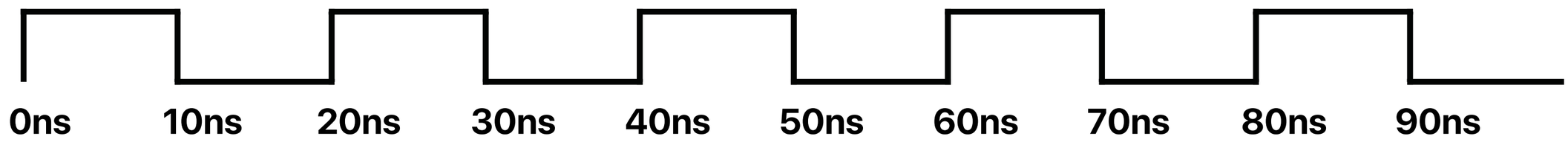
High-level State Machines & RTL Design

Prof. Usagi

Recap: Clock signal



- Clock -- Pulsing signal for enabling latches; ticks like a clock
- **The clock's period must be longer than the longest delay from the state register's output to the state register's input, known as the critical path.**
- Synchronous circuit: sequential circuit with a clock
- Clock period: time between pulse starts
  - Above signal: period = 20 ns
- Clock cycle: one such time interval
  - Above signal shows 3.5 clock cycles
- Clock duty cycle: time clock is high
  - 50% in this case
- Clock frequency: 1/period
  - Above : freq = 1 / 20ns = 50MHz;

Recap: Frequency

- Consider the following adders. Assume each gate delay is 1ns and the delay in a register is 2ns. Please rank their maximum operating frequencies
  - ① 32-bit CLA made with 8 4-bit CLA adders $\frac{1}{17ns} = 58.8MHz$
  - ② 32-bit CRA made with 32 full adders $\frac{1}{64ns} = 15.6MHz$
  - ③ 32-bit serial adders made with 4-bit CLA adders $\frac{1}{5ns} = 200MHz$
  - ④ 32-bit serial adders made with 1-bit full adders $\frac{1}{4ns} = 250MHz$
  - A. (1) > (2) > (3) > (4)
  - B. (2) > (1) > (4) > (3)
  - C. (2) > (1) > (3) > (4)
  - D. (4) > (3) > (2) > (1)
  - E. (4) > (3) > (1) > (2)

Recap: Area/Delay of adders

- Consider the following adders?
  - ① 32-bit CLA made with 8 4-bit CLA adders
  - ② 32-bit CRA made with 32 full adders
  - ③ 32-bit serial adders made with 4-bit CLA adders
  - ④ 32-bit serial adders made with 1-bit full adders

Each CLA — 2-gate delay — 8*2+1 ~ 17

Each carry — 2-gate delay — 64

Each CLA — (3-gate delay + 2-gate delay)*8 cycles — 5*8+1 = 41

Each CLA — (2-gate delay + 2-gate delay)*32 cycles — 4*32 = 128

A. Area: (1) > (2) > (3) > (4) Delay: (1) < (2) < (3) < (4)

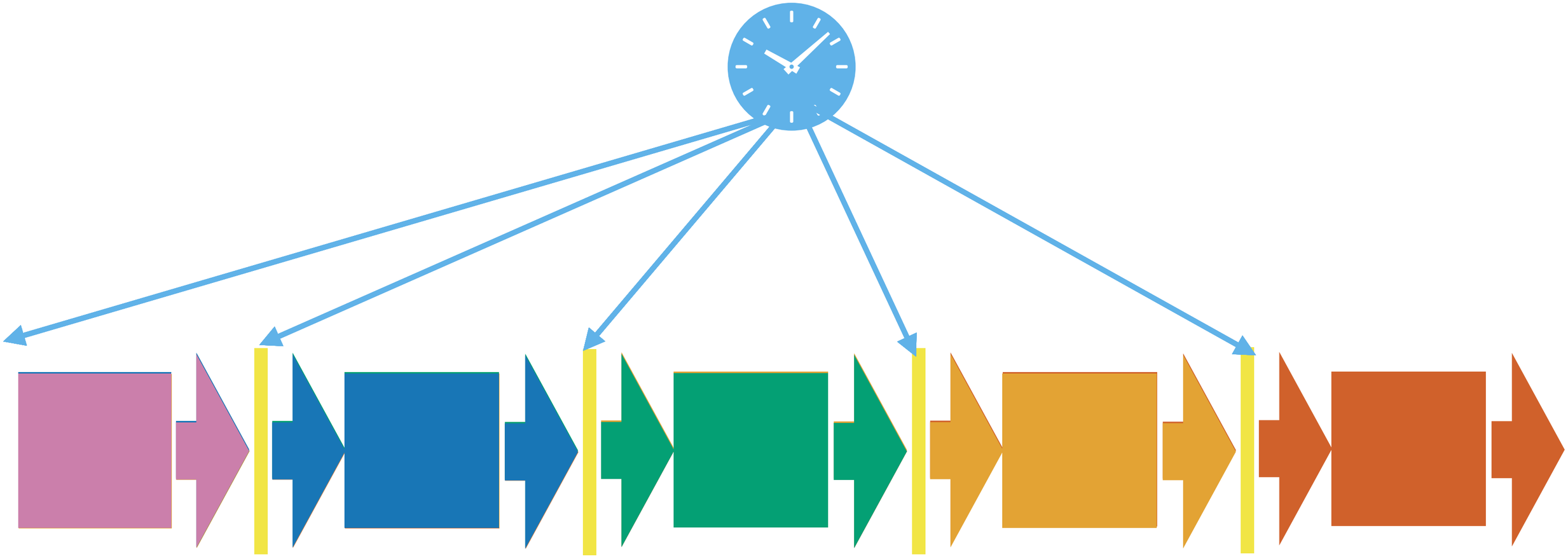
B. Area: (1) > (3) > (2) > (4) Delay: (1) < (3) < (2) < (4)

C. Area: (1) > (3) > (4) > (2) Delay: (1) < (3) < (4) < (2)

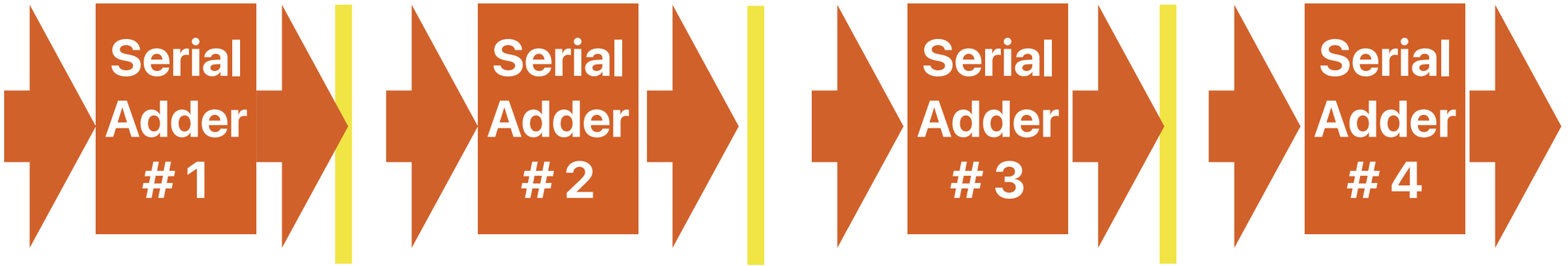
D. Area: (1) > (2) > (3) > (4) Delay: (1) < (3) < (2) < (4)

