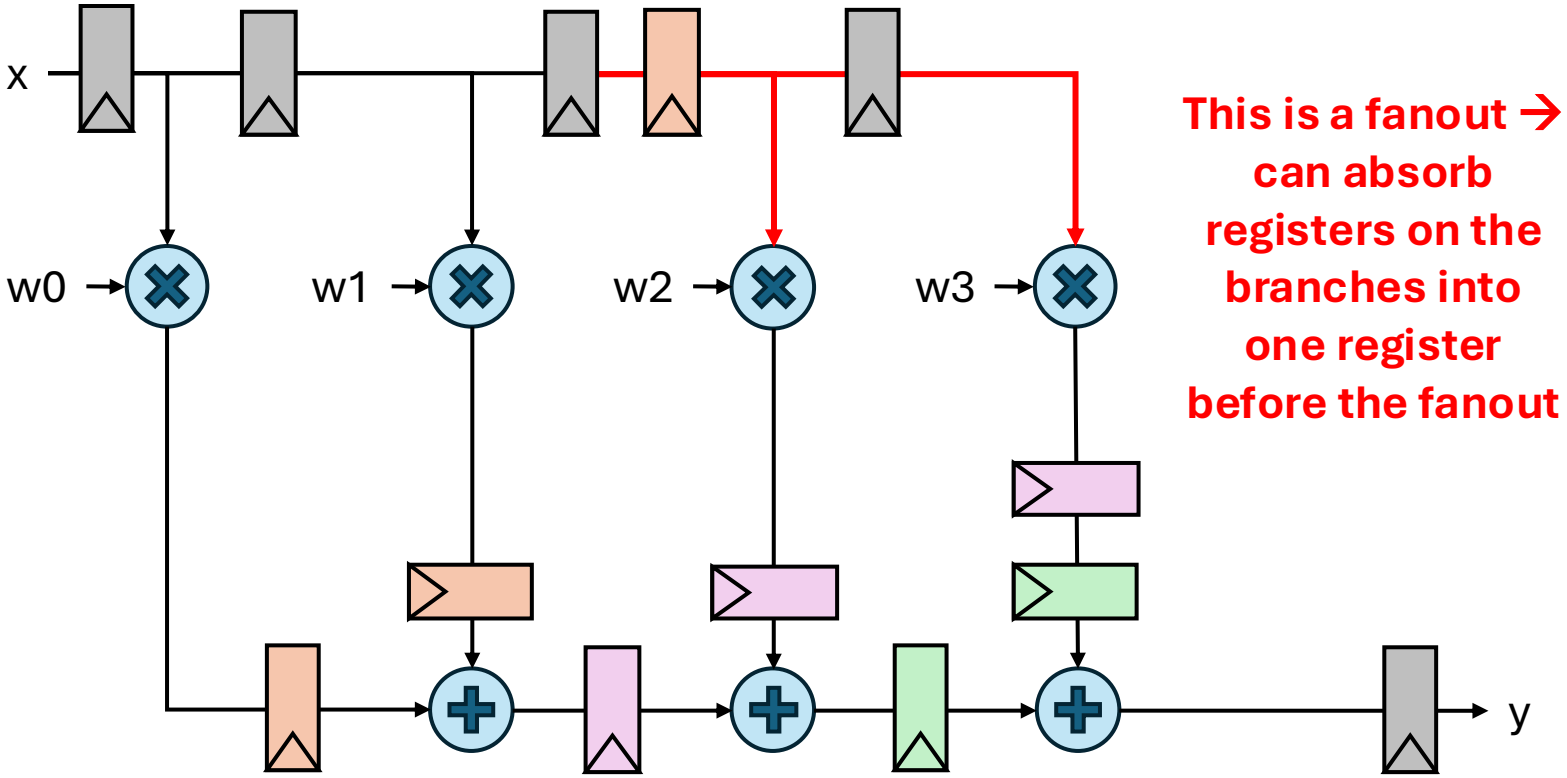
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



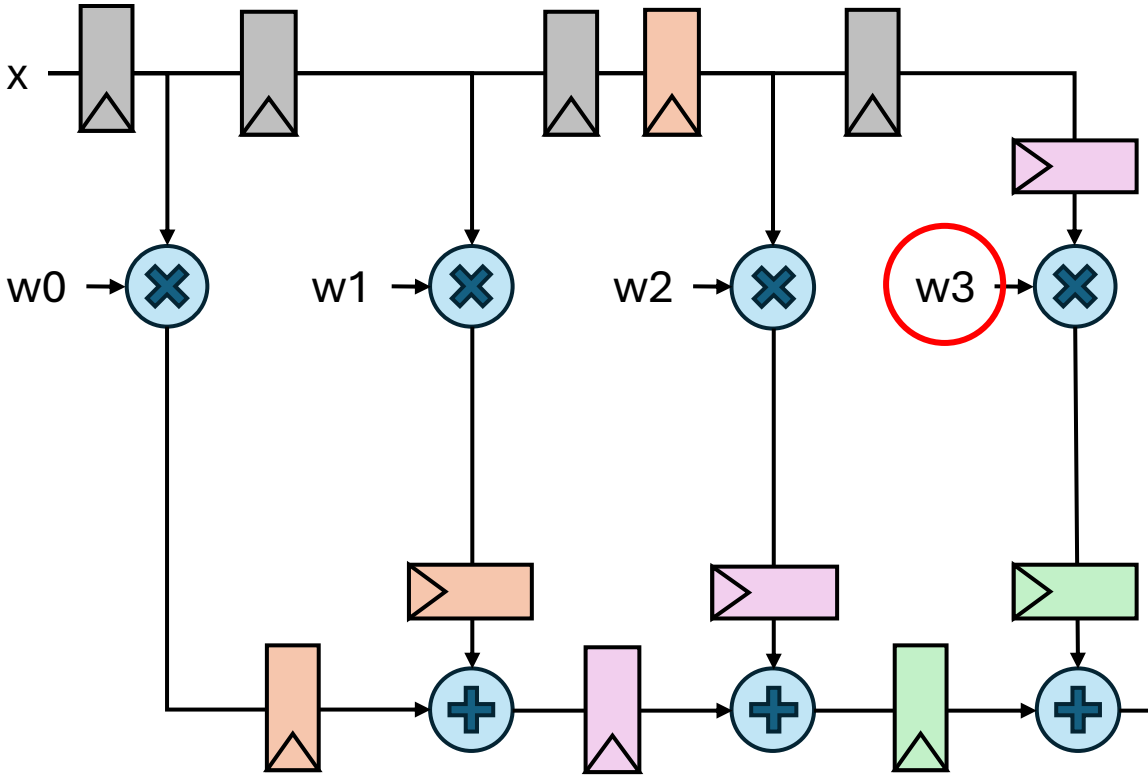
Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



Back to the FIR Filter Example …

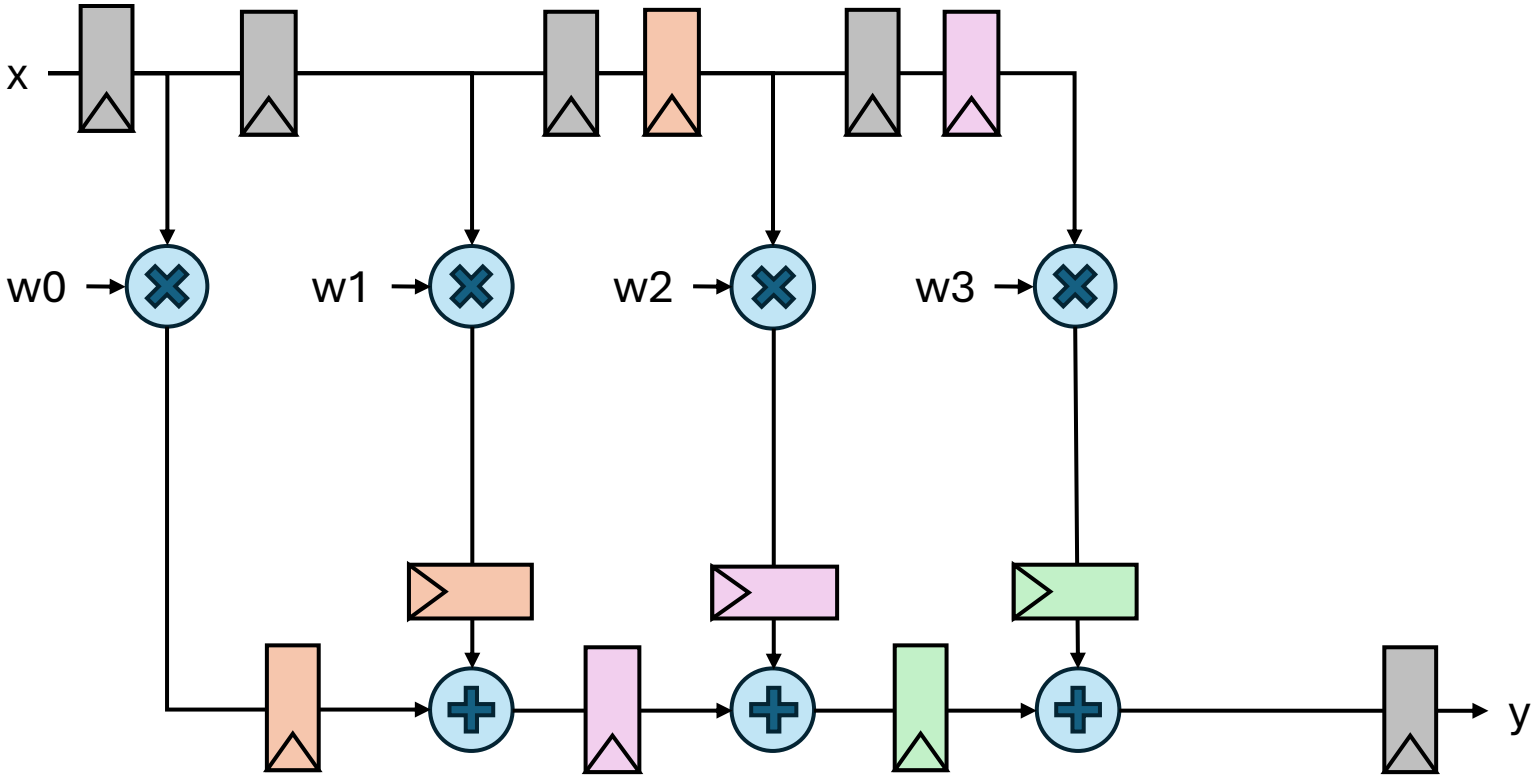
- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



Since the *w* values are all constants, no need to add registers on them when moving registers from multiplier outputs to inputs

Back to the FIR Filter Example …

- FIR filters are common structures for digital signal processing
- 1-D FIR filter with 4 taps performs the following computation:
  $y[t] = x[t]\, w0 + x[t-1]\, w1 + x[t-2]\, w2 + x[t-3]\, w3$



Retiming

- Retiming is an approach for optimizing pipelining decisions

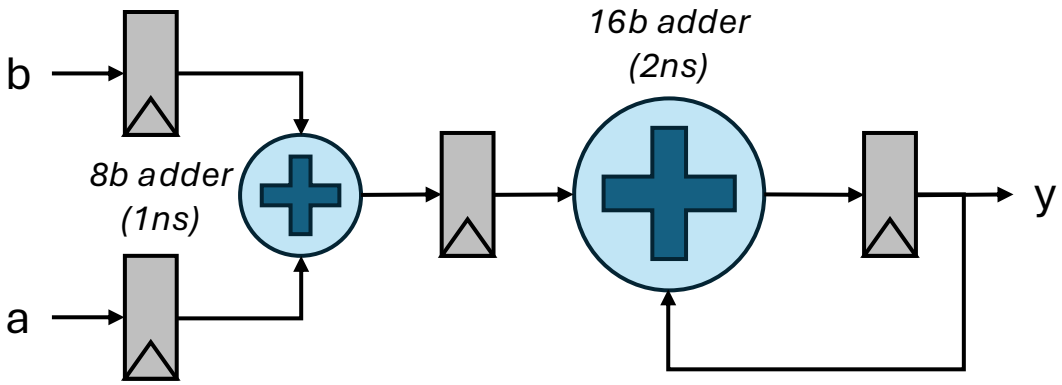
Retiming

- Retiming is an approach for optimizing pipelining decisions
- Retiming algorithms are implemented in most modern CAD tools → Beyond the scope of this course

Retiming

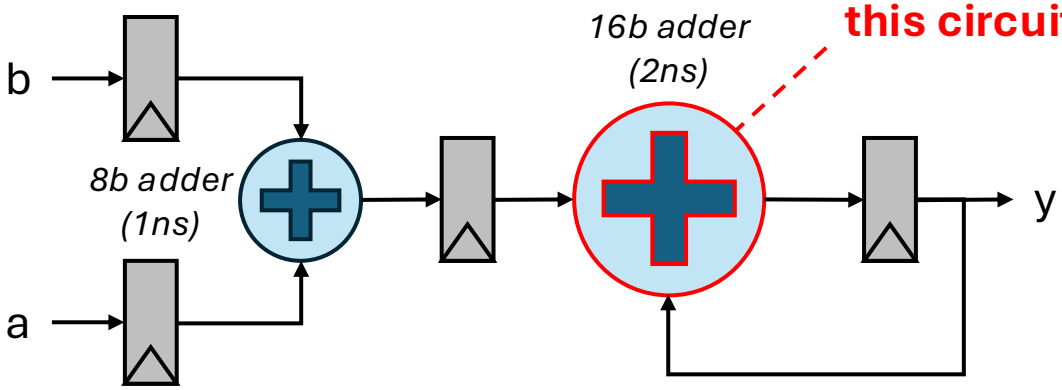
- Retiming is an approach for optimizing pipelining decisions
- Retiming algorithms are implemented in most modern CAD tools → Beyond the scope of this course
- Manual heuristic pipelining
  - Add a bunch of registers at the outputs of the circuit
  - Push a register back as far as possible until you hit the inputs or another register
  - Stop adding registers if there are no more locations to push
  - Move registers around using some simple rules to optimize frequency first, then minimize the number of used registers

Pipelining Feedback Paths



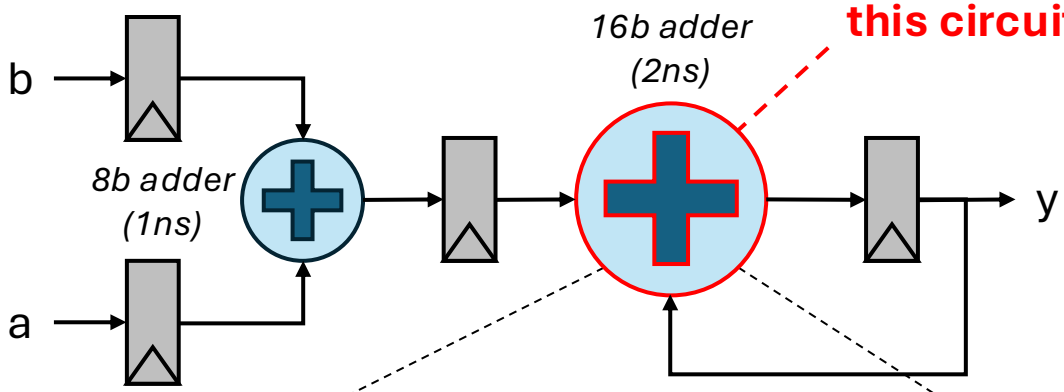
Pipelining Feedback Paths

16b adder is the critical path of this circuit (2ns)

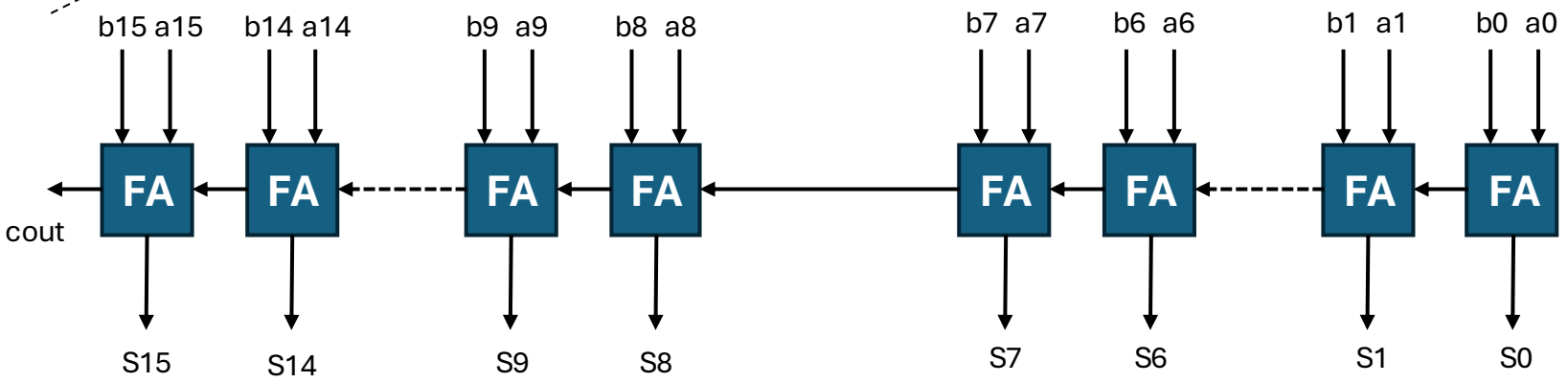


Pipelining Feedback Paths

16b adder is the critical path of this circuit (2ns)

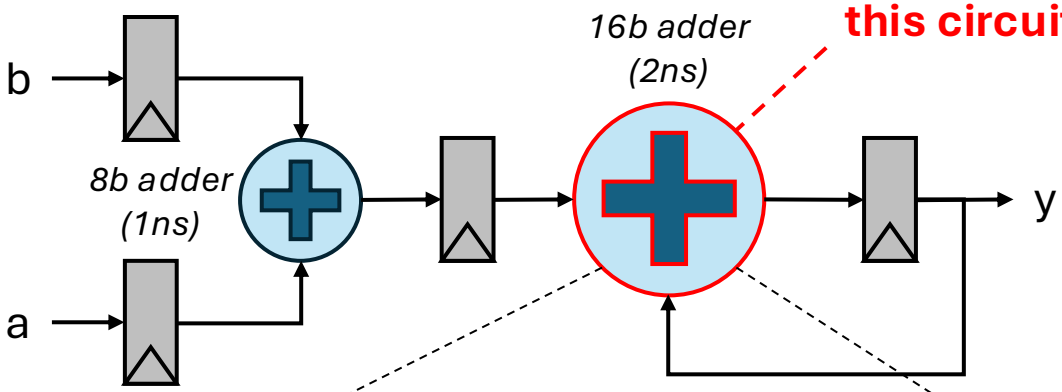


Pipelining an adder is straightforward!