E. Area: (1) > (3) > (2) > (4) Delay: (1) < (3) < (4) < (2)

Recap: Pipelining

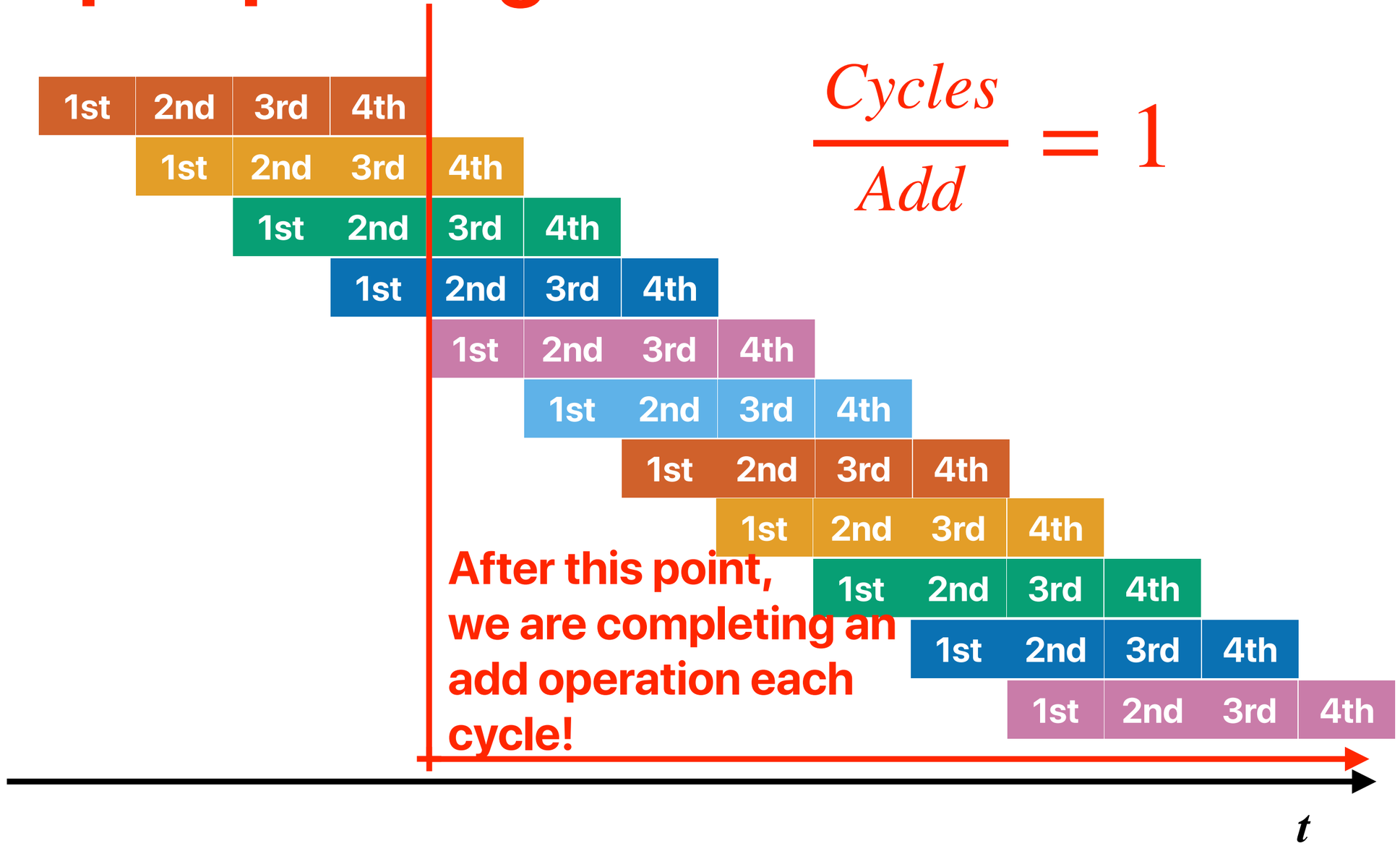


Recap: Pipelining a 4-bit serial adder



Recap: Pipelining a 4-bit serial adder

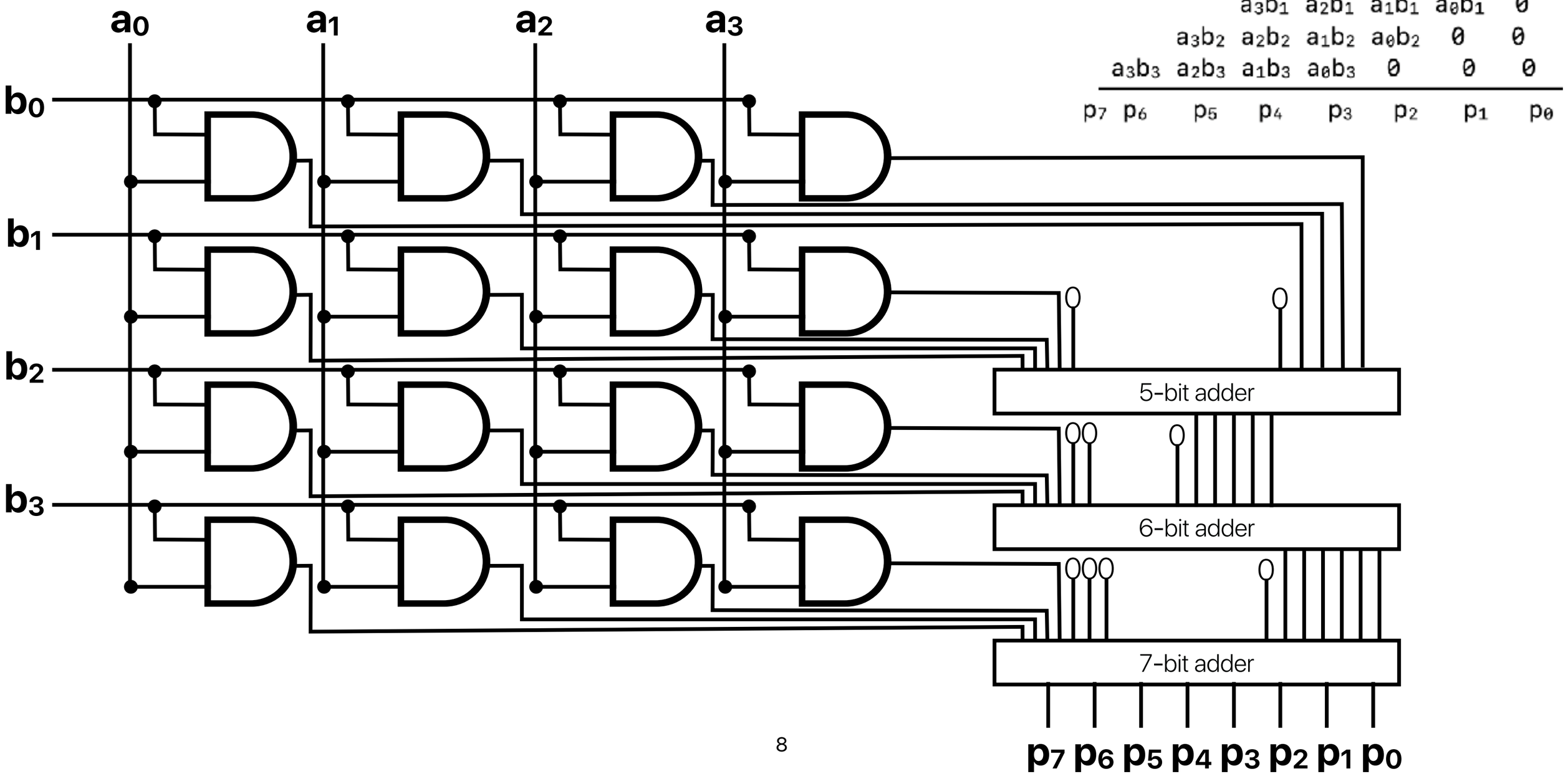
```
add a, b
add c, d
add e, f
add g, h
add i, j
add k, l
add m, n
add o, p
add q, r
add s, t
add u, v
```



$$\frac{Cycles}{Add} = 1$$

Recap: Array style

			a_3	a_2	a_1	a_0	
		×	b_3	b_2	b_1	b_0	
			a_3b_0	a_2b_0	a_1b_0	a_0b_0	
		a_3b_1	a_2b_1	a_1b_1	a_0b_1	0	
	a_3b_2	a_2b_2	a_1b_2	a_0b_2	0	0	
a_3b_3	a_2b_3	a_1b_3	a_0b_3	0	0	0	
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0



Recap: Gate-delays of 32-bit array-style multipliers

- What's the estimated gate-delay of a 32-bit multiplier? (Assume adders are composed of 4-bit CLAs)
  - A. 0 — 100
  - B. 100 — 500
  - C. 500 — 1000
  - D. 1000 — 1500
  - E. > 1500

Each n-bit adder is roundup(n/4)*2+1

We need 33-64 bit adders

33 - 36 -bit adders —> (9*2+1) gate delays *4

37 - 40 -bit adders —> (10*2+1) gate delays *4

41 - 44 -bit adders —> (11*2+1) gate delays *4

45 - 48 -bit adders —> (12*2+1) gate delays *4

49 - 52 -bit adders —> (13*2+1) gate delays *4

53 - 56 -bit adders —> (14*2+1) gate delays *4

57 - 60 -bit adders —> (15*2+1) gate delays *4

61 - 64 -bit adders —> (16*2+1) gate delays *4

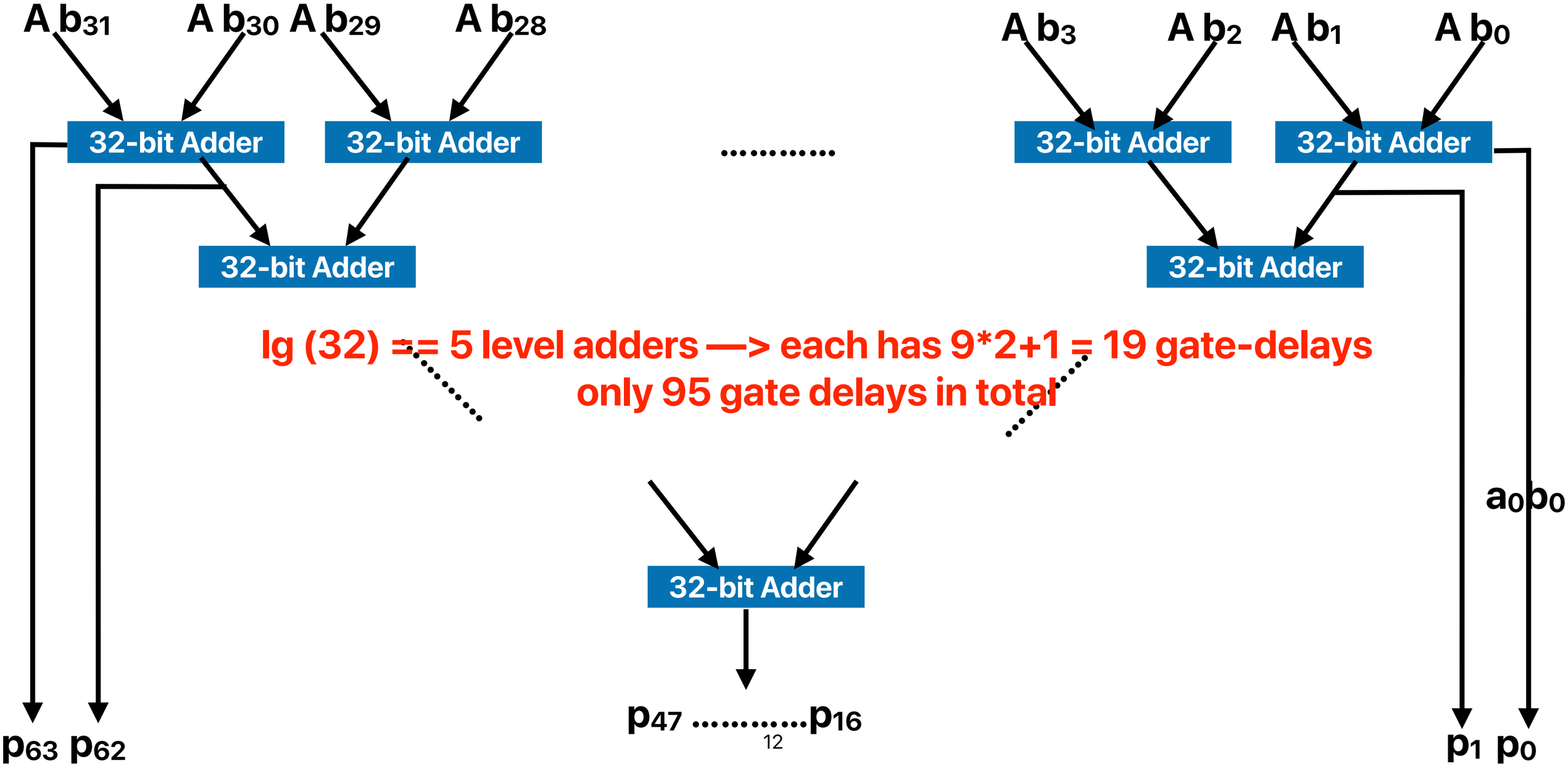
4*2*(9+10+11+12+13+14+15+16+1) = 808

Outline

- More multipliers
- HLSM
- RTL Design
- Designing a simple “microprocessor”

More on multipliers

Parallel-tree Multiplier



Sequential Logic based Multiplier!

Binary multiplication

- Thinking about how you do this by hand in decimal!

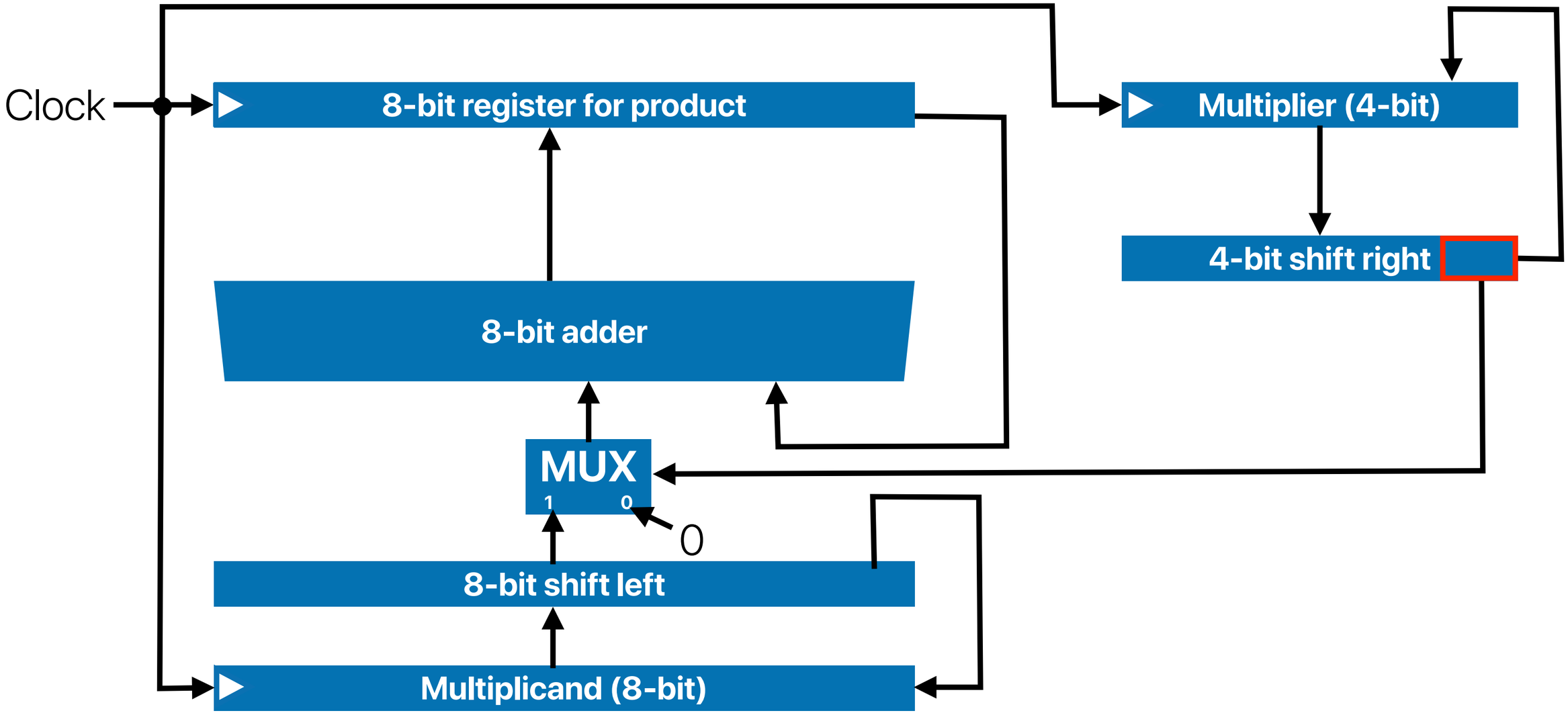
$$m = A \times B$$

```
      1 2 3 4          0 1 1 1
    × 5 6 7 8        × 1 1 0 0
-------------    -------------
      9 8 7 2          0 0 0 0
    8 6 3 8          0 0 0 0
  7 4 0 4          0 1 1 1
6 1 7 0          0 1 1 1
-------------    -------------
7 0 0 6 6 5 2    1 0 1 0 1 0 0
```