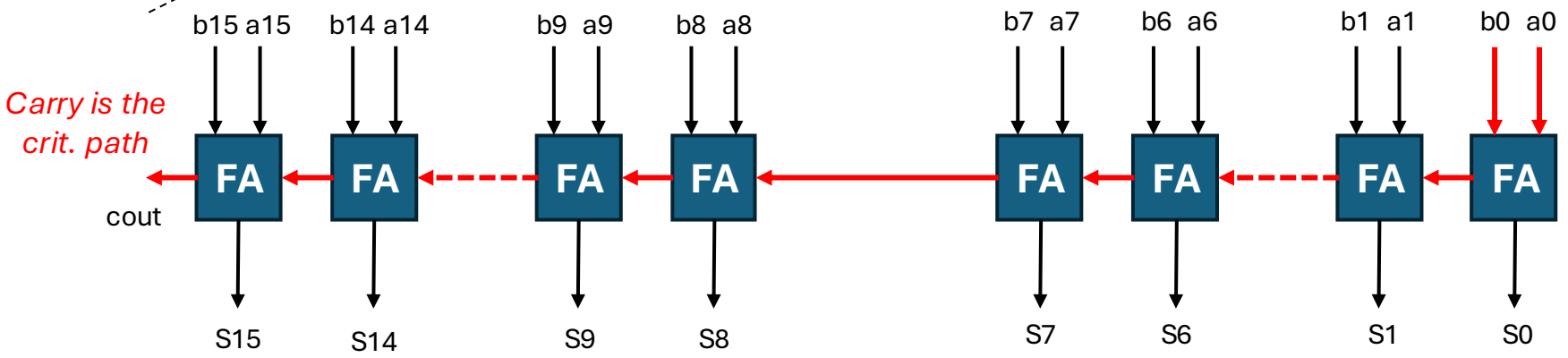


Pipelining Feedback Paths

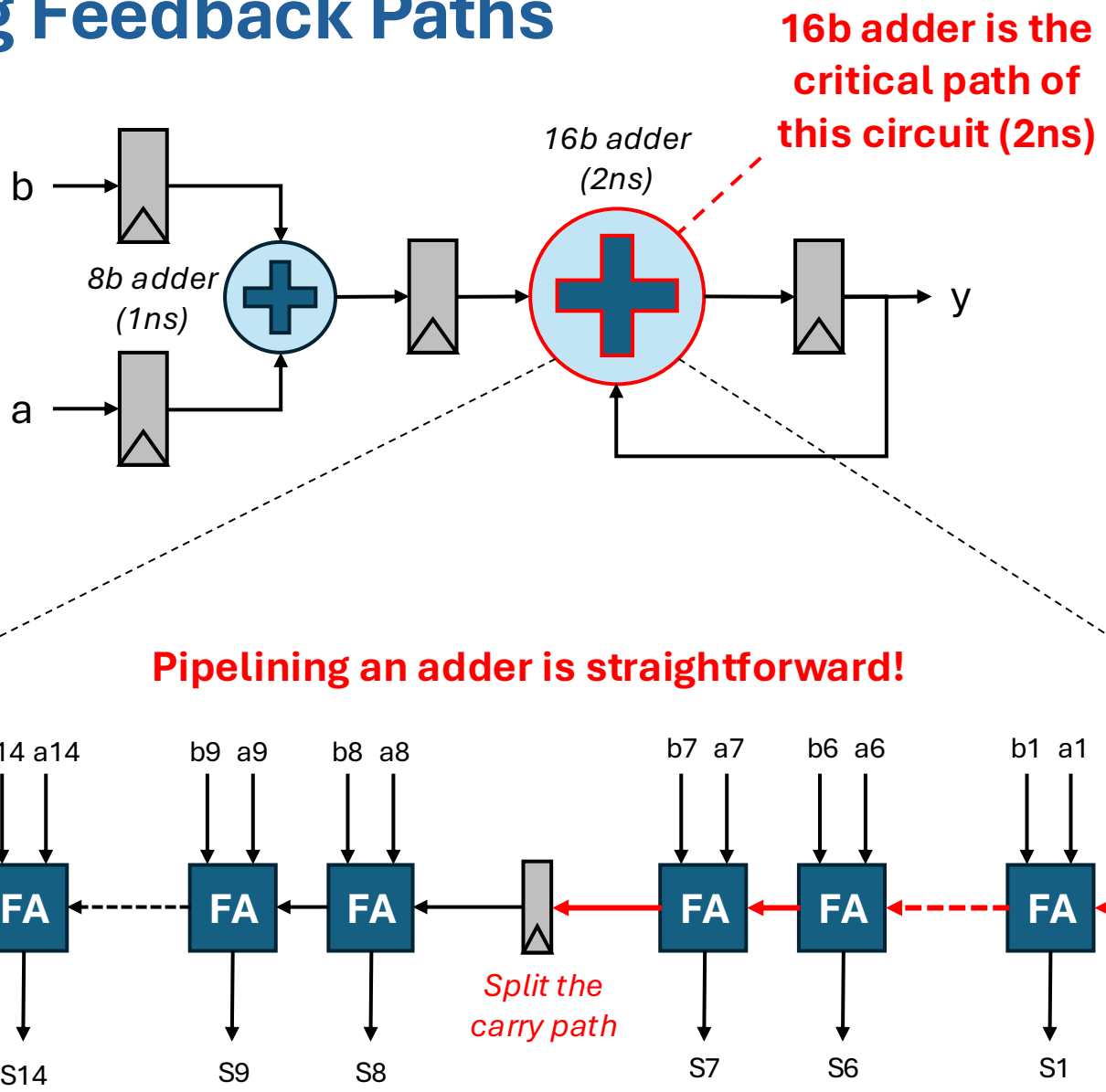
16b adder is the critical path of this circuit (2ns)



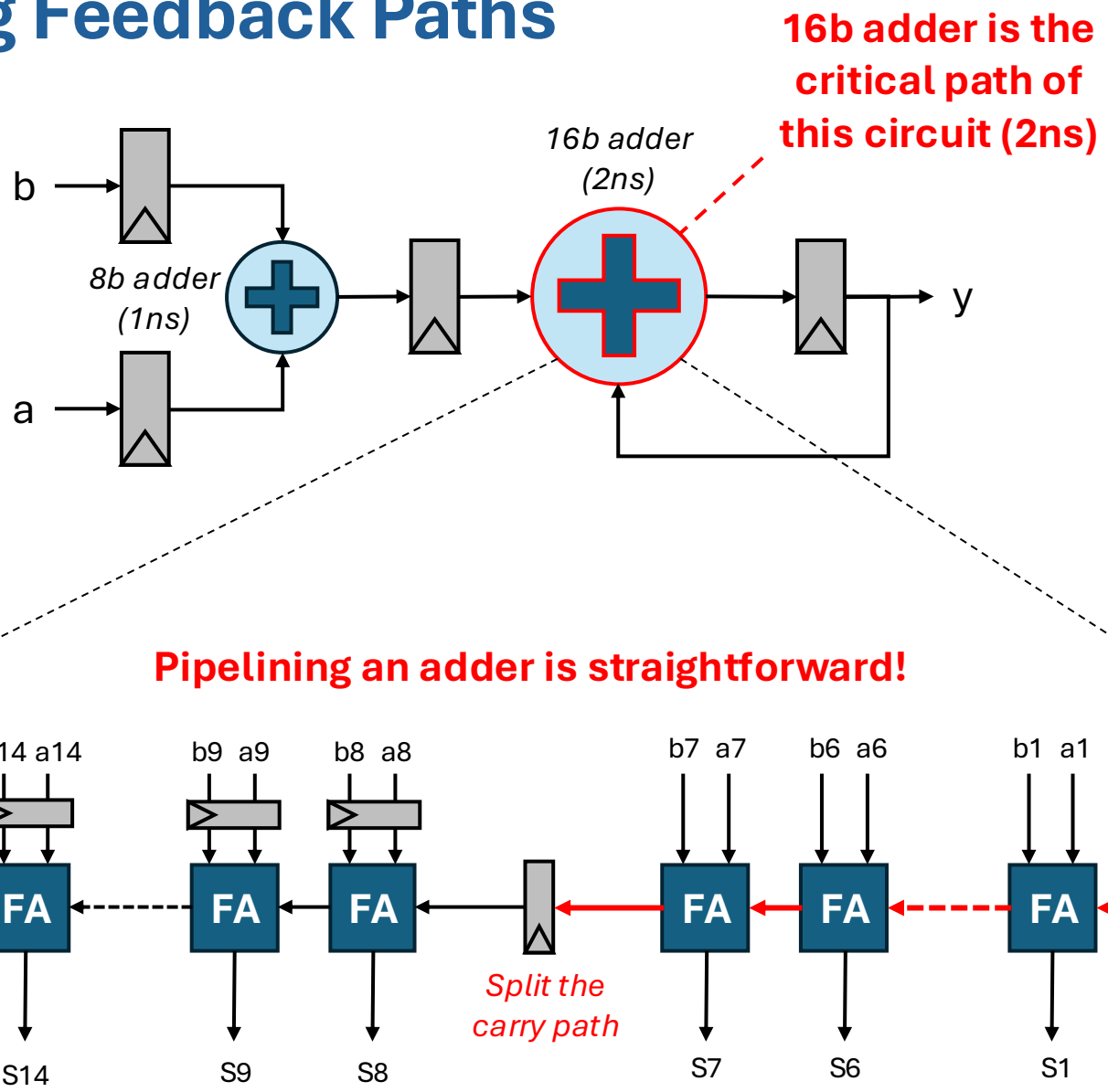
Pipelining an adder is straightforward!



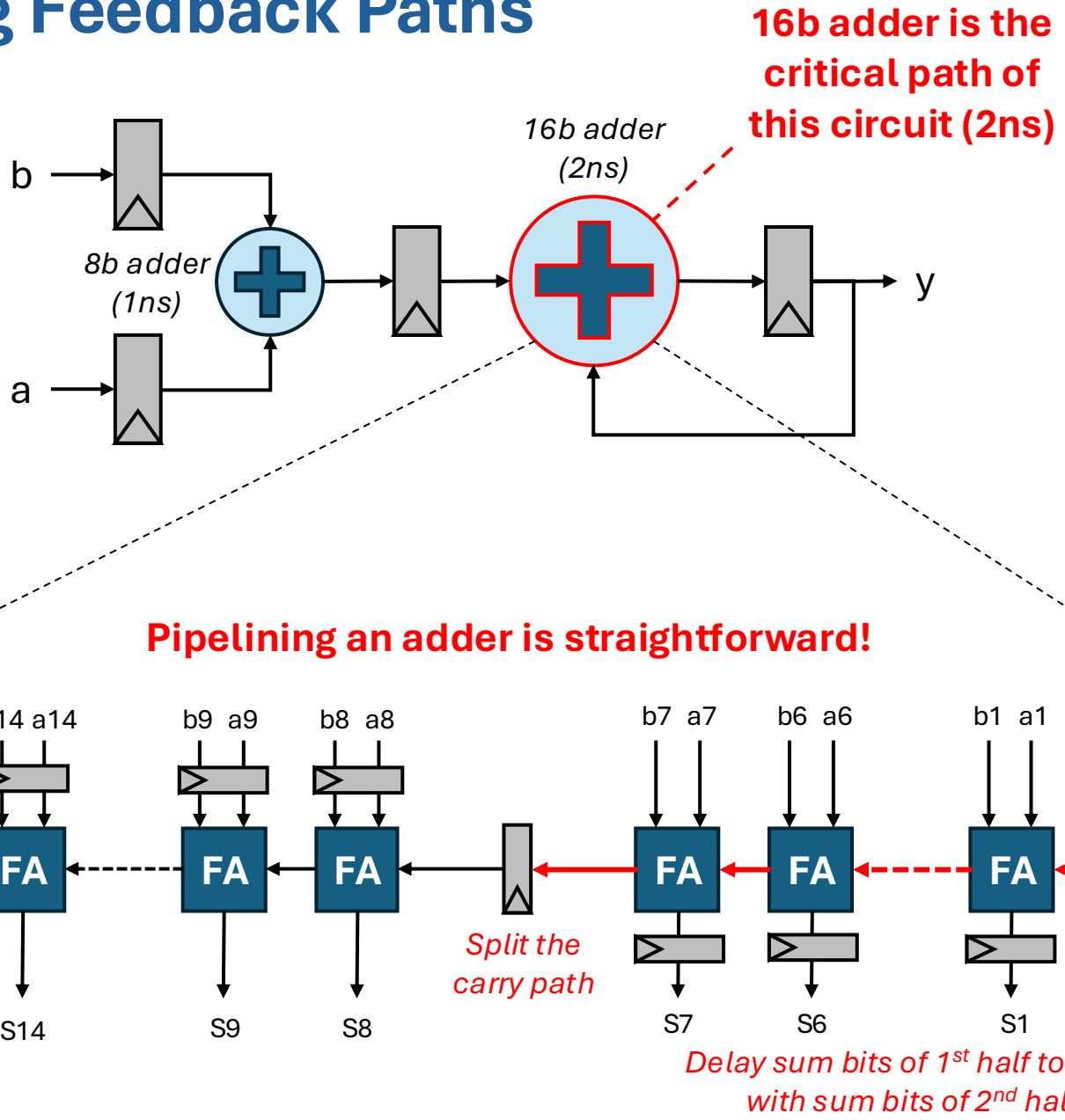
Pipelining Feedback Paths



Pipelining Feedback Paths

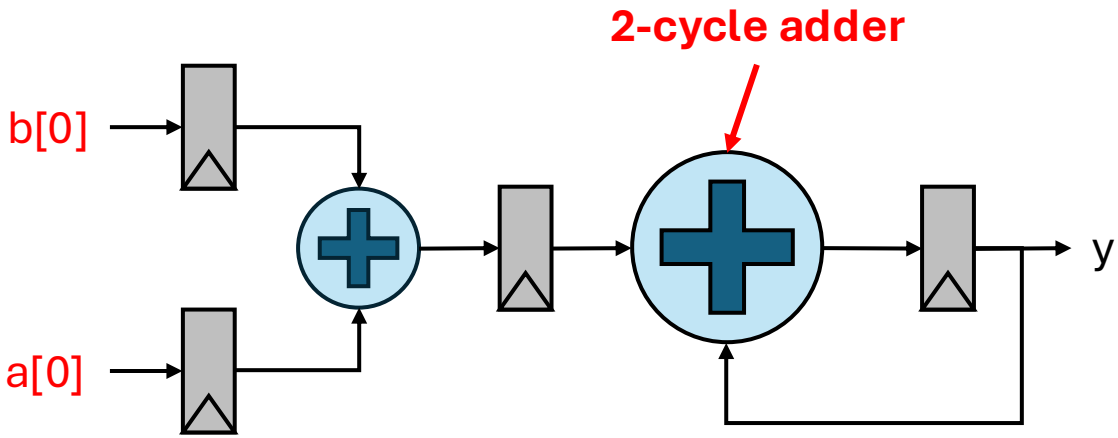


Pipelining Feedback Paths



Pipelining Feedback Paths

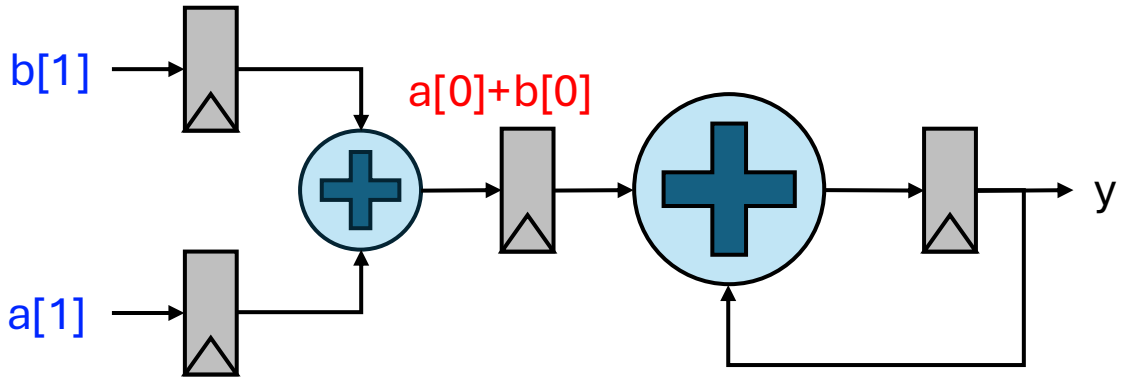
- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



Outputs:

Pipelining Feedback Paths

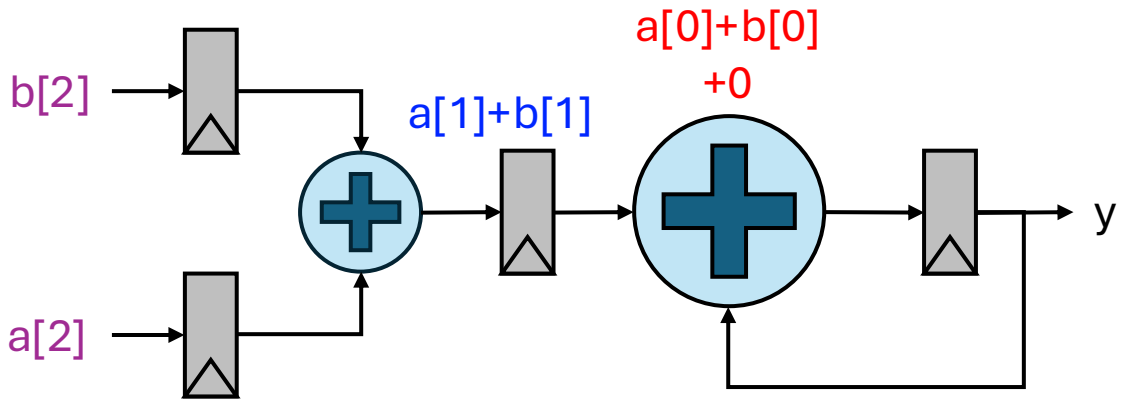
- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



Outputs:

Pipelining Feedback Paths

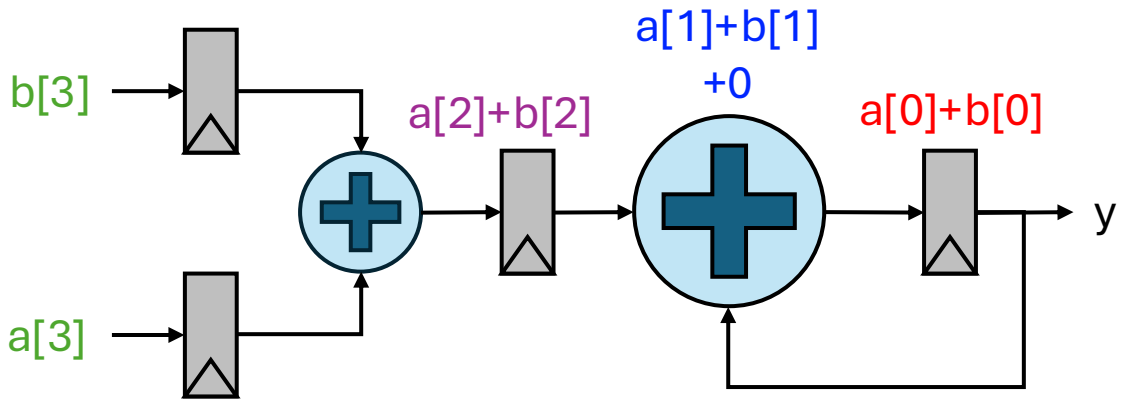
- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



Outputs:

Pipelining Feedback Paths

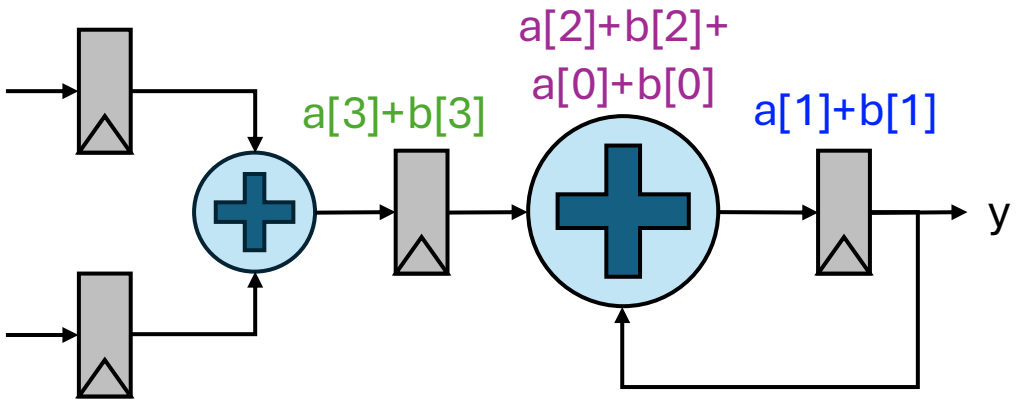
- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



Outputs: a[0]+b[0] ✔

Pipelining Feedback Paths

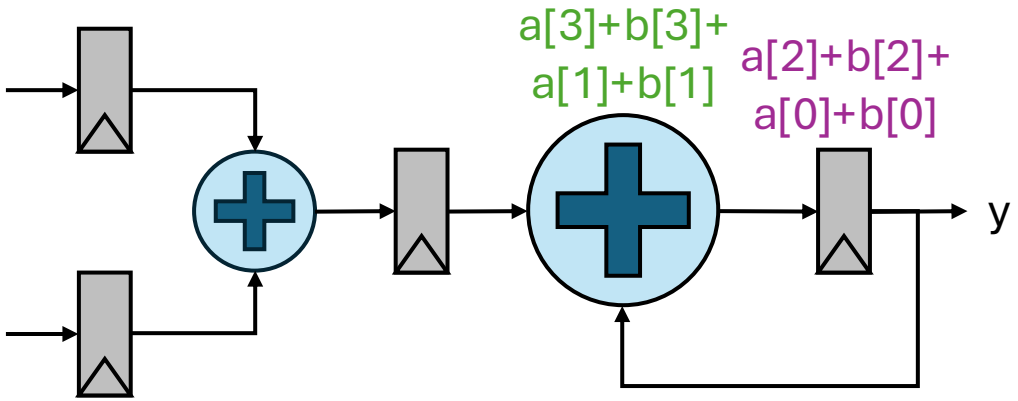
- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



Outputs: a[0]+b[0] ✓
a[1]+b[1] ✗

Pipelining Feedback Paths

- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



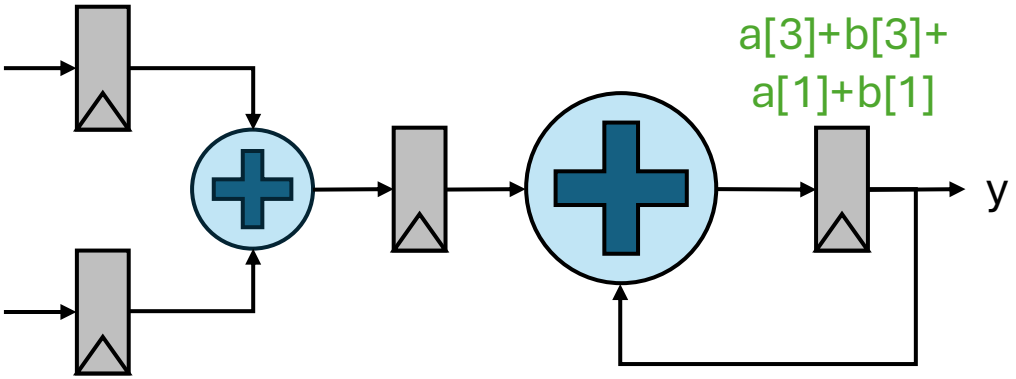
Outputs: a[0]+b[0] ✓

a[1]+b[1] ✗

a[2]+b[2]+a[0]+b[0] ✗

Pipelining Feedback Paths

- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



Outputs: a[0]+b[0] ✓

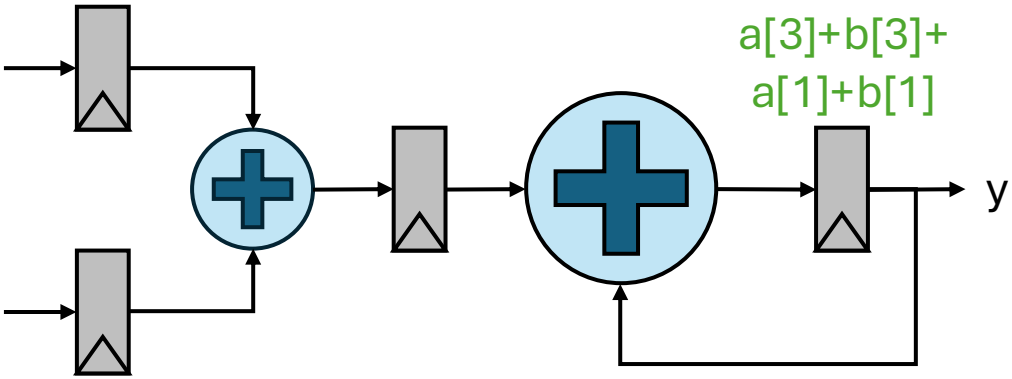
a[1]+b[1] ✗

a[2]+b[2]+a[0]+b[0] ✗

a[3]+b[3]+a[1]+b[1] ✗

Pipelining Feedback Paths

- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:

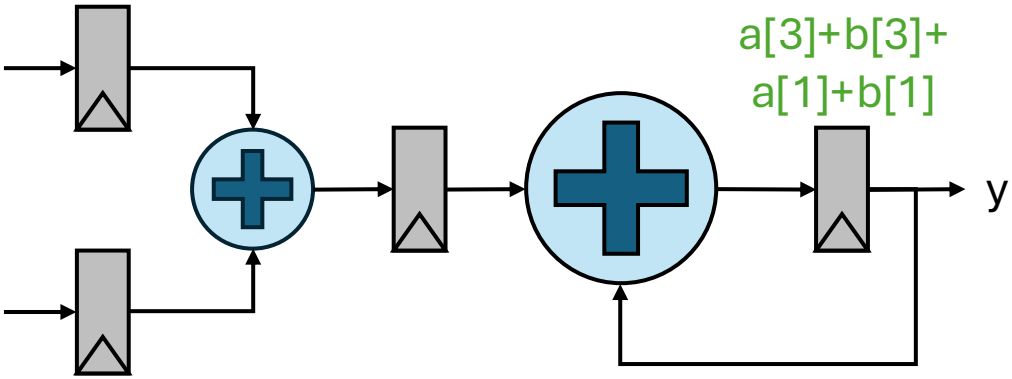


Outputs: a[0]+b[0] ✓
a[0]+b[0] ✓
a[1]+b[1]+a[0]+b[0] ✓
a[1]+b[1]+a[0]+b[0] ✓

To make this circuit function correctly, provide new inputs every other cycle & 0s in-between

Pipelining Feedback Paths

- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



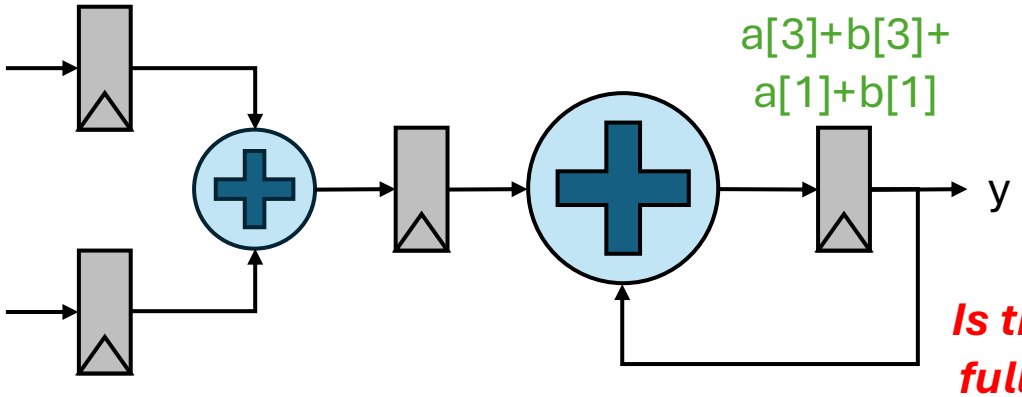
Outputs: a[0]+b[0] ✓
a[0]+b[0] ✓
a[1]+b[1]+a[0]+b[0] ✓
a[1]+b[1]+a[0]+b[0] ✓

To make this circuit function correctly, provide new inputs every other cycle & 0s in-between

Defeats purpose of pipelining!

Pipelining Feedback Paths

- Two-cycle adder does not work in this circuit
  - Feedback is produced delayed by one cycle
- Example:



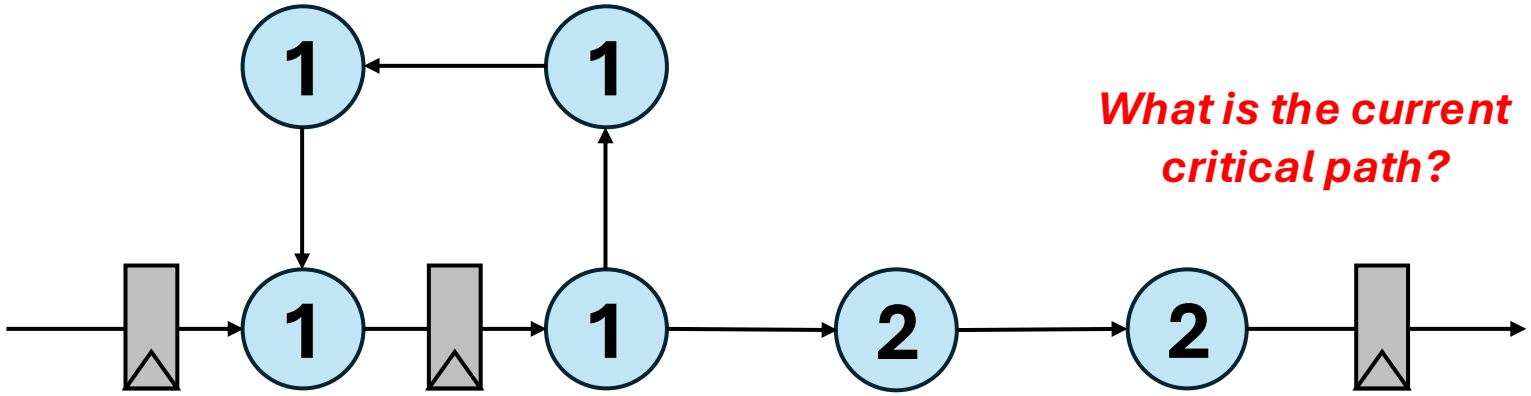
Outputs: a[0]+b[0] ✔
a[0]+b[0] ✔
a[1]+b[1]+a[0]+b[0] ✔
a[1]+b[1]+a[0]+b[0] ✔

Is there a way to keep full throughput while maintaining correct functionality?

To make this circuit function correctly, provide new inputs every other cycle & 0s in-between
Defeats purpose of pipelining!

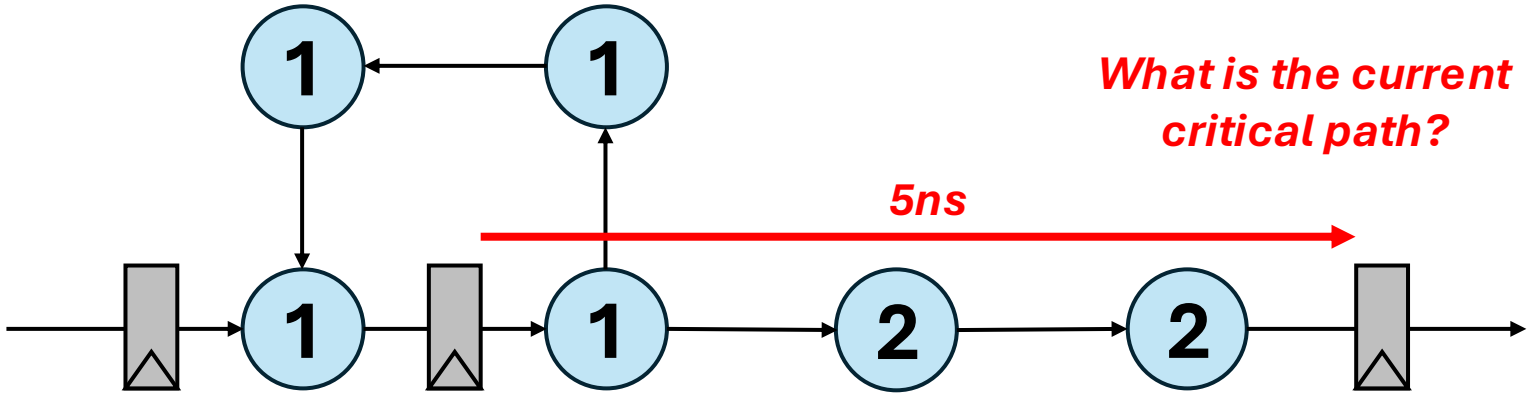
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds



C-Slow Pipelining Example

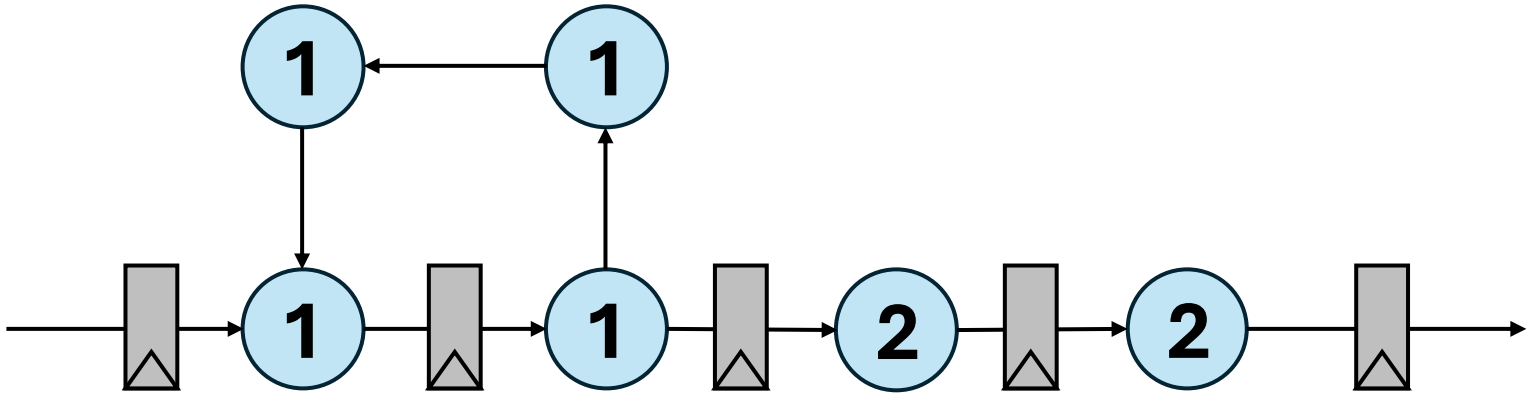
- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds



Throughput = 6 ops x 200 MHz = 1200 MOPS

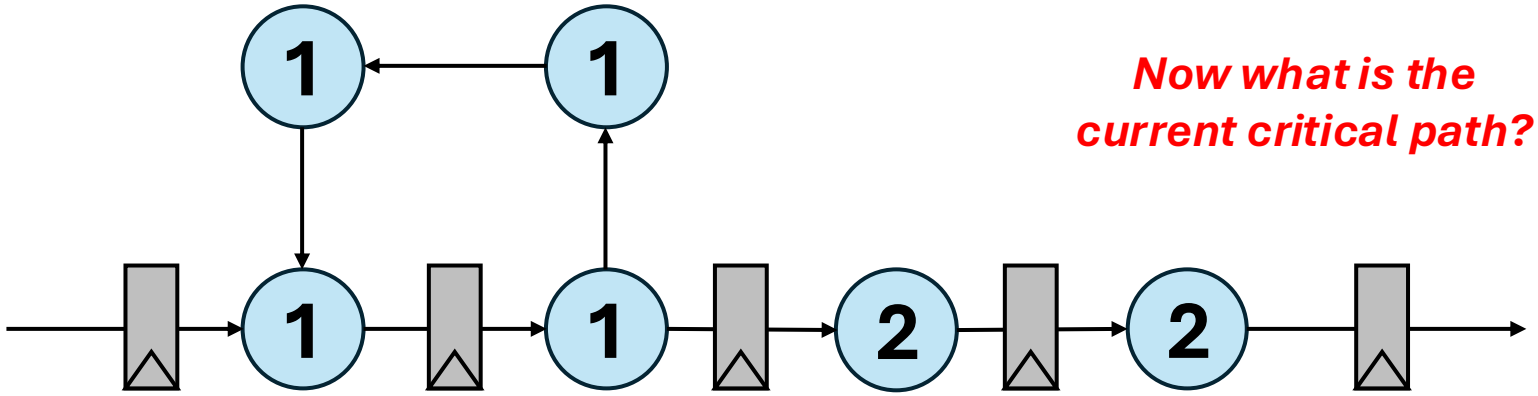
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path



C-Slow Pipelining Example

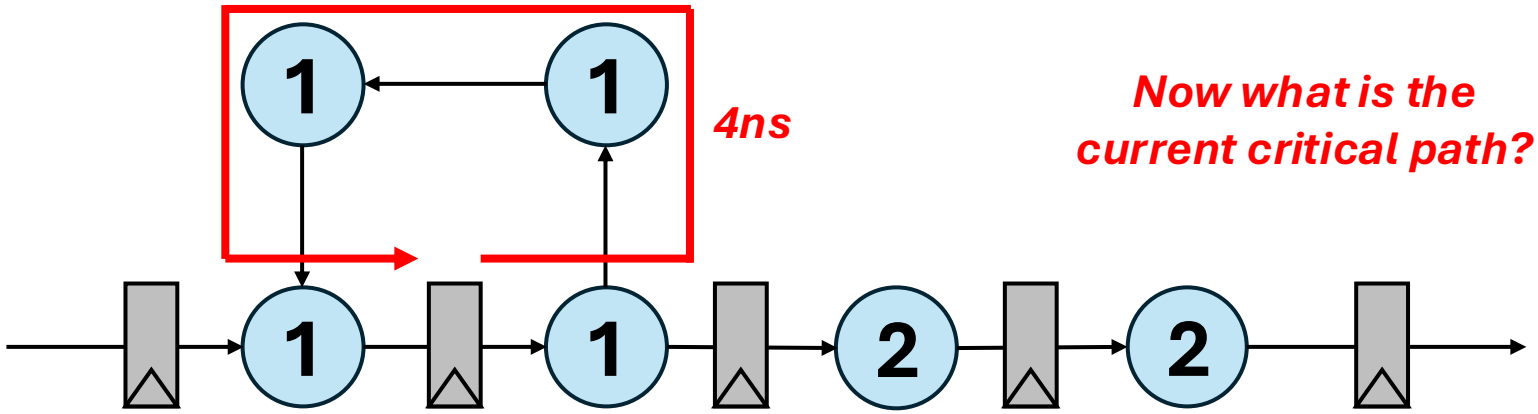
- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path



Now what is the current critical path?