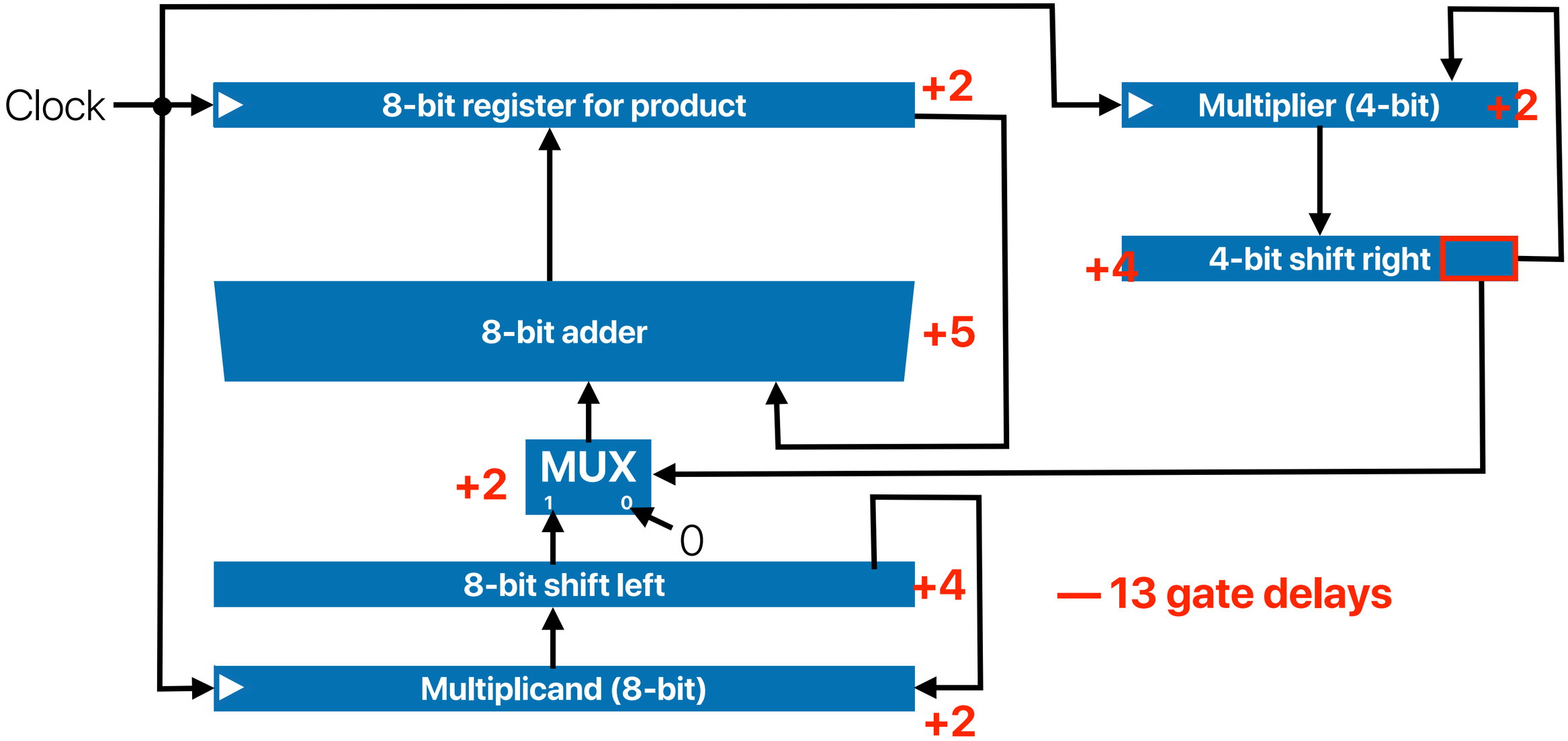
			a_3	a_2	a_1	a_0	
			× b_3	b_2	b_1	b_0	
pp1			a_3b_0	a_2b_0	a_1b_0	a_0b_0	
pp2		a_3b_1	a_2b_1	a_1b_1	a_0b_1	0	
pp3	a_3b_2	a_2b_2	a_1b_2	a_0b_2	0	0	
pp4	a_3b_3	a_2b_3	a_1b_3	a_0b_3	0	0	0
	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

$$m_{i+1} = m_i + Ab_i 2^i$$

4-bit serial shift-and-add multiplier



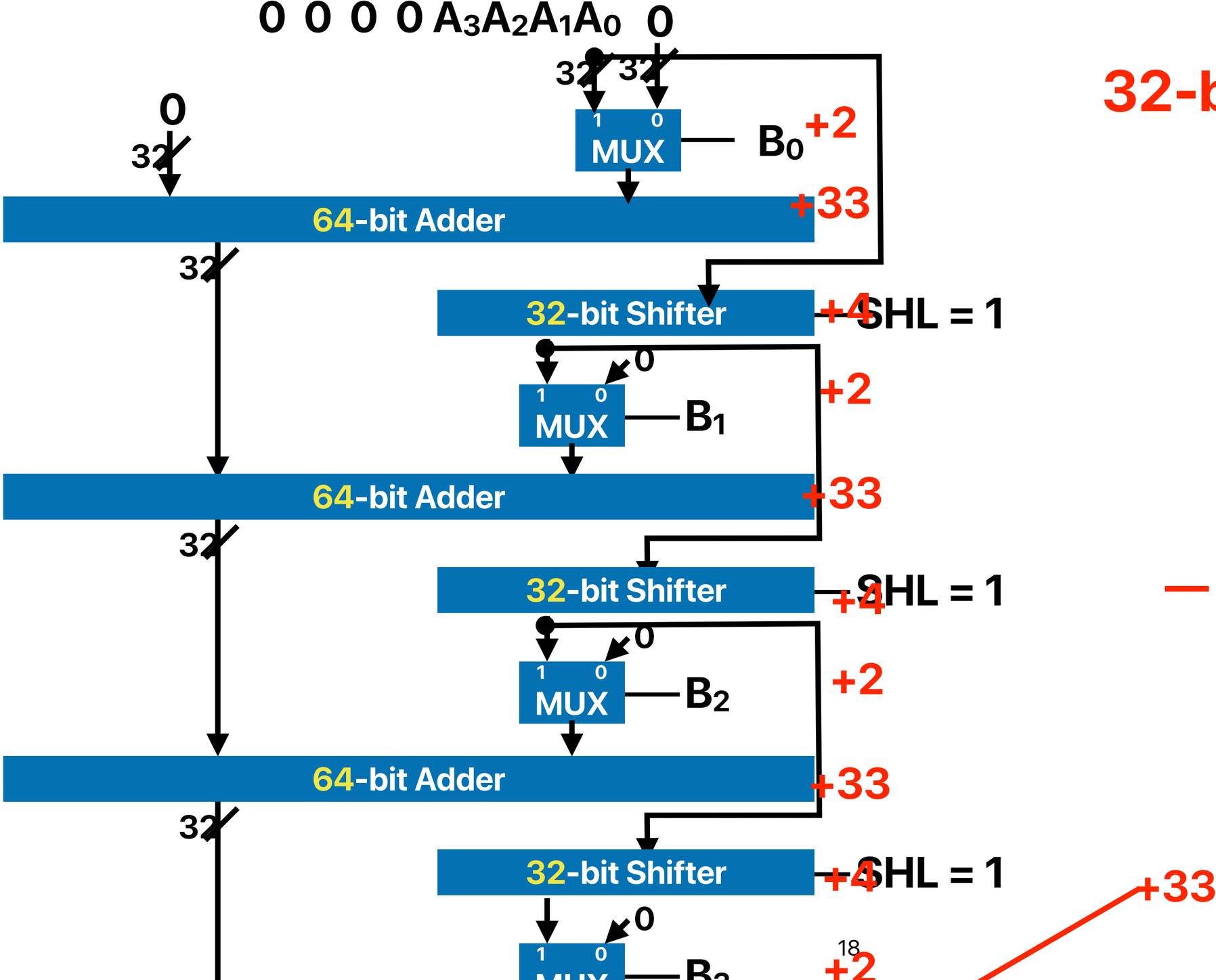
4-bit serial shift-and-add multiplier



Poll close in 1:30

Latency of multipliers

- Consider the following multipliers and assume each gate delay is 1ns and the delay in a register is 2ns. If all circuits can operate their maximum frequency, please identify the multiplier with shortest end-to-end latency in generating the result for multiplying two 32-bit numbers
  A. 32-bit shift and add multipliers
  B. 32-bit array-style multipliers
  C. Pipelined 32-bit serial shift-and-add multiplier