C-Slow Pipelining Example

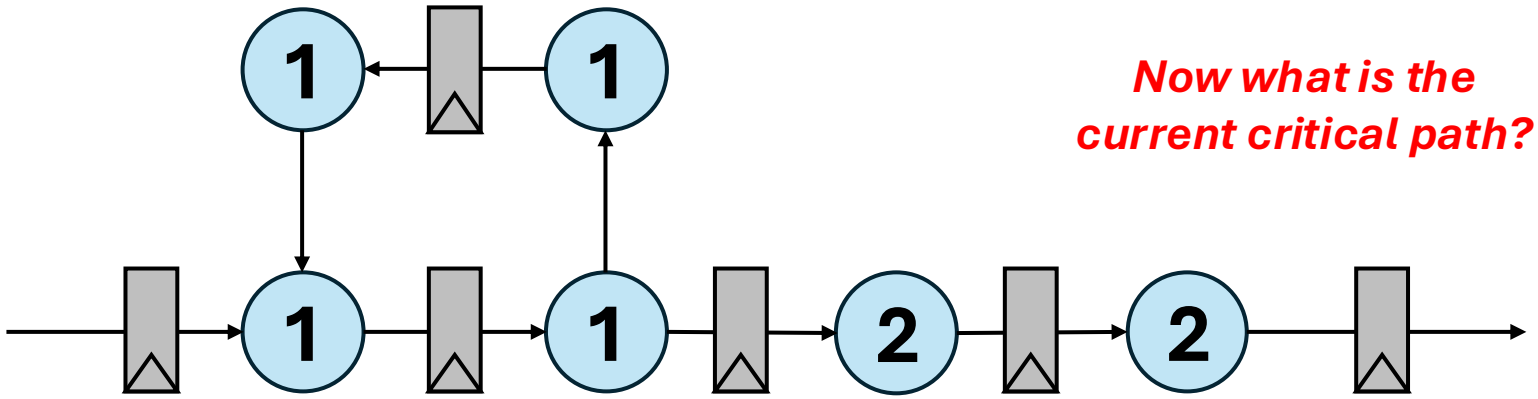
- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path



Throughput = 6 ops x 250 MHz = 1500 MOPS

C-Slow Pipelining Example

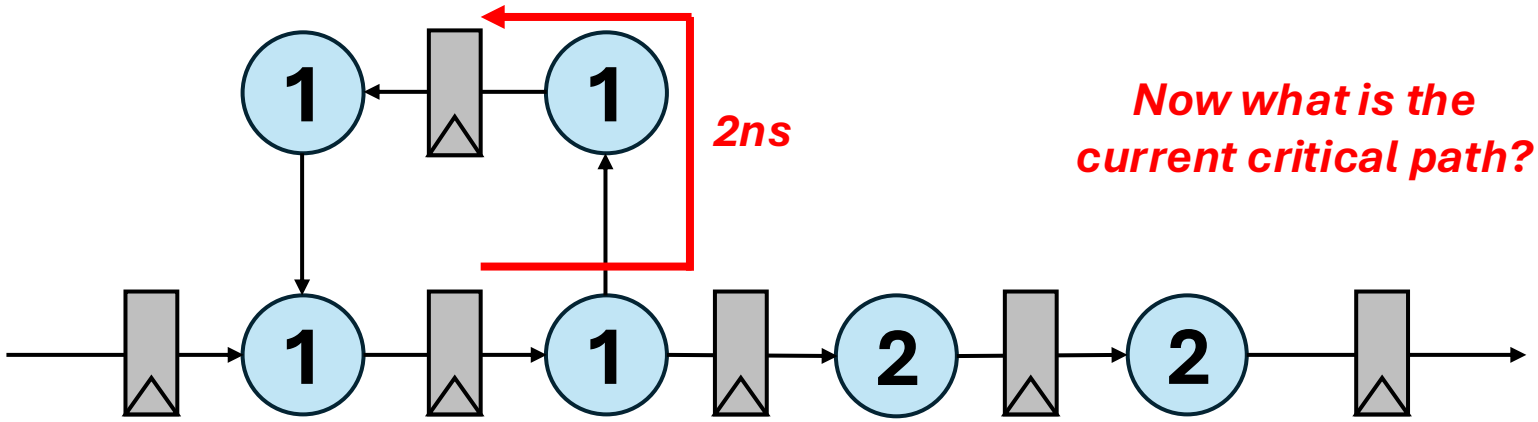
- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency



Now what is the current critical path?

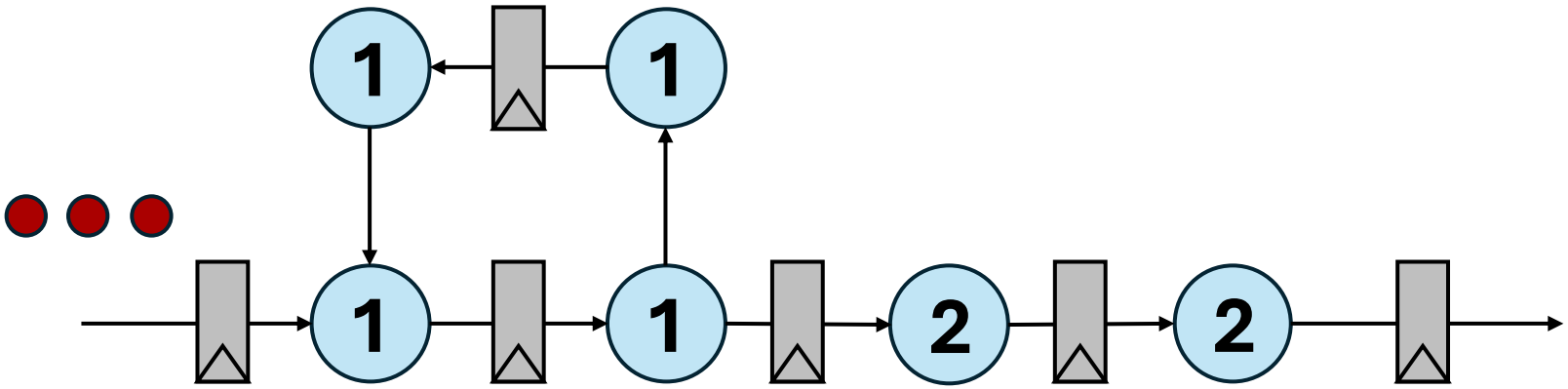
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency



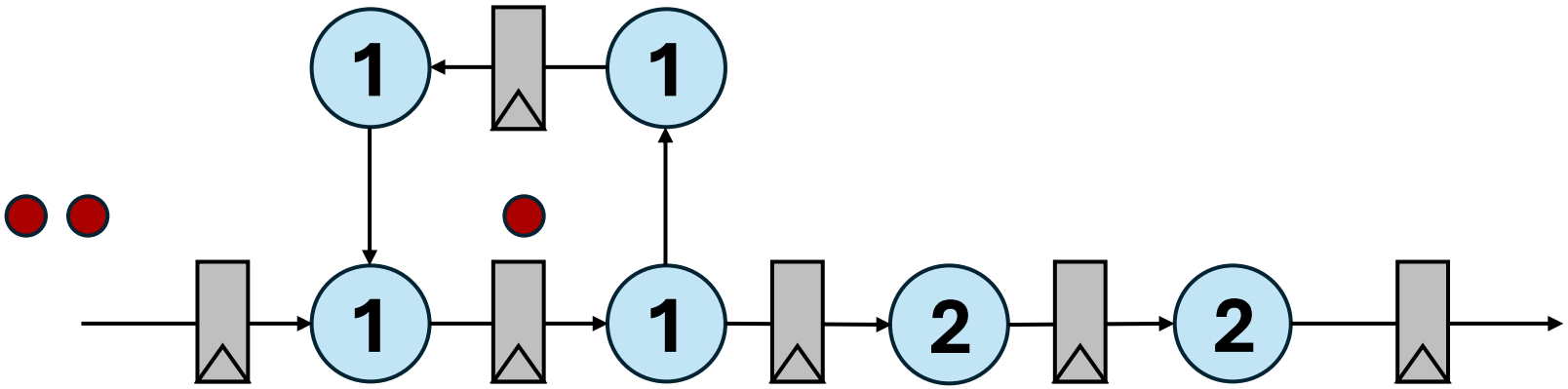
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



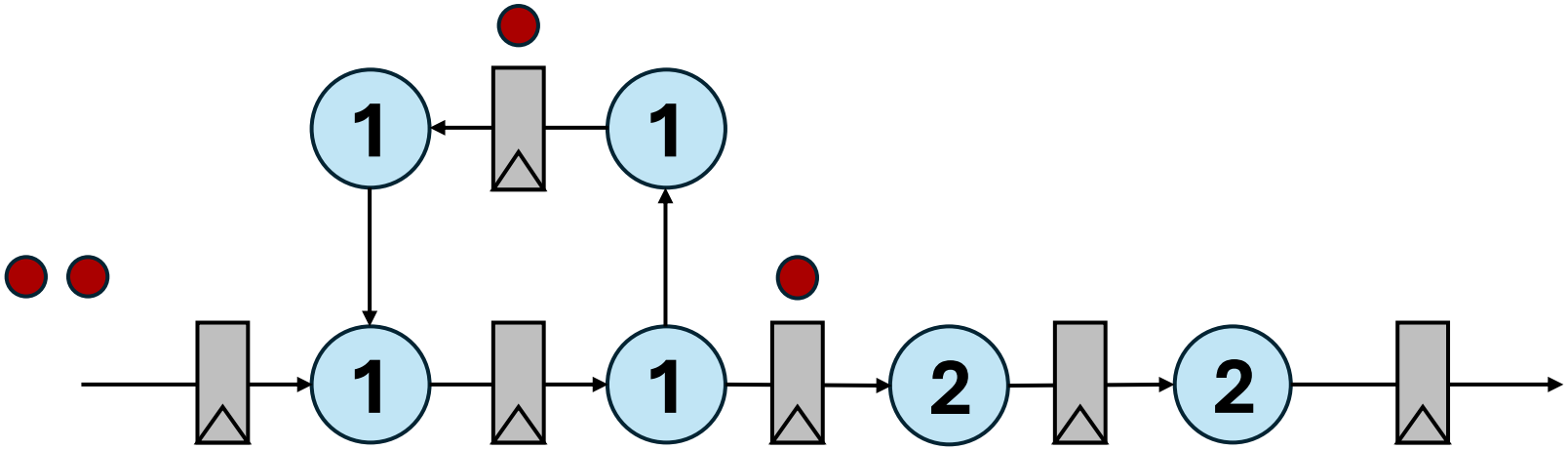
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



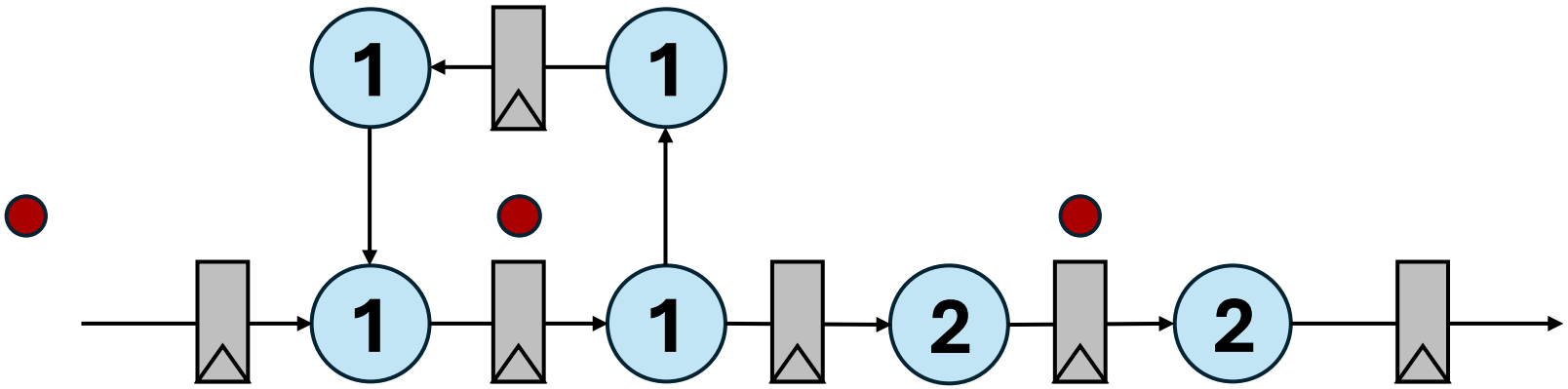
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



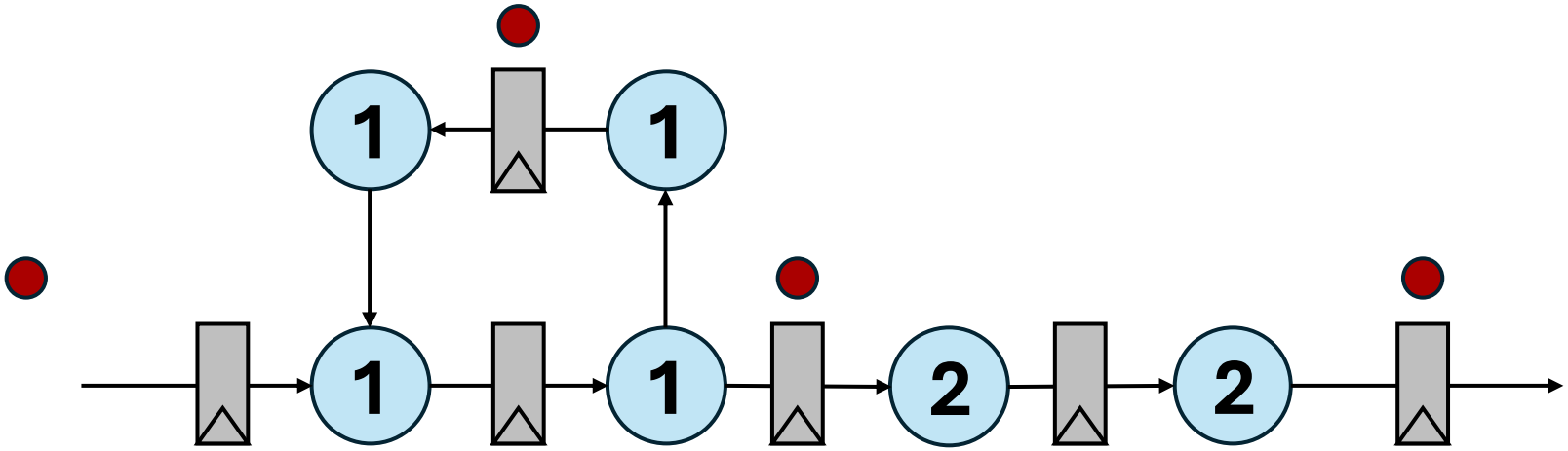
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



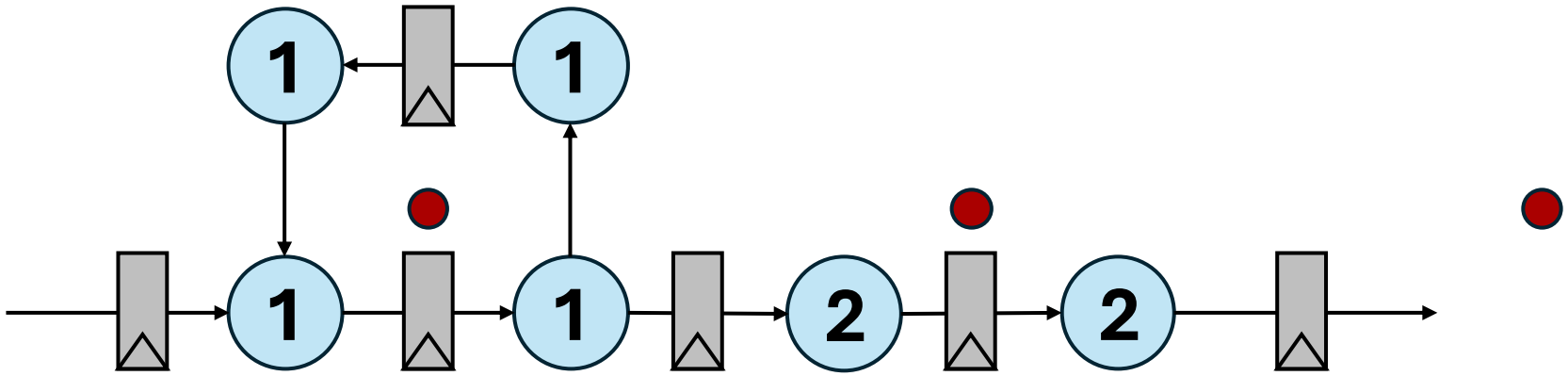
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



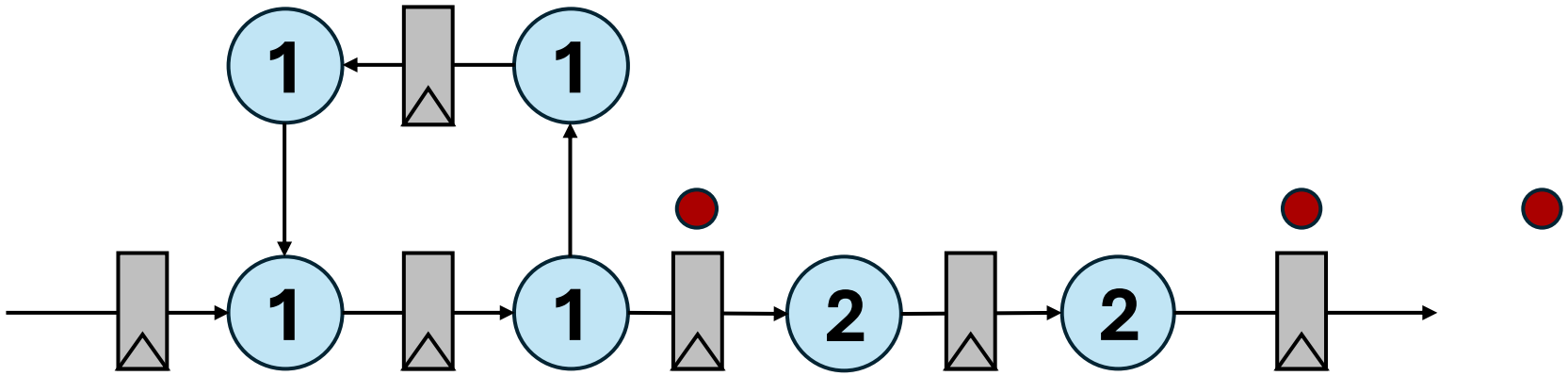
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



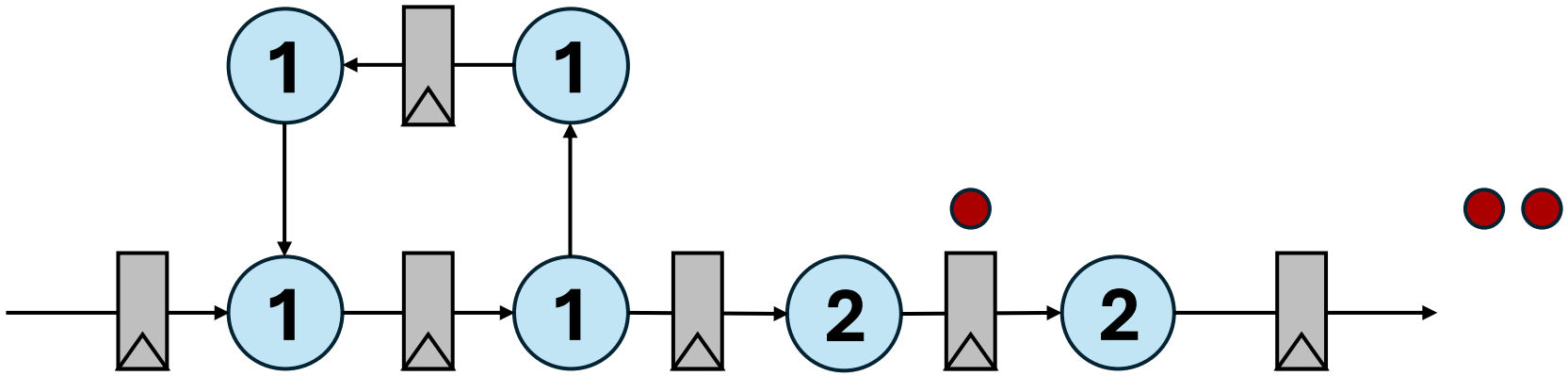
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



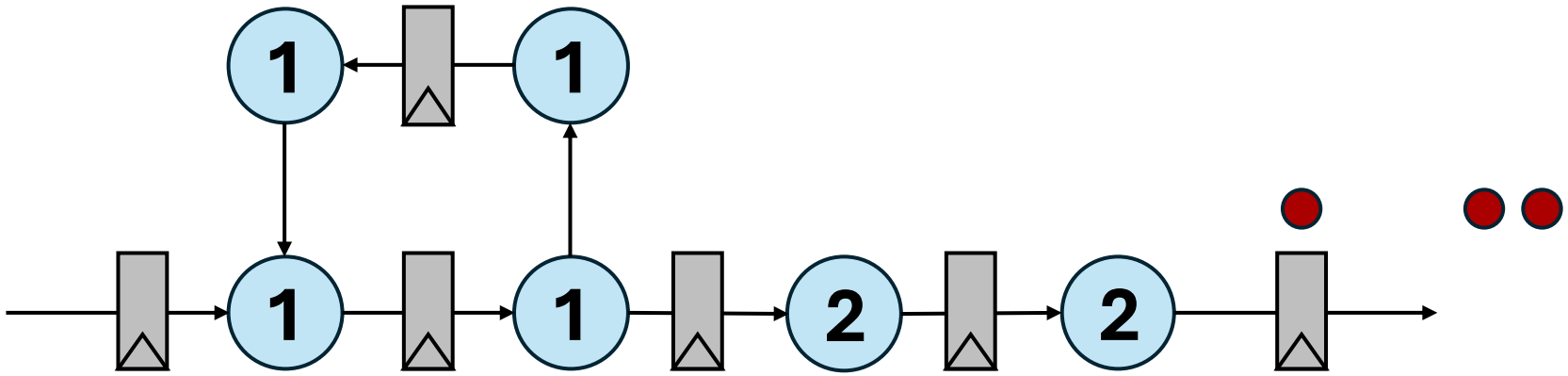
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



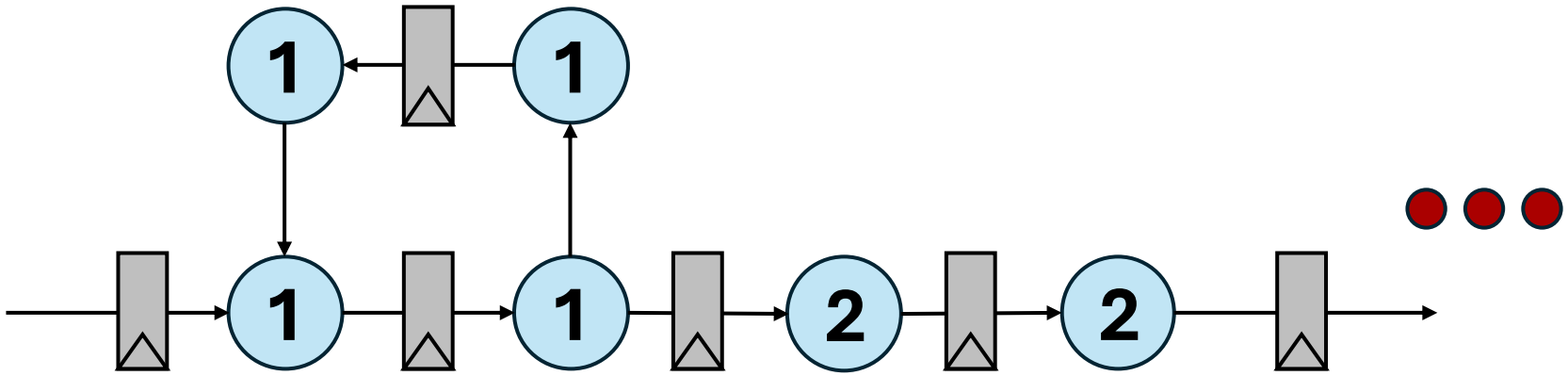
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



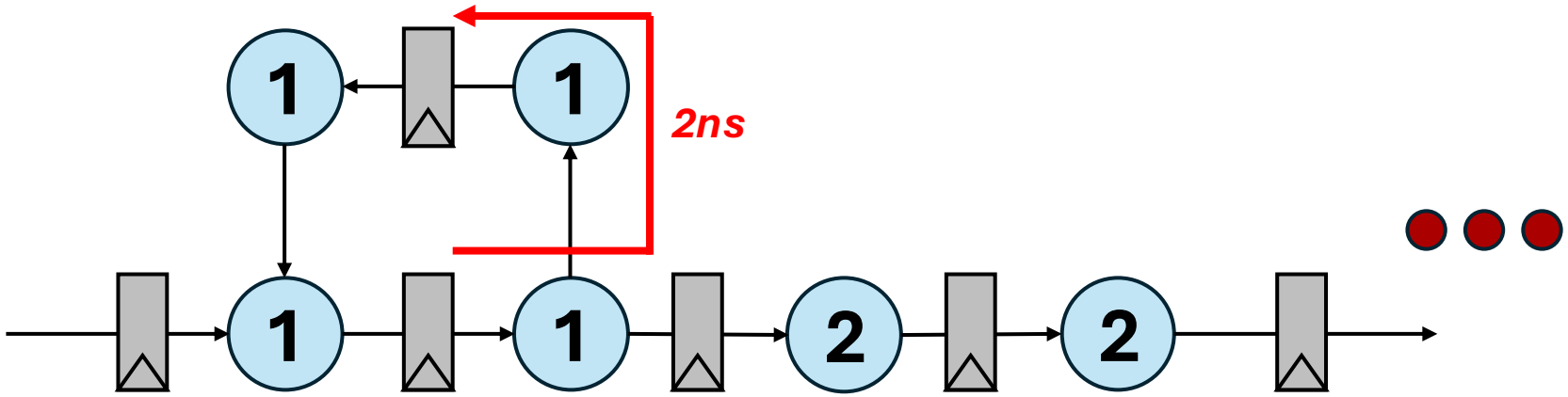
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



C-Slow Pipelining Example

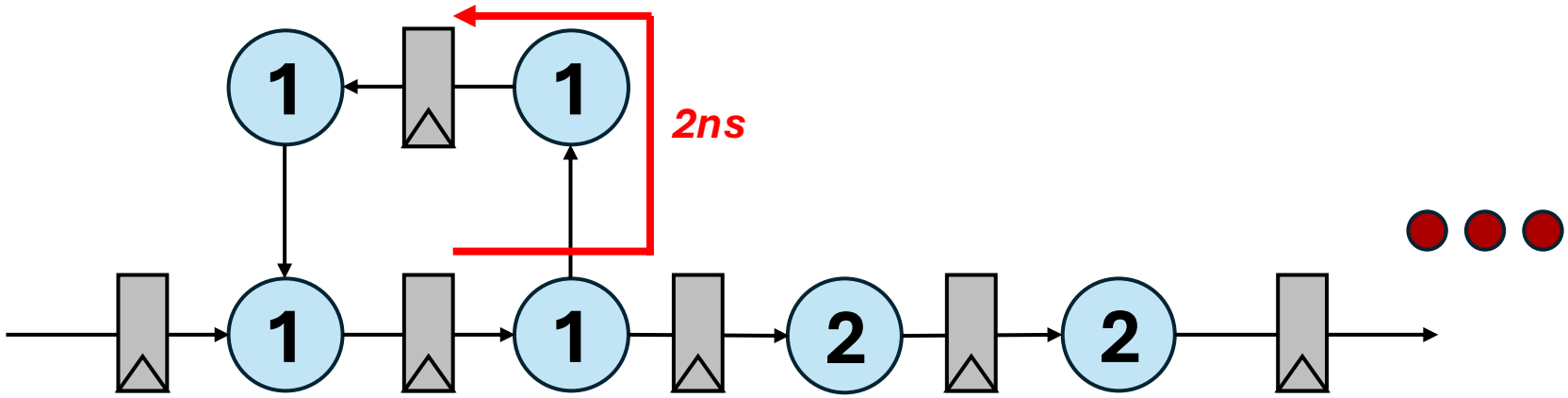
- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**



Throughput =

C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle**

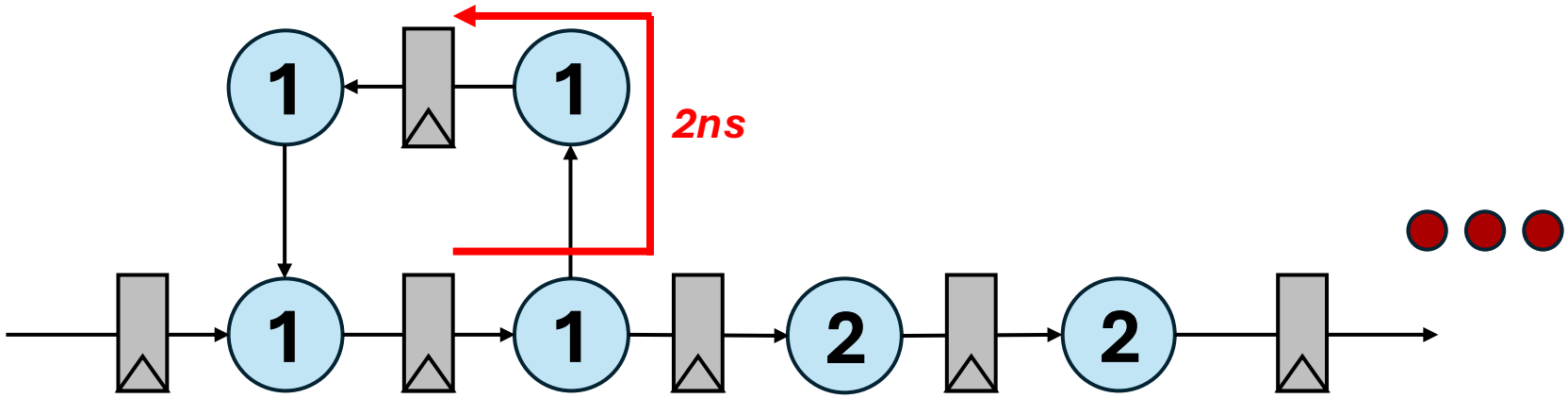


Throughput = 3 ops x 500 MHz = 1500 MOPS

(every other operation is idle!)

C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input every cycle → No improvement**

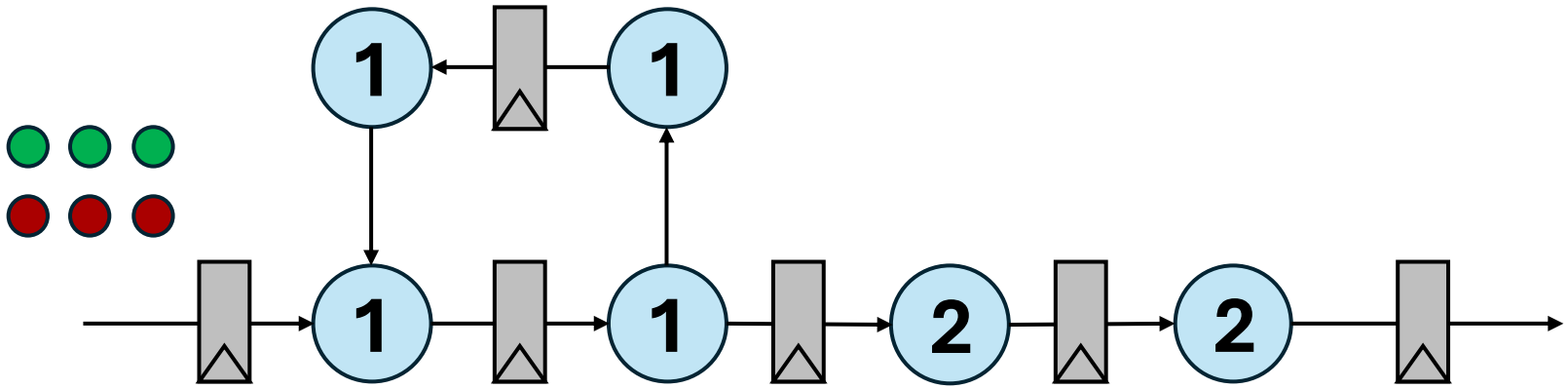


Throughput = 3 ops x 500 MHz = 1500 MOPS

(every other operation is idle!)