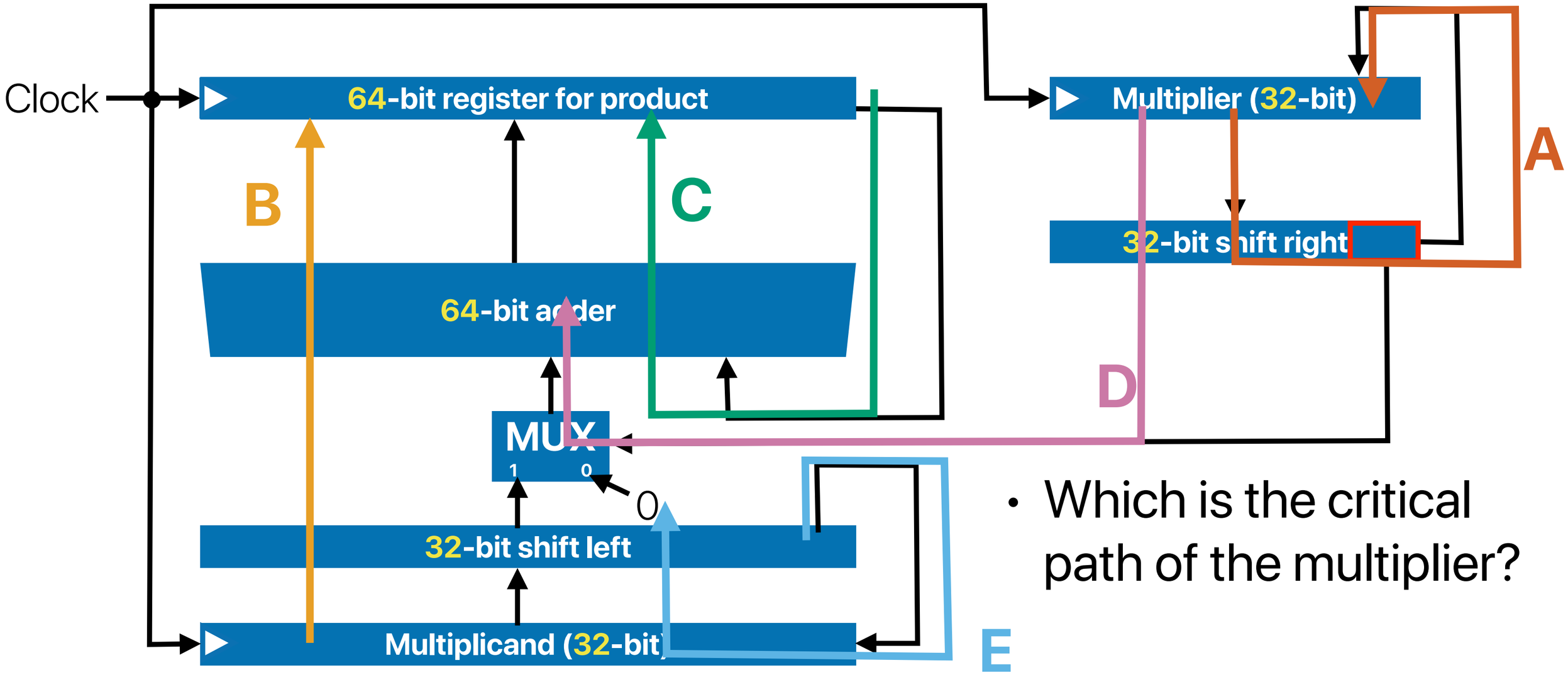


32-bit shift and add

— 39*32 gate delays

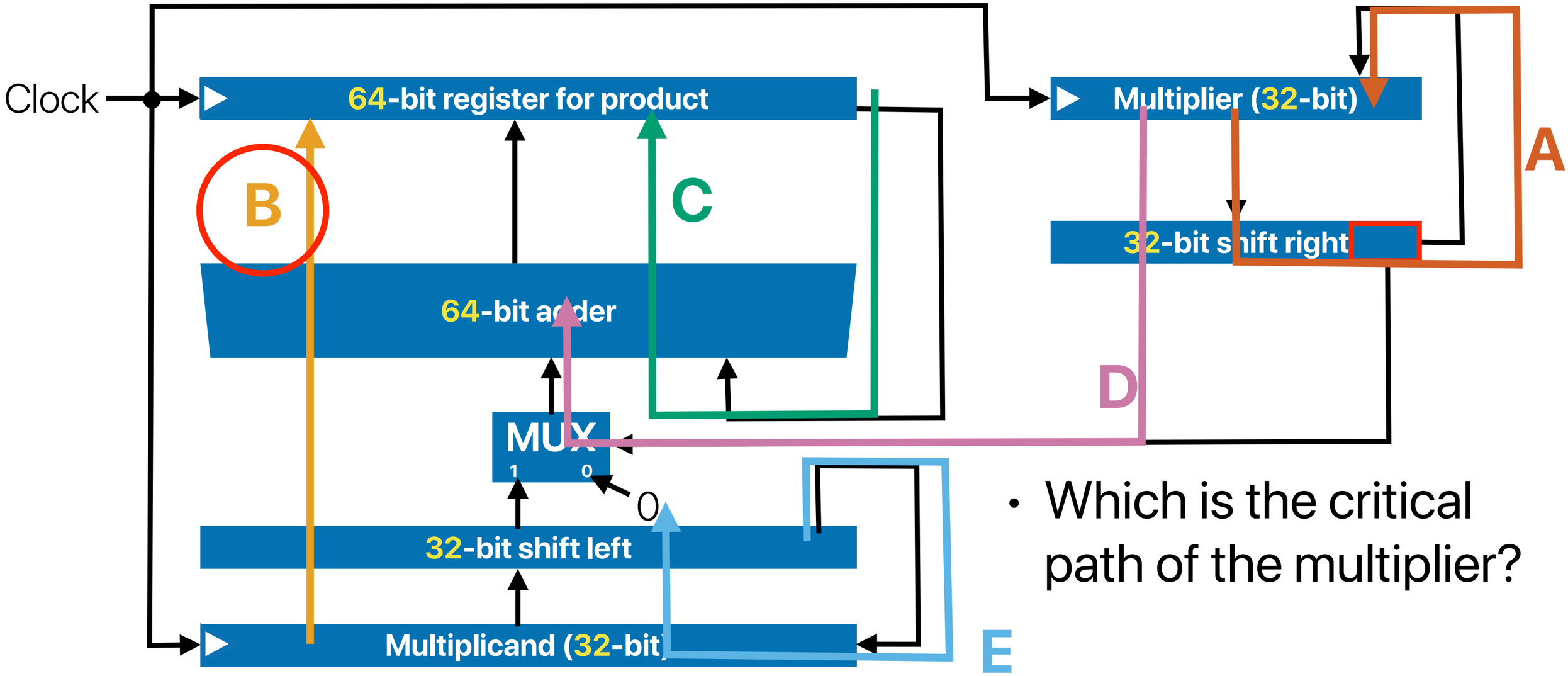
Poll close in 1:30

32-bit serial shift-and-add multiplier



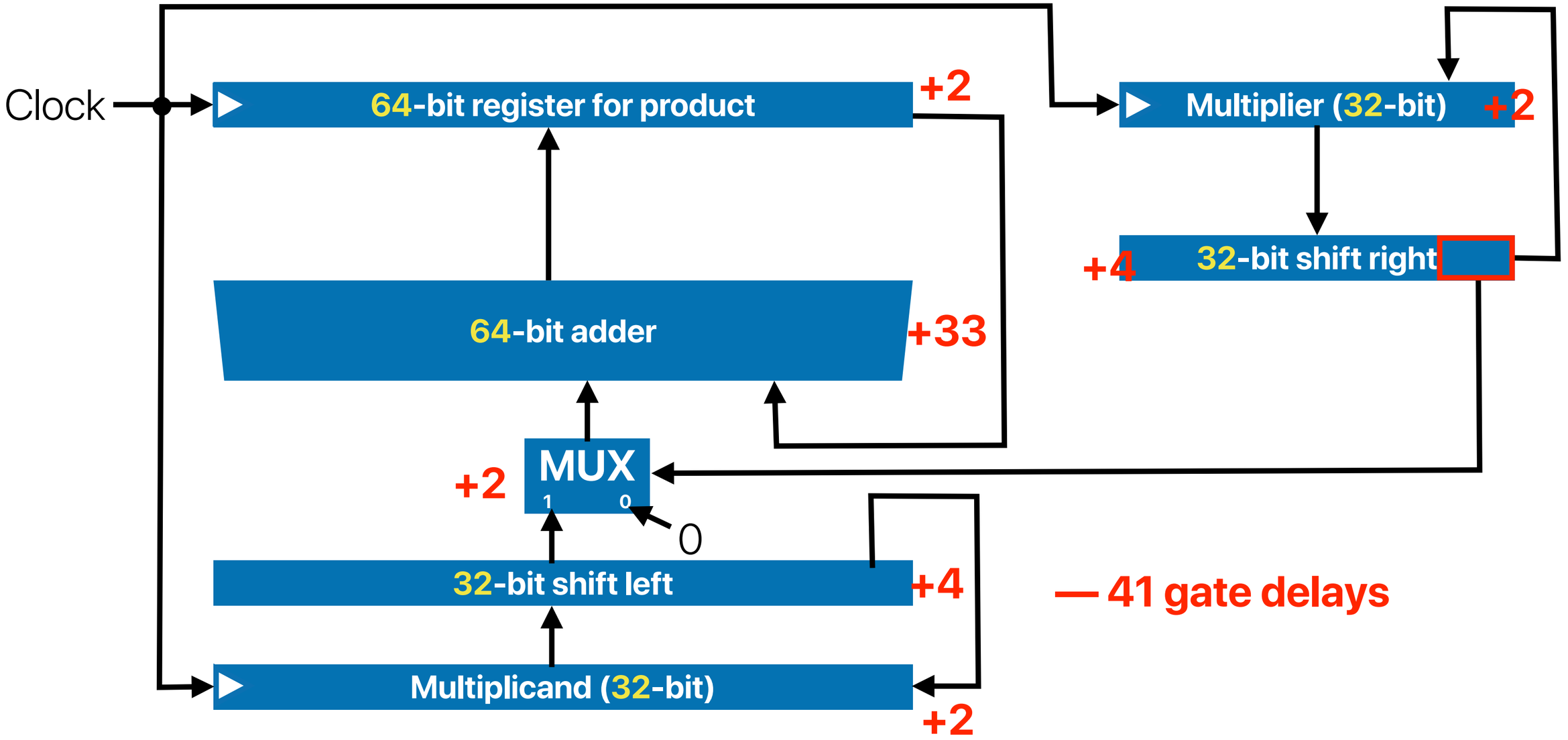
- Which is the critical path of the multiplier?

32-bit serial shift-and-add multiplier



- Which is the critical path of the multiplier?

32-bit serial shift-and-add multiplier



— 41 gate delays

Latency of multipliers

- Consider the following multipliers and assume each gate delay is 1ns and the delay in a register is 2ns. If all circuits can operate their maximum frequency, please identify the multiplier with shortest end-to-end latency in generating the result for multiplying two 32-bit numbers
  A. 32-bit shift and add multipliers — 39*32 = 1248 gate delays
  B. 32-bit array-style multipliers — 808 gate delays
  C. Pipelined 32-bit serial shift-and-add multiplier — 41*32 = 1312 gate delays

Poll close in 1:30

Throughput of multipliers

- Consider the following multipliers and assume each gate delay is 1ns and the delay in a register is 2ns. If all circuits can operate their maximum frequency, please identify the multiplier with shortest end-to-end latency in generating the result for multiplying two million pairs of 32-bit numbers
  A. 32-bit shift and add multipliers
  B. 32-bit array-style multipliers
  C. Pipelined 32-bit serial shift-and-add multiplier

Throughput of multipliers

- Consider the following multipliers and assume each gate delay is 1ns and the delay in a register is 2ns. If all circuits can operate their maximum frequency, please identify the multiplier with shortest end-to-end latency in generating the result for multiplying two million pairs of 32-bit numbers