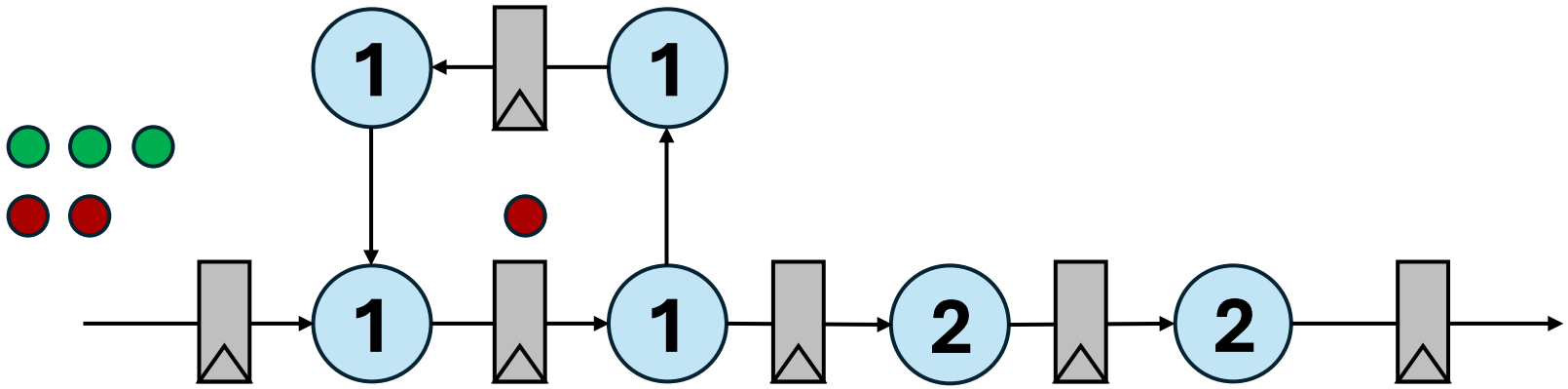
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



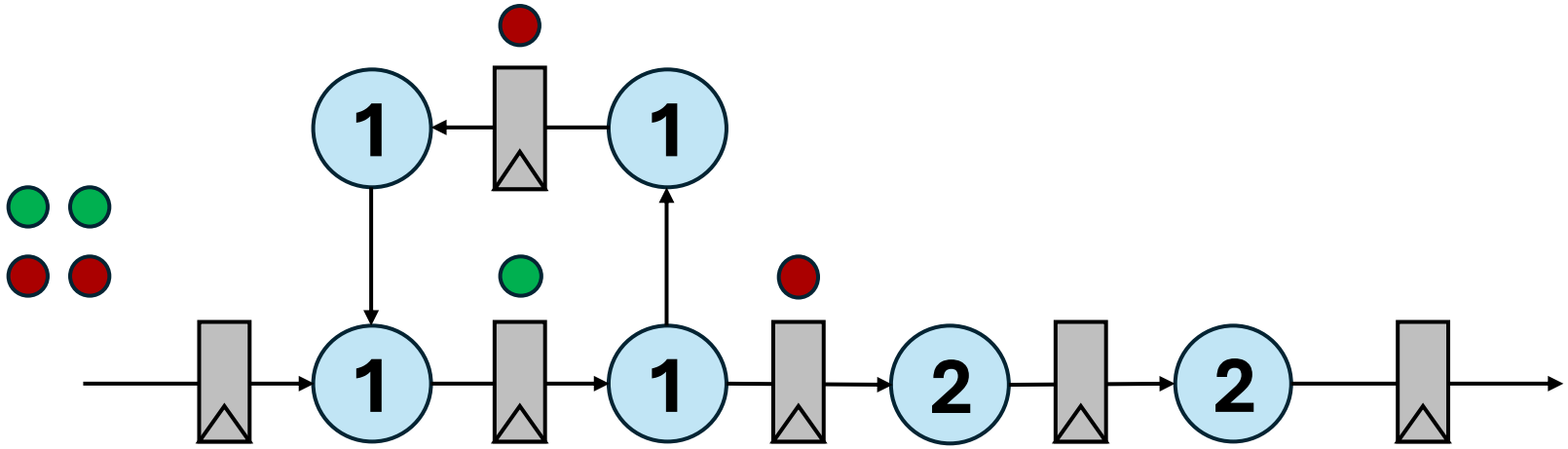
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



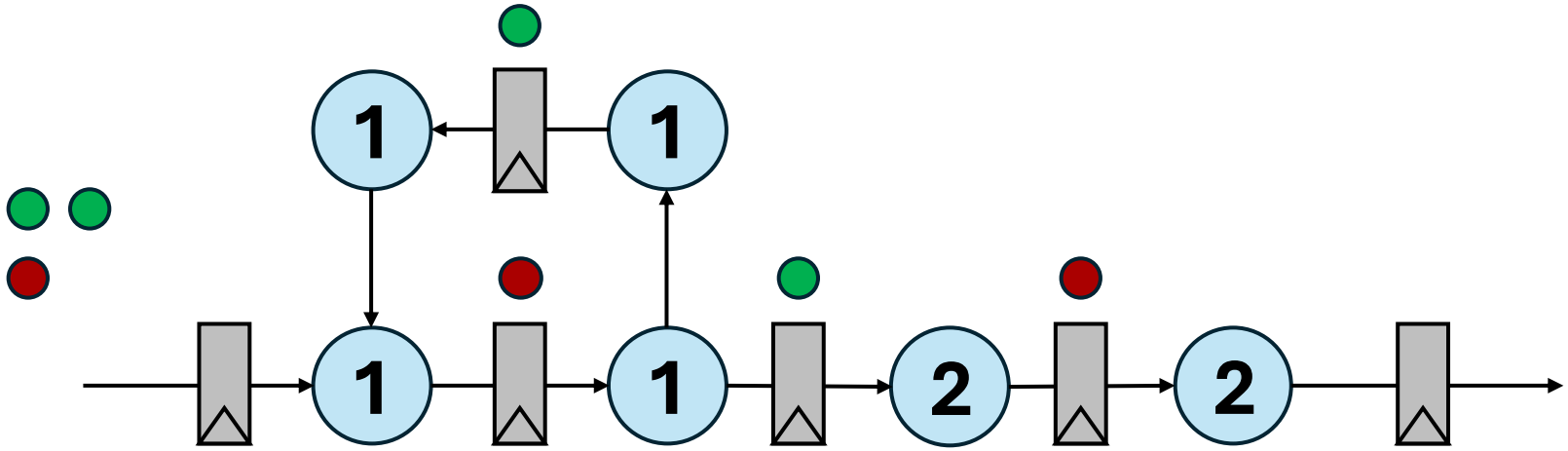
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



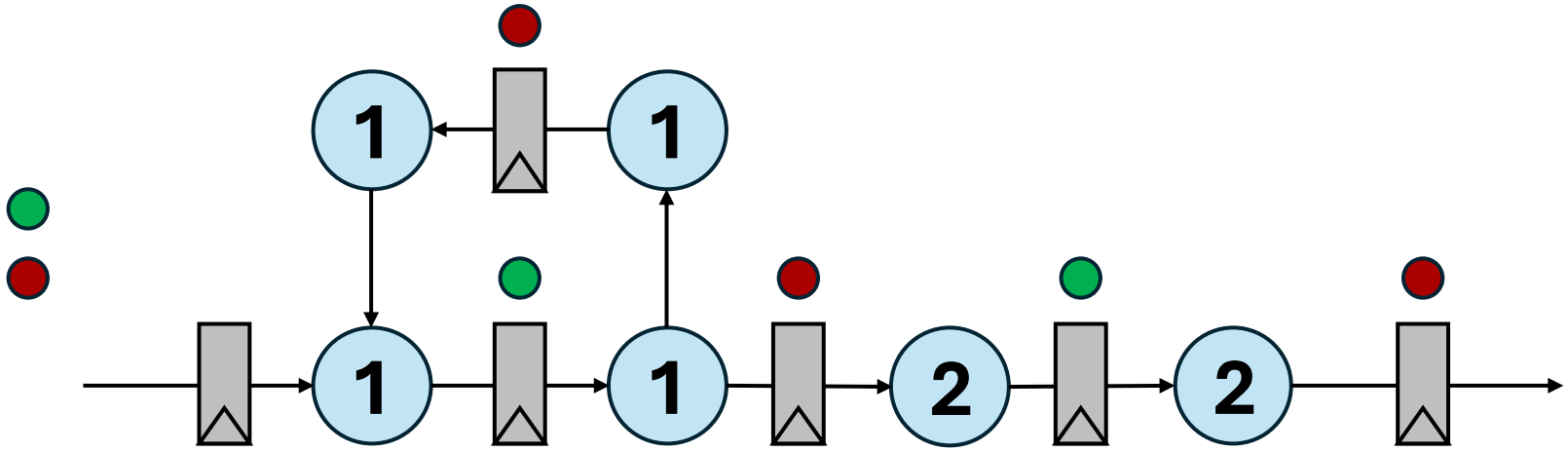
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



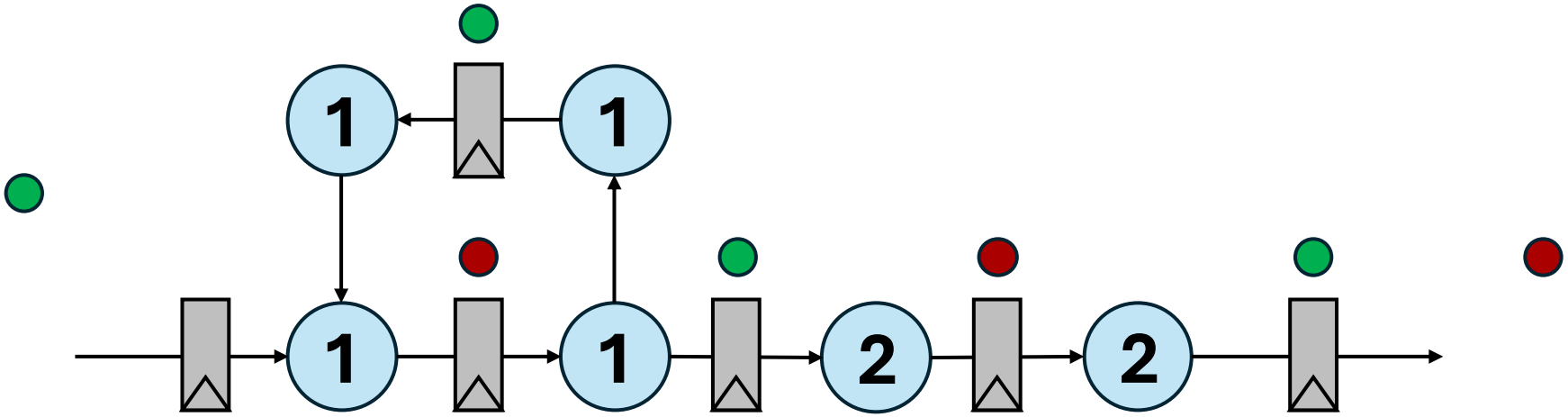
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



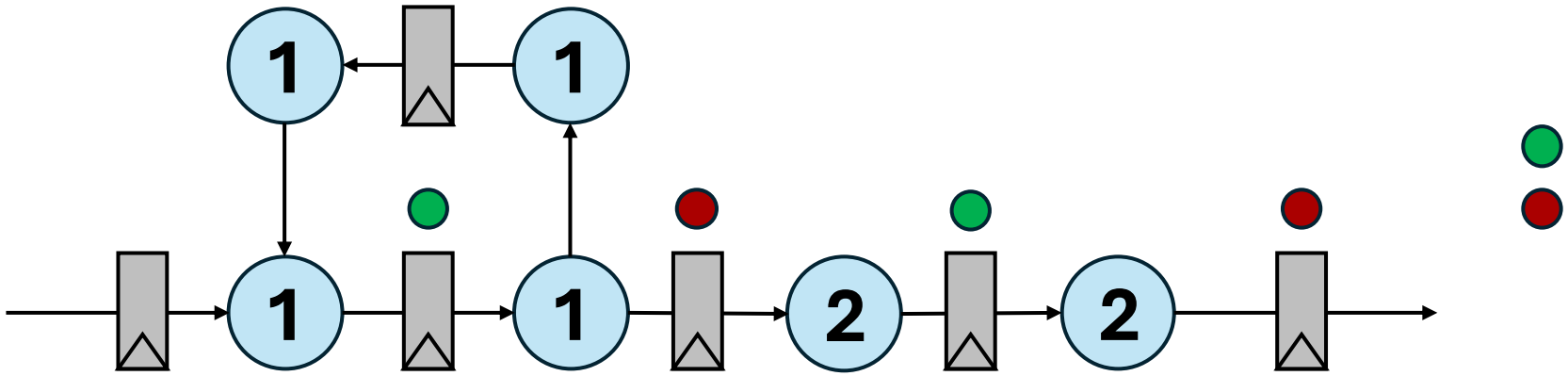
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



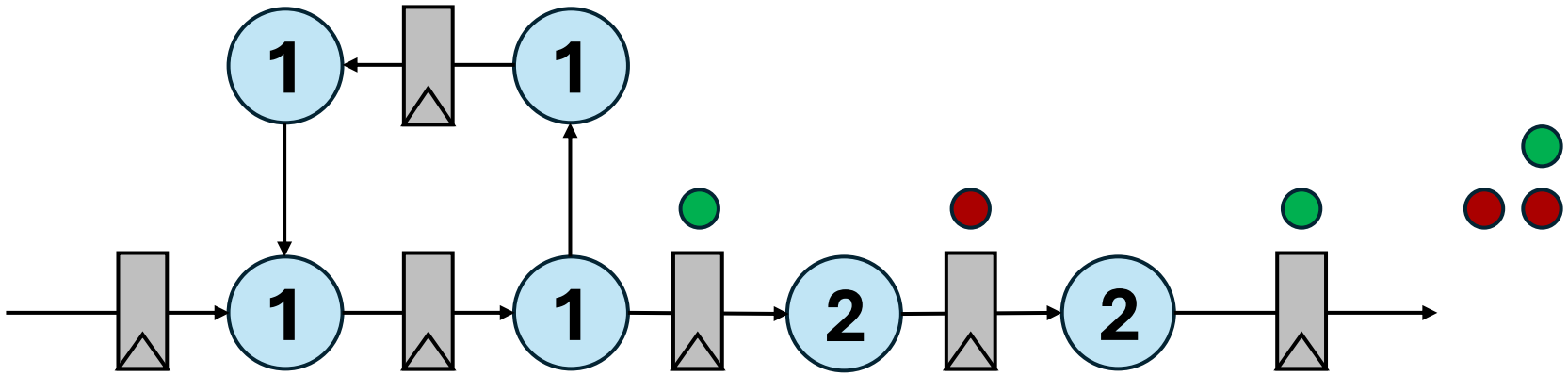
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



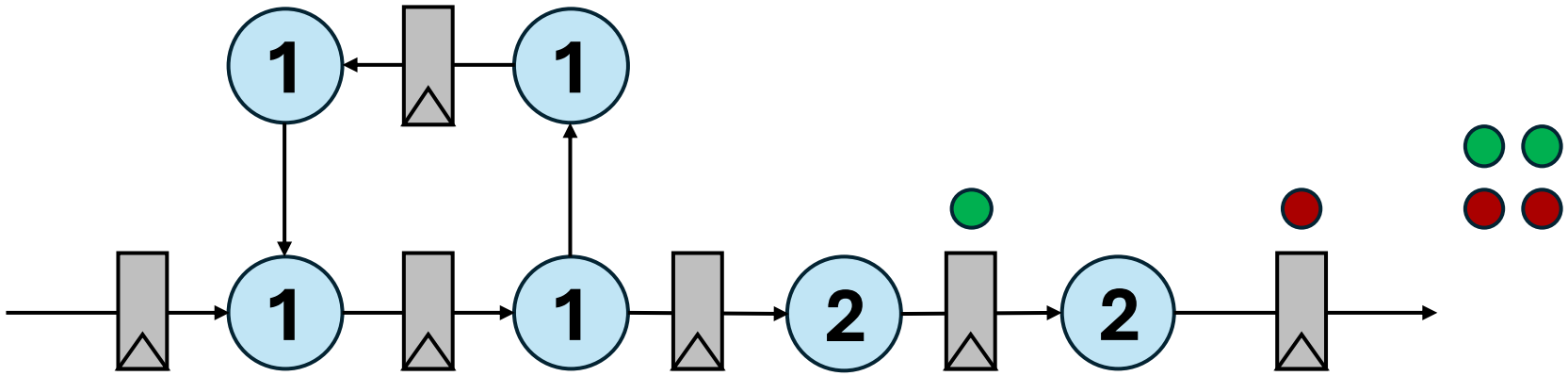
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



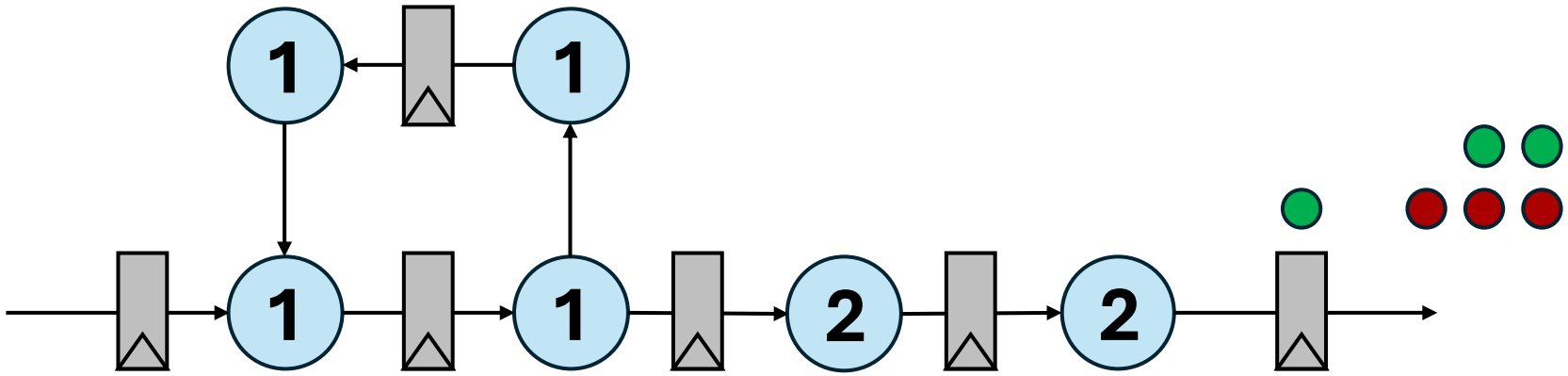
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



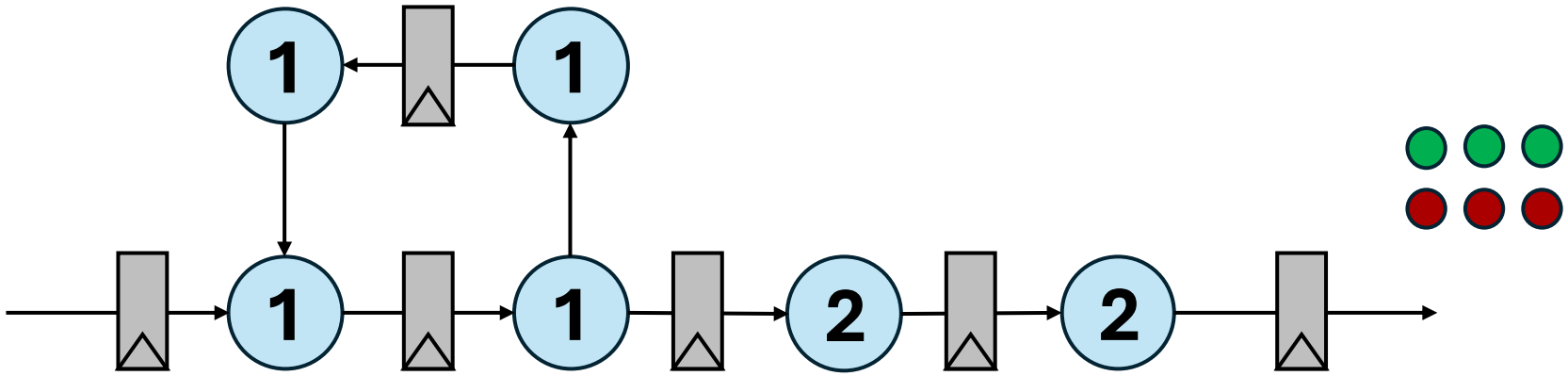
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



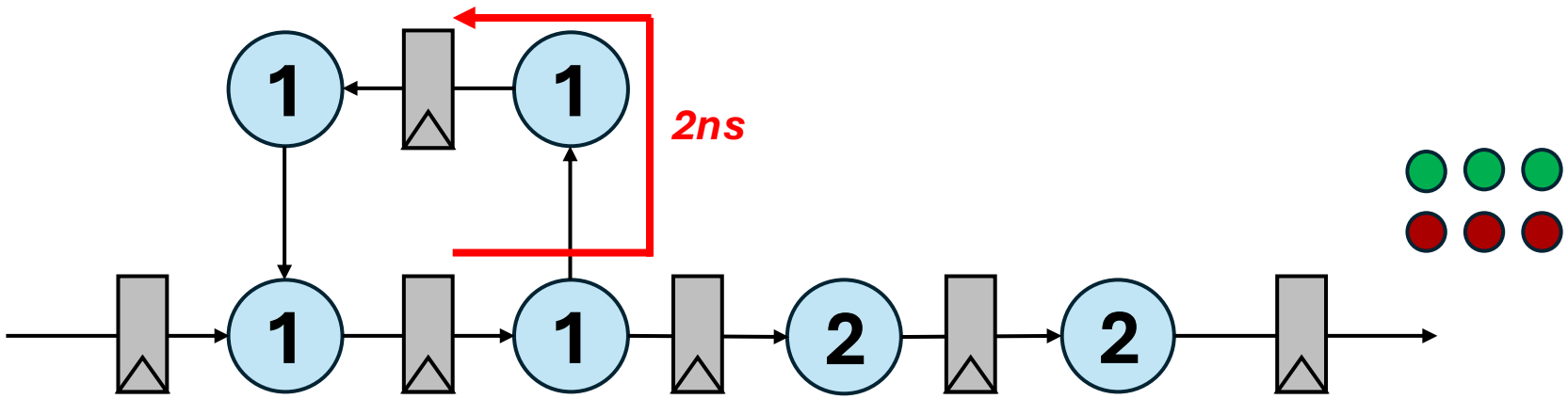
C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



C-Slow Pipelining Example

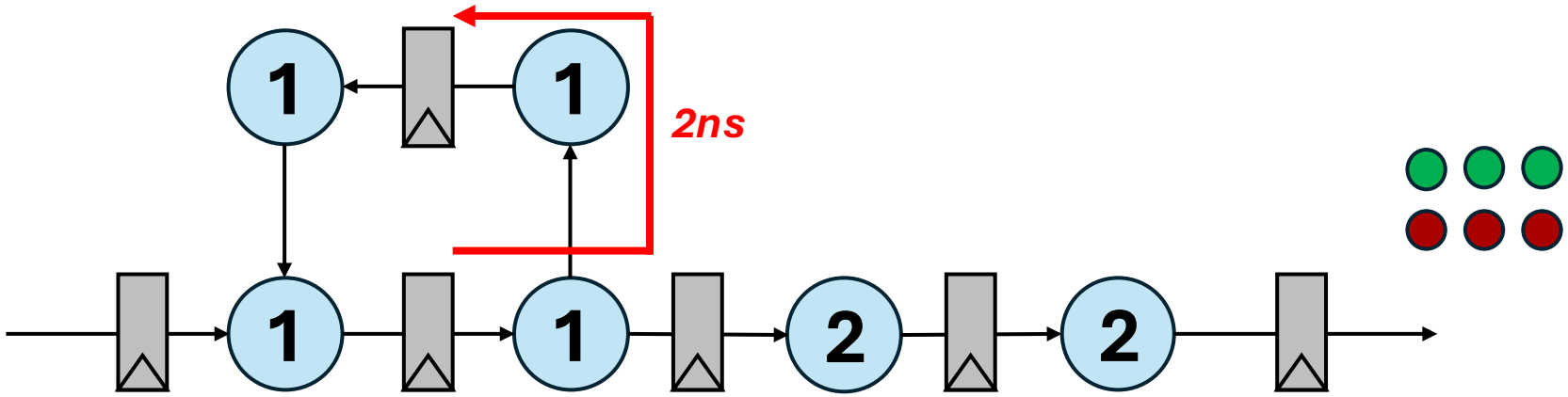
- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



Throughput =

C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**

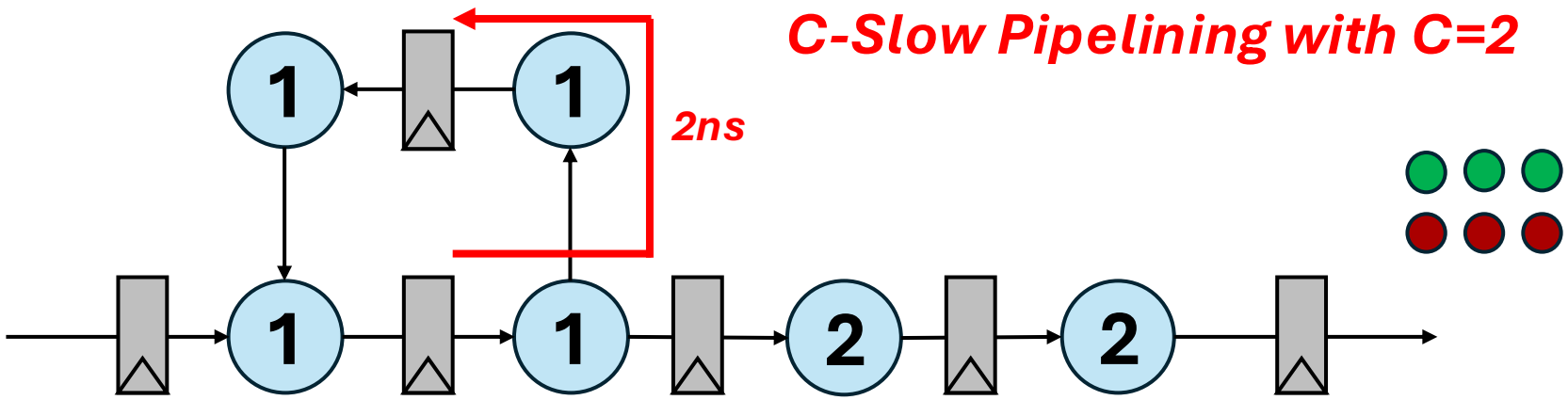


Throughput = 6 ops x 500 MHz = 3000 MOPS

(no idle operations)

C-Slow Pipelining Example

- Maximize the throughput of the circuit below via pipelining …
  - Each circle is an operation and the number in it represents the operation delay in nanoseconds
- First, pipeline the feedforward path to cut the critical path
- Pipelining the feedback loop improves frequency **but can no longer feed a new input of the same stream every cycle**



Throughput = 6 ops x 500 MHz = 3000 MOPS

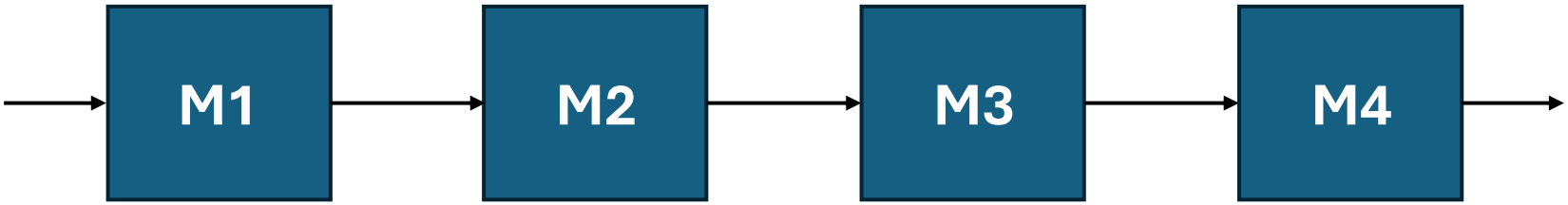
(no idle operations)

Throughput Balancing

- Pipelining can improve the throughput of a module

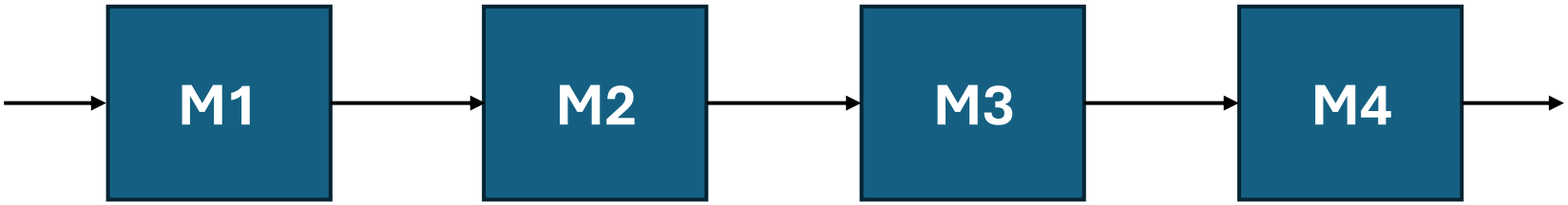
Throughput Balancing

- Pipelining can improve the throughput of a module
- Many designs in practice have several cascaded modules processing data that flows from one module to the next → streaming architectures



Throughput Balancing

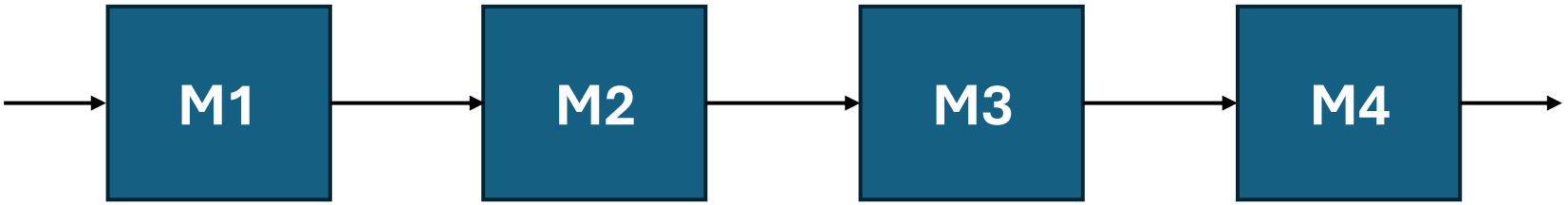
- Pipelining can improve the throughput of a module
- Many designs in practice have several cascaded modules processing data that flows from one module to the next → streaming architectures



Throughput of the entire structure = Lowest throughput of all stages

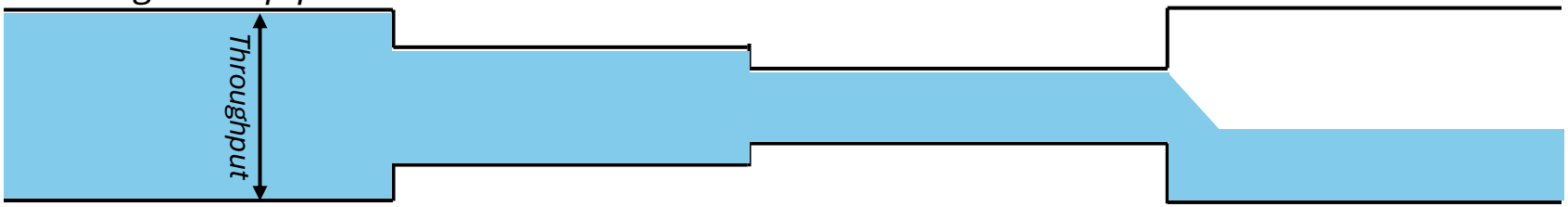
Throughput Balancing

- Pipelining can improve the throughput of a module
- Many designs in practice have several cascaded modules processing data that flows from one module to the next → streaming architectures



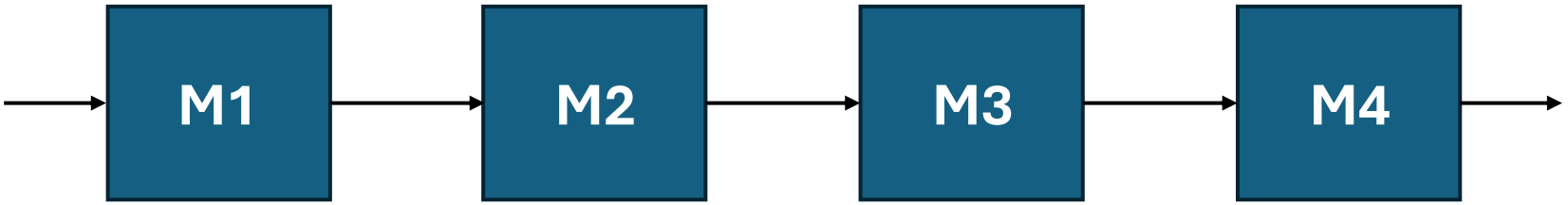
Throughput of the entire structure = Lowest throughput of all stages

Analogous to pipes

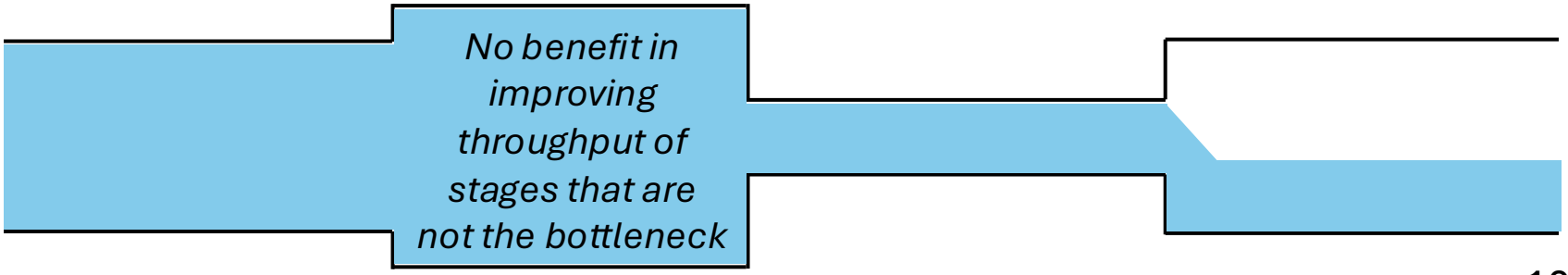


Throughput Balancing

- Pipelining can improve the throughput of a module
- Many designs in practice have several cascaded modules processing data that flows from one module to the next → streaming architectures

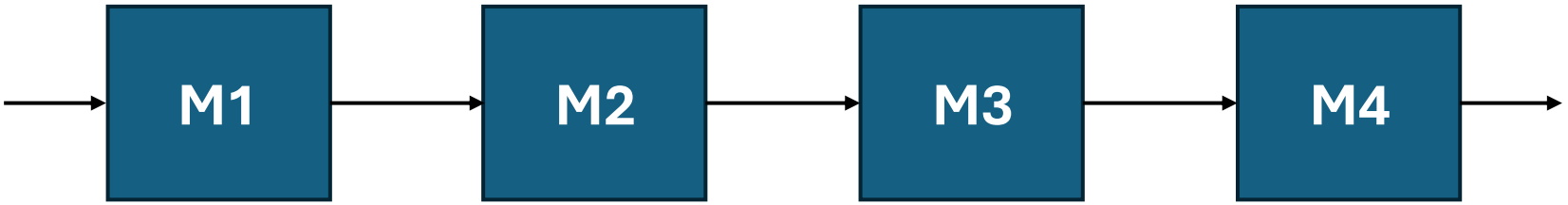


Throughput of the entire structure = Lowest throughput of all stages



Throughput Balancing

- Pipelining can improve the throughput of a module
- Many designs in practice have several cascaded modules processing data that flows from one module to the next → streaming architectures