  A. 32-bit shift and add multipliers
  B. 32-bit array-style multipliers
  C. Pipelined 32-bit serial shift-and-add multiplier

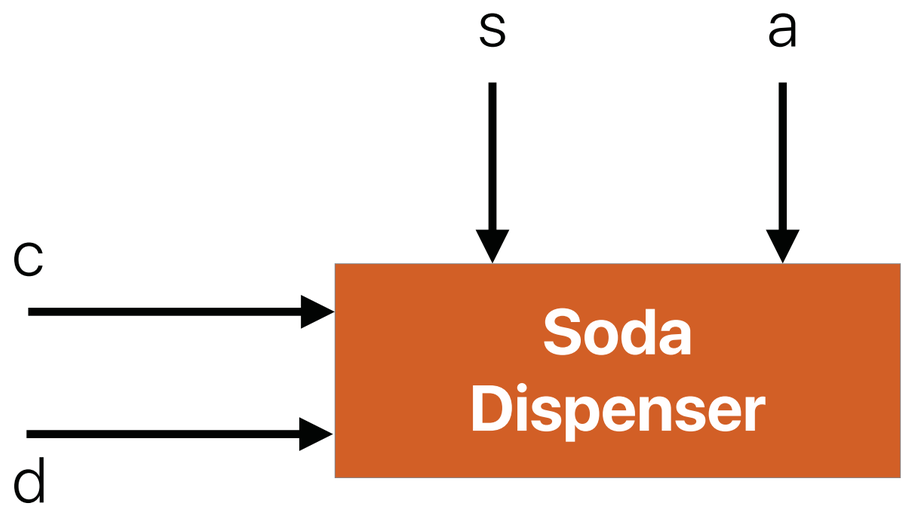
Let’s put all things together!

- We have learned all datapath components for an ALU!
  - Register
  - Shifter
  - Adders
  - Multiplier
- Processor has only one clock generator
  - Each datapath component has a different latency
  - We have make some of the above “serial”
  - How to **control**?

HLSM — High-Level State Machine

High-Level State Machine

- Some behaviors may be too complex to describe by using classical FSMs
- Soda dispenser
  - c: bit input, 1 when coin deposited
  - a: 8-bit input: value of the deposited coin
  - s: 8-bit input: cost of a soda
  - d: bit output, processor sets it to 1 when total value of deposited coins equals or exceeds cost of a soda



Poll close in 1:30

HLSMs v.s. FSMs

- How does the HLSM differ from the FSM for this problem?
  A. The HLSM stores multibit data, but the FSM doesn't
  B. The FSM stores the state but the HLSM doesn't
  C. Implementing HLSM and FSM requires multibit data registers
  D. All of the above
  E. None of the above

HLSMs v.s. FSMs

- How does the HLSM differ from the FSM for this problem?
  A. The HLSM stores multibit data, but the FSM doesn't
  B. The FSM stores the state but the HLSM doesn't
  C. Implementing HLSM and FSM requires multibit data registers
  D. All of the above
  E. None of the above

Poll close in 1:30

HLSMs v.s. FSMs

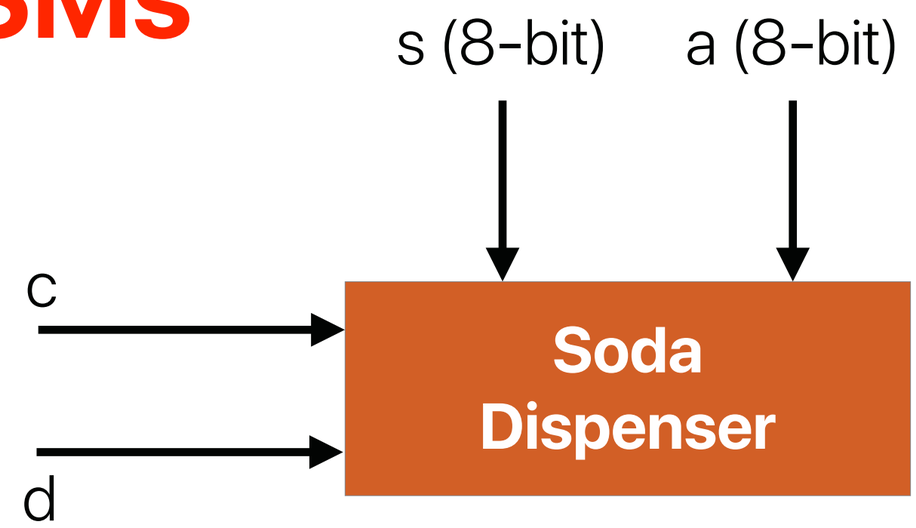
- Which of the following are common between HLSMs and FSMs?
  A. Transitions happen at the edge of a clock
  B. They both have external complex data
  C. All of the above
  D. None of the above