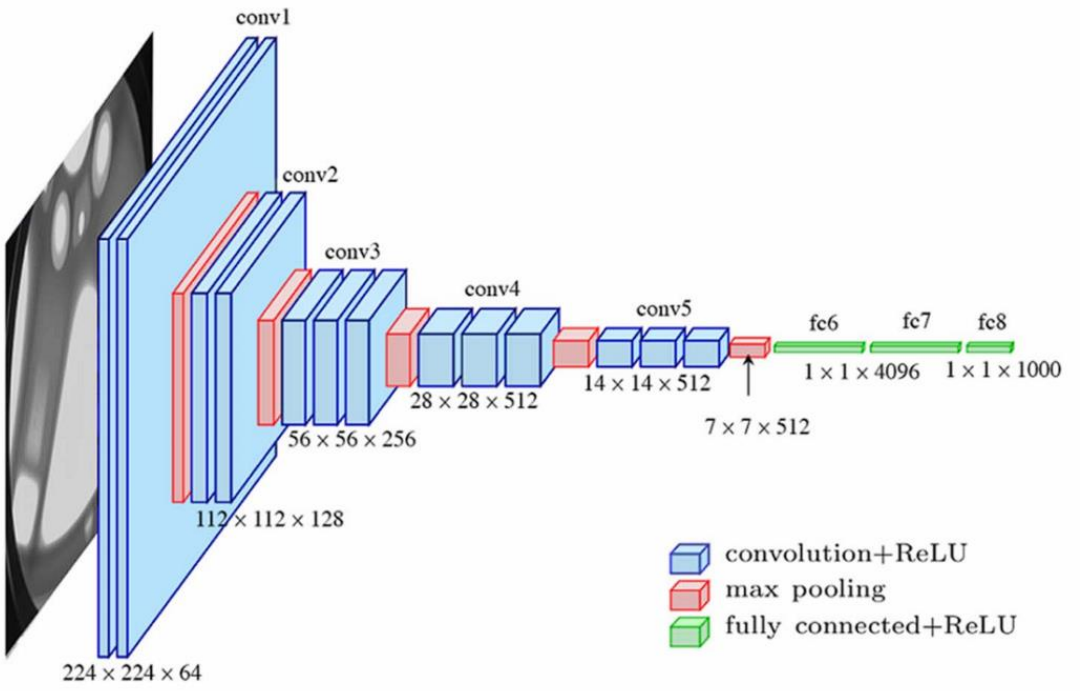


Throughput of the entire structure = Lowest throughput of all stages

Ideally, want throughput of all stages of the structure to be balanced

Streaming Architecture Example: HPIPE

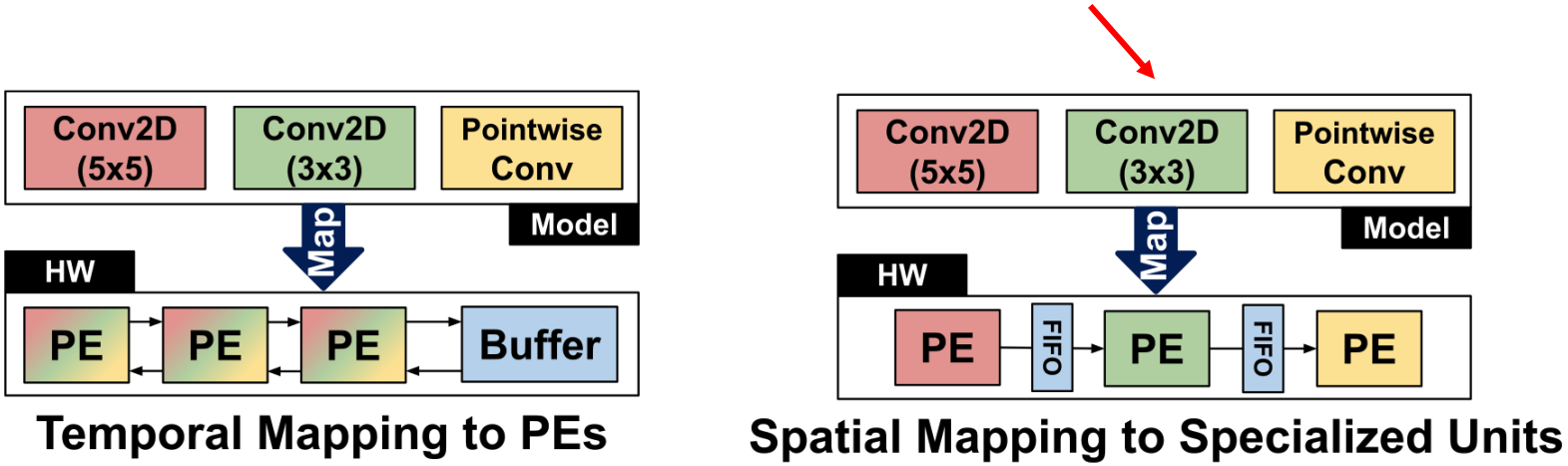
- Convolutional Neural Network (CNN) FPGA-based accelerator
- Many cascaded layers
  - Output of one layer is fed as an input to the next layer



Source: viso.ai

Streaming Architecture Example: HPIPE

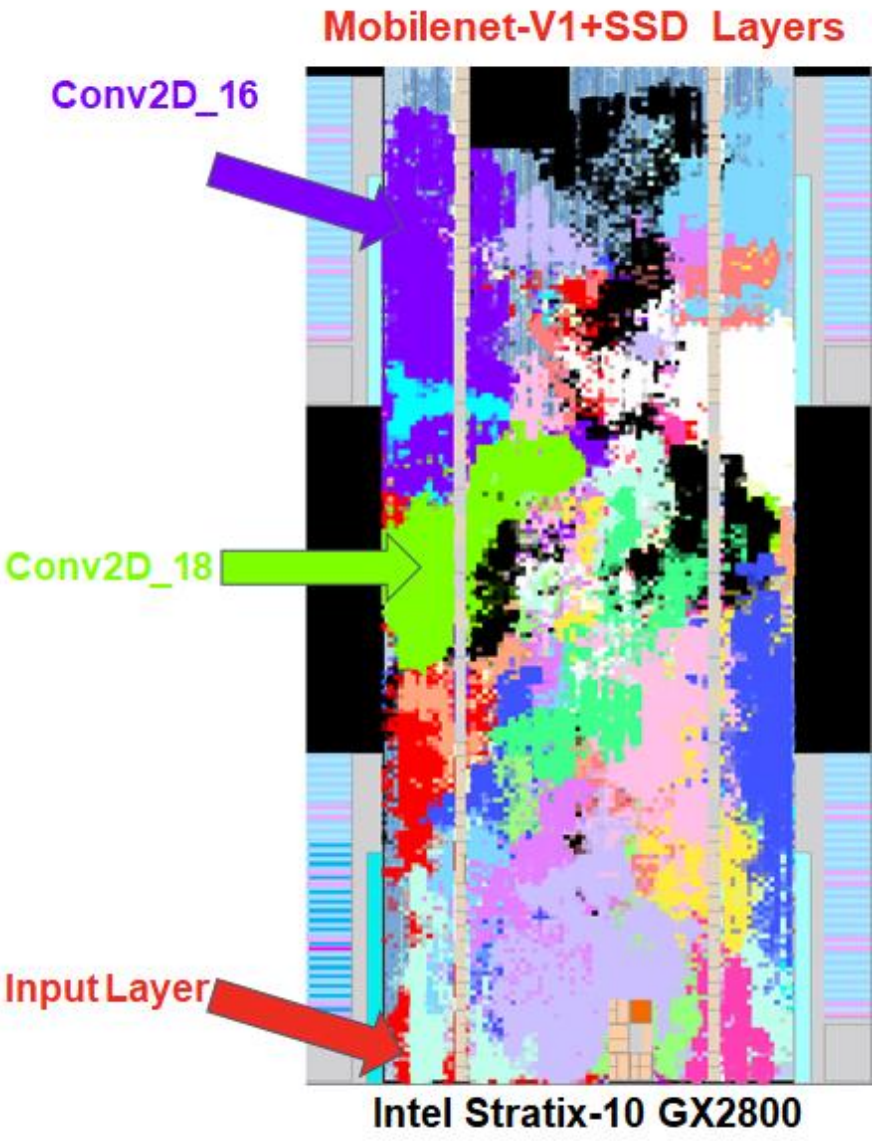
- Convolutional Neural Network (CNN) FPGA-based accelerator
- Many cascaded layers
  - Output of one layer is fed as an input to the next layer
- HPIPE maps each CNN layer to a processing element (PE) and cascades them in a streaming-style architecture



Example from: M.Hall, V. Betz, "HPIPE: Heterogeneous Layer-Pipelined & Sparse-Aware CNN Inference for FPGAs", 2020 [Link]

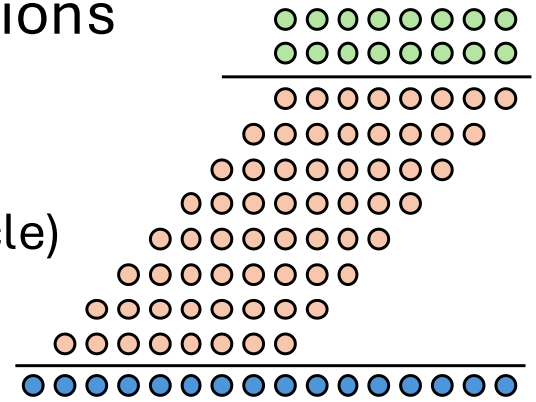
Streaming Architecture Example: HPIPE

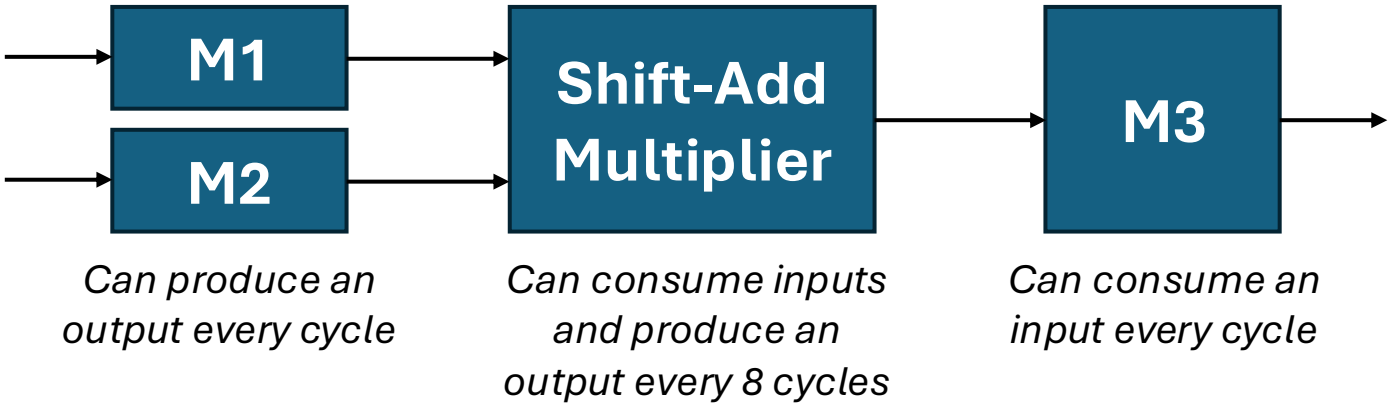
Throughput of the entire accelerator is the throughput of the slowest CNN layer!



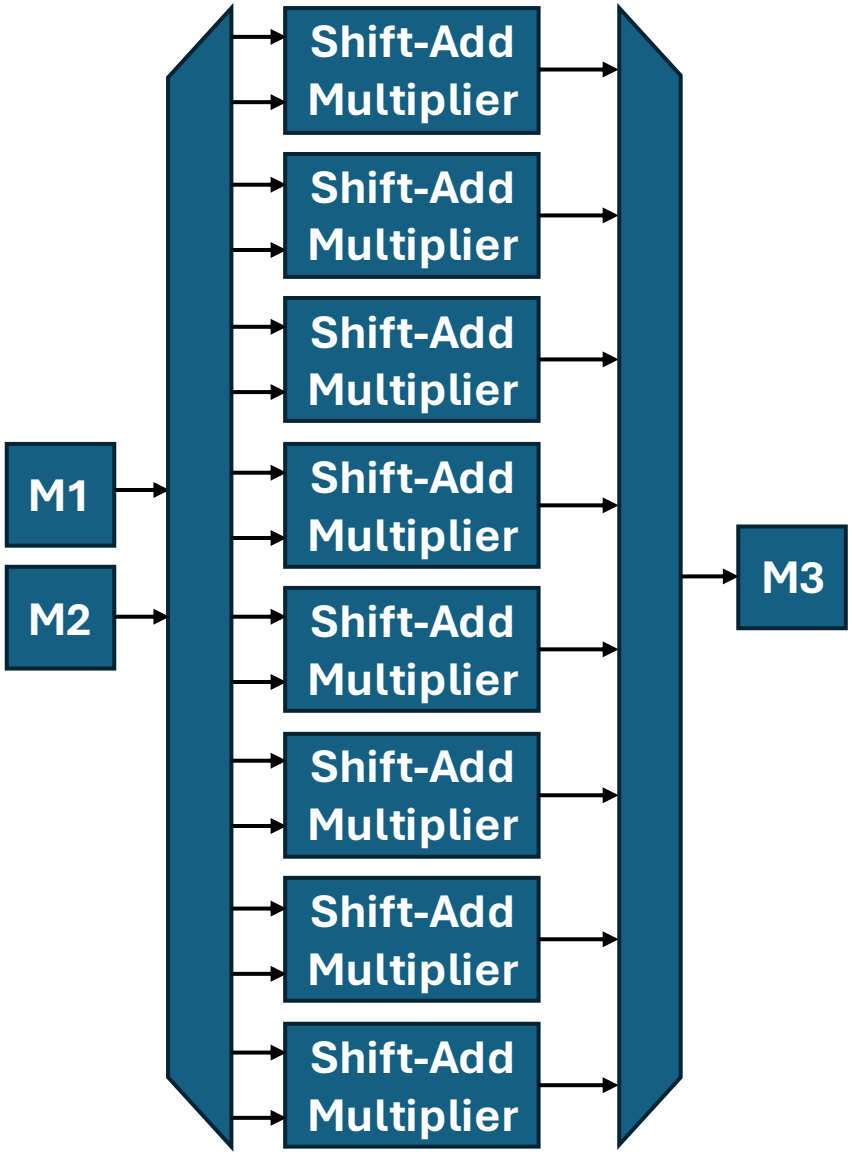
Handling Low-Throughput Components

- Sometimes pipelining is not enough to improve throughput of certain components → Multi-cycle operations
  - Example: shift-add multiply
  - 8b x 8b multiplication takes 8 cycles
  - Throughput = 1/8 x Frequency (1/8 ops per cycle)



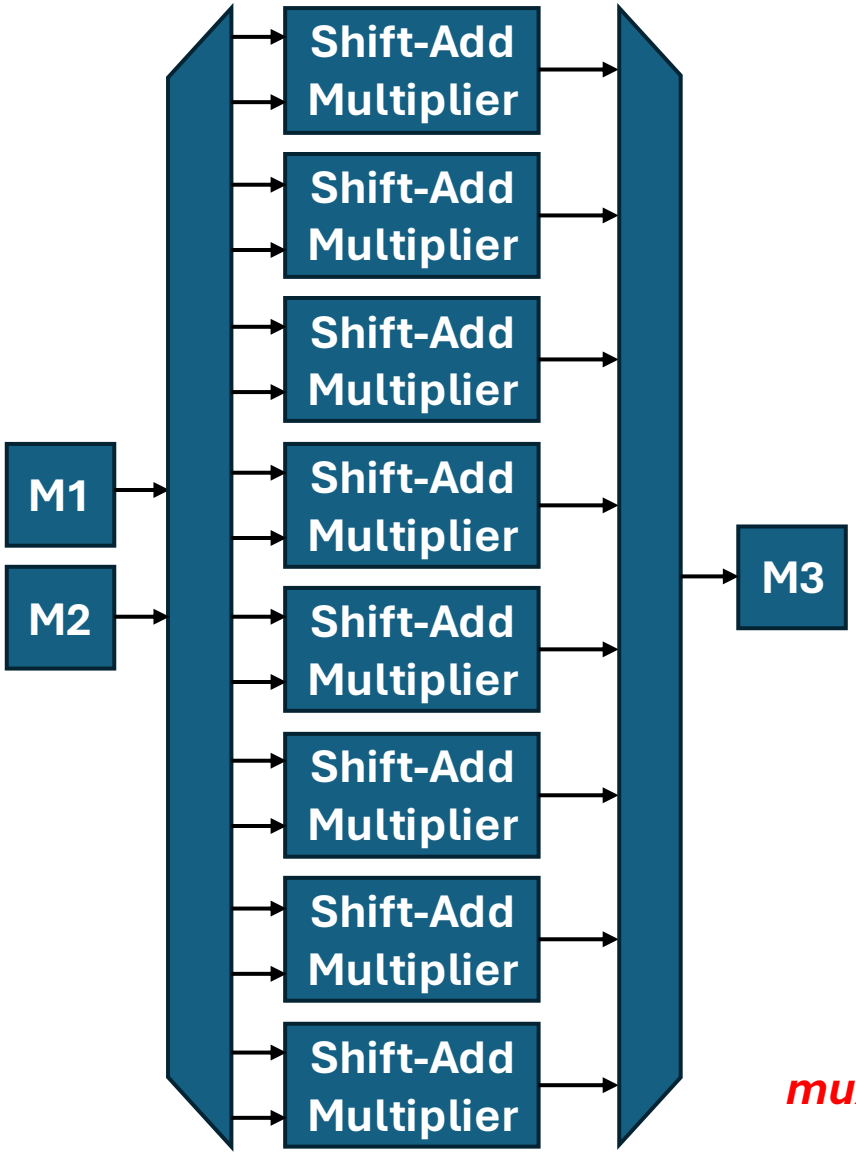


Throughput Balancing by Replication



- Perform multiplication c = a x b on inputs arriving every cycle
- Instantiate 8 copies of the shift-add multiplier module
- Use 1:8 demultiplexer to steer inputs to the eight components
- Use 8:1 multiplexer to collect results from each block

Throughput Balancing by Replication

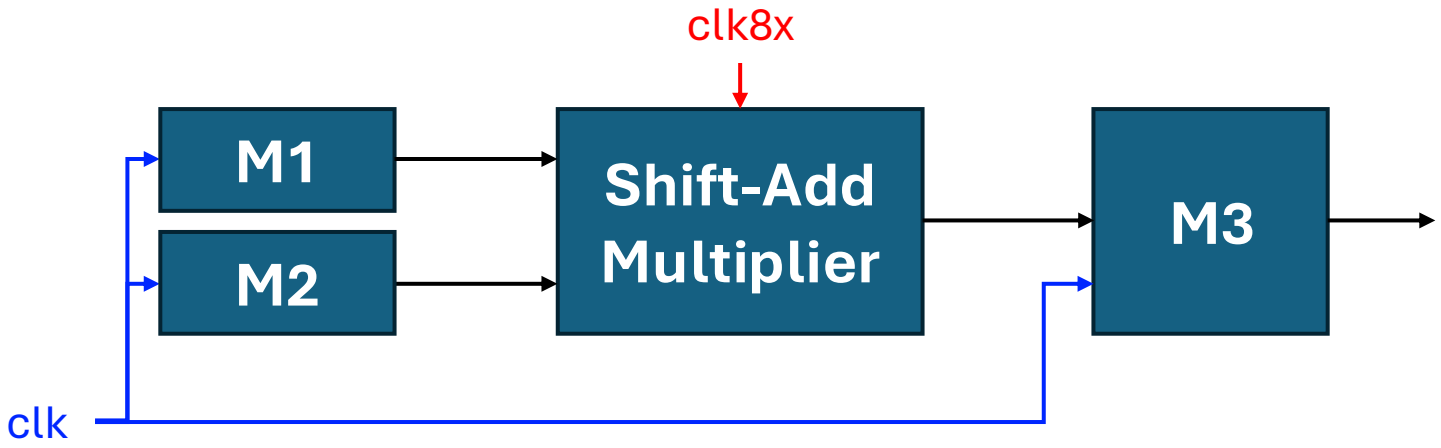


- Perform multiplication c = a x b on inputs arriving every cycle
- Instantiate 8 copies of the shift-add multiplier module
- Use 1:8 demultiplexer to steer inputs to the eight components
- Use 8:1 multiplexer to collect results from each block

What to use as the mux/demux select lines?

Throughput Balancing by Multi-Pumping

- Supply two different clocks (phase aligned, integer multiples)
- Shift-add multiplier runs on a clock that is 8 times as fast as the other modules
  - One cycle for M1, M2, M3 is 8 cycles for shift-add multiplier
- Crossing clock domains is tricky
  - Needs synchronizers or asynchronous FIFOs
  - Out of the scope of this course



Next Lecture

Latency-insensitive design style