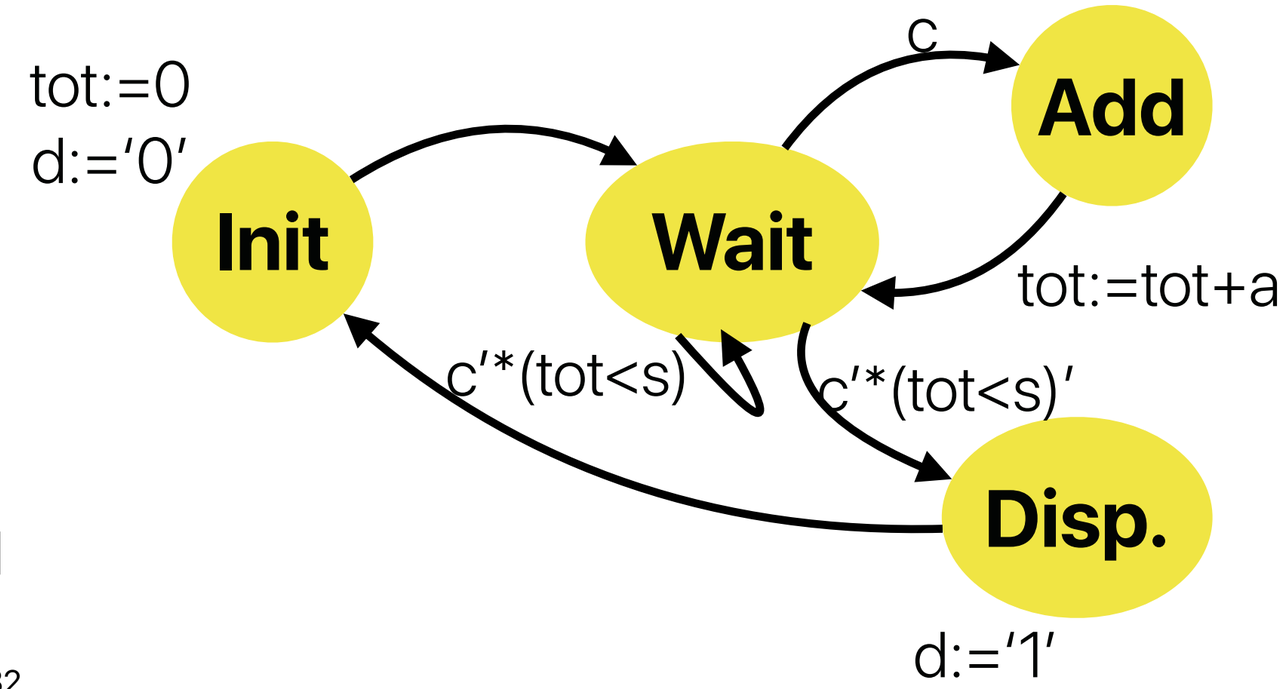
HLSMs v.s. FSMs

- Which of the following are common between HLSMs and FSMs?
  A. Transitions happen at the edge of a clock
  B. They both have external complex data
  C. All of the above
  D. None of the above

Benefits of HLSMs

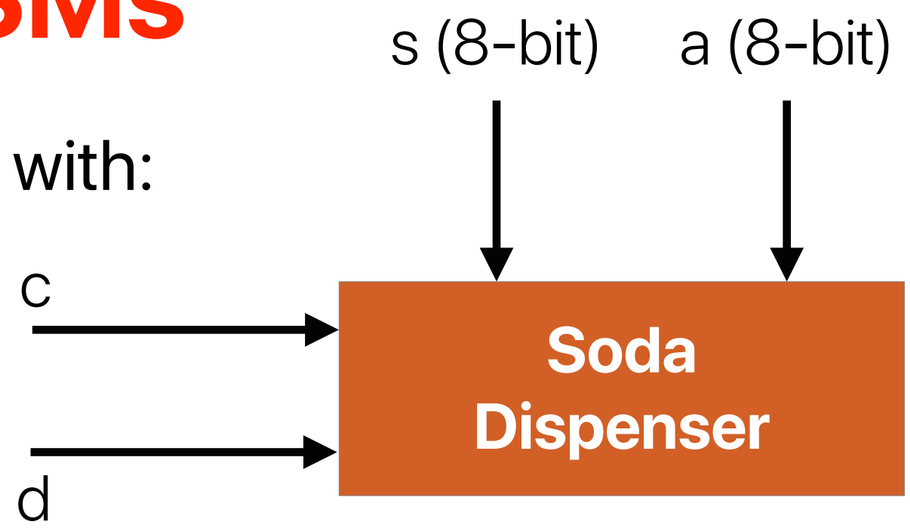
- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output
  - Local storage
  - Arithmetic operations
- Conventions
  - Each transition is implicitly ANDed with a rising edge of the clock
  - Any bit output not explicitly assigned a value in a state is implicitly assigned to 0. This convention does not apply for multibit outputs
  - Every HLSM multibit output is registered

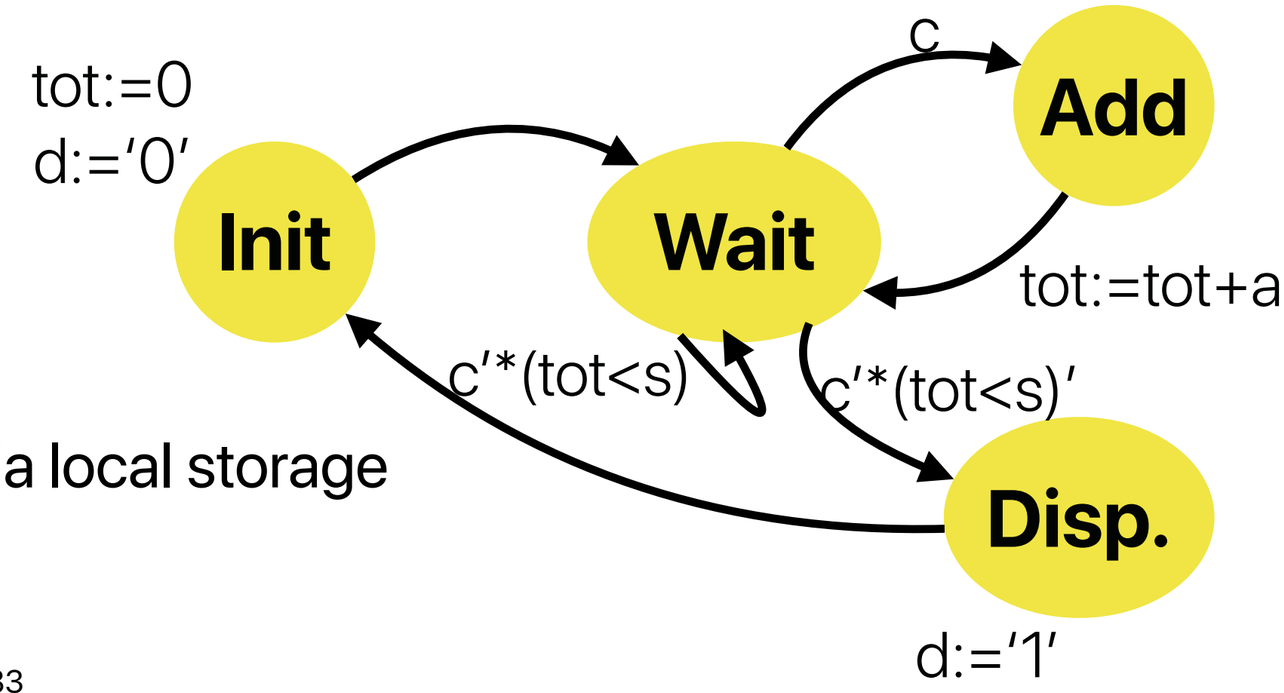




Benefits of HLSMs

- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output
  - Local storage
  - Arithmetic operations
- Conventions
  - Numbers:
    - Single-bit: '0' (single quotes)
    - Integer: 0 (no quotes)
    - Multi-bit: “0000” (double quotes)
  - == for comparison equal
  - Multi-bit outputs must be registered via local storage
  - – // precedes a comment





RTL(Register Transfer Level) Design

RTL Design Process

- Step 1: Capture a high-level state machine
  - Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is high level because the transition conditions and the state actions are more than just Boolean operations on single-bit input and outputs
  - Recommendations:
    - Always list all inputs, outputs and local registers on top of your HLSM diagram
    - Clearly specify the size in bits of each of them
    - On states: update the value of registers, update of outputs
    - On transitions: express conditions in terms of the HLSM inputs or state of the internal values and arithmetic operations between them.

RTL Design Process

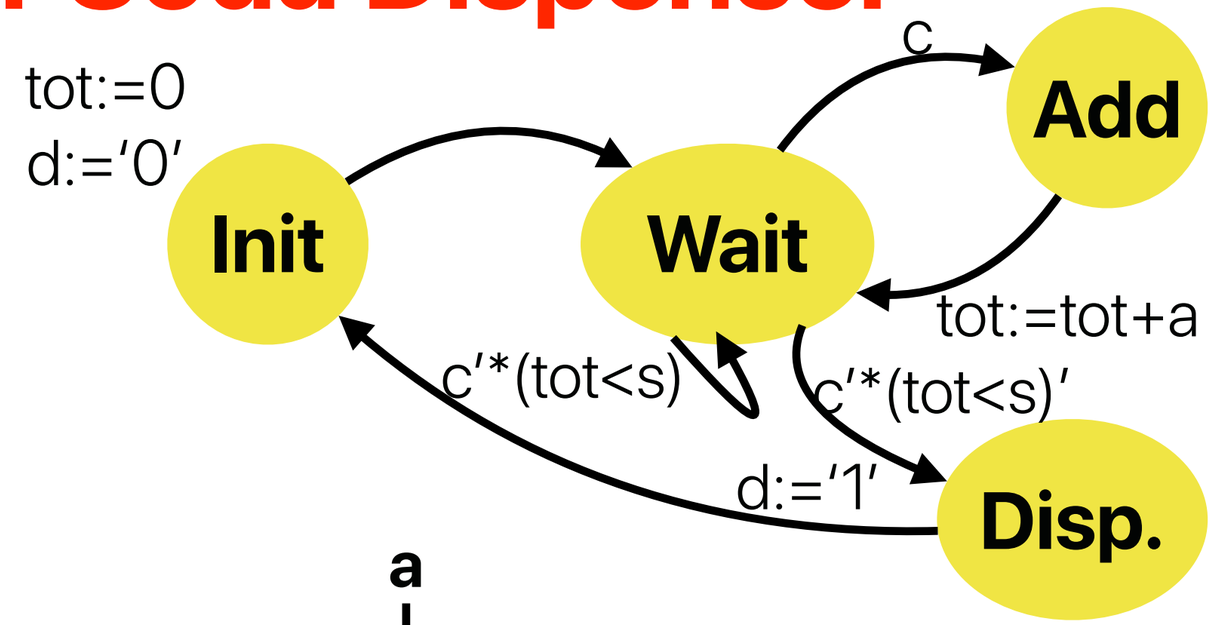
- Step 2: Convert it to a circuit
  - Create a datapath
    - Create a datapath to carry out the data operations of the high level state machine
    - Elements of your datapaths can be registers, adders, comparators, multipliers, dividers, etc.
  - Connect the datapath to a controller
    - Connect the datapath to a controller block.
    - Connect the external control inputs and outputs to the controller block.
    - Clearly label all control signals that are exchanged between the datapath and the controller
  - Derive the controller's FSM
    - Convert the high-level state machine to a finite state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath
- Final Step Implement the FSM as a state register and logic

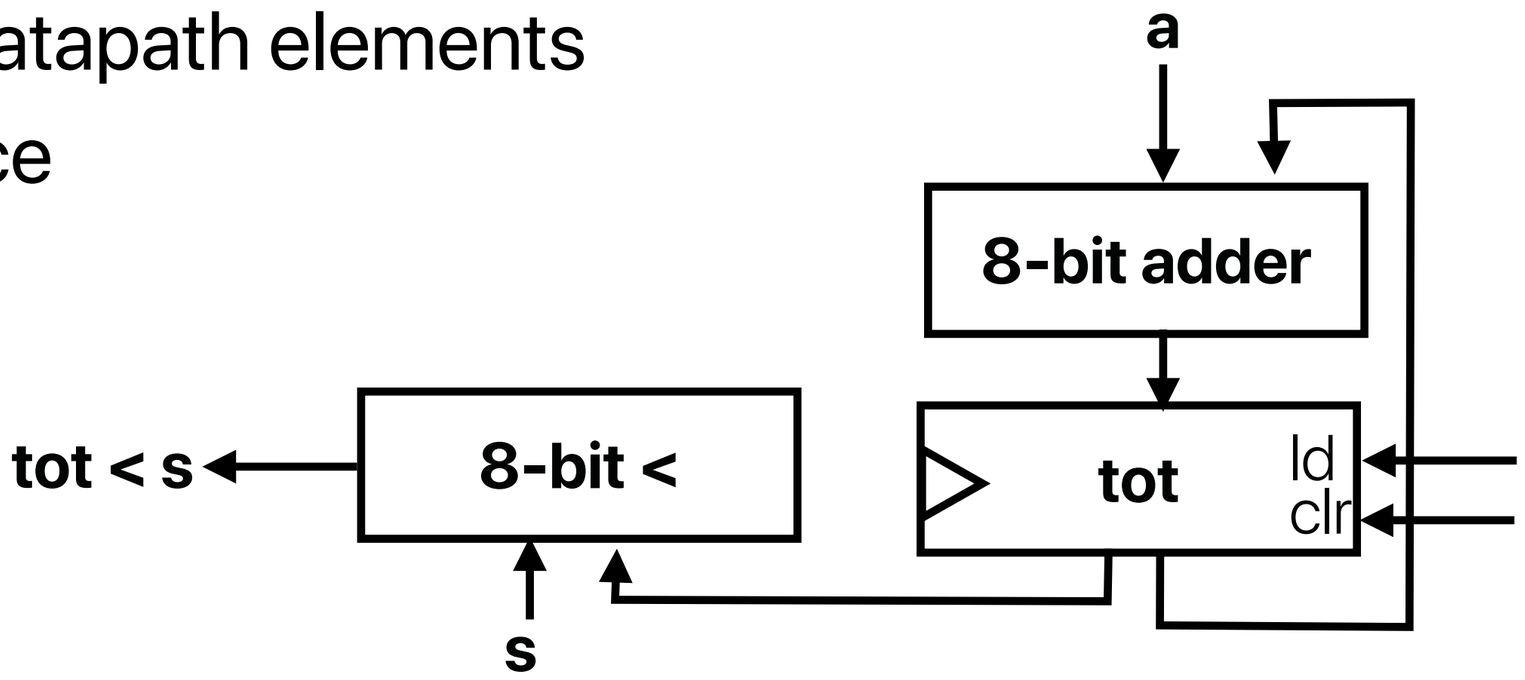
RTL Design Summary

- Capture the behavior with HLSM
- Convertit to a circuit
  - High-level architecture (datapath and control path)
  - Datapath capable of HLSM's data operations
  - Design controller to control the datapath

Create Datapath for Soda Dispenser

- Register: tot
- Comparator: to compare tot and s
- Adder: to update tot = tot + a
- Connect datapath elements
- I/O interface





Announcement

- iEval — Capture your screenshot and you will receive a full credit assignment
- Lab 6 is up — due on 6/2
  - Watch the video and read the instruction BEFORE your session
  - There are links on both course webpage and iLearn lab section
  - Submit through iLearn > Labs
- Office Hours
  - All office hours share the same meeting instance — if you have registered once, you cannot do it again.
  - Zoom does not resend registration confirmation and does not allow us to “re-approve” if you have registered
  - The only way is to dig out the e-mail from Zoom
- Final exam will be held during the campus scheduled period to avoid conflicts
  - 6/11 11:30am — 2:59:59pm
  - About the same format as midterm, but longer
  - Will have a final review on 6/6 to help you prepare

Electrical
Computer Science
Engineering

120A

つづく