CS3350B Computer Organization
Chapter 2: Synchronous Circuits
Part 3: State Circuits

Iqra Batool

Department of Computer Science
University of Western Ontario, Canada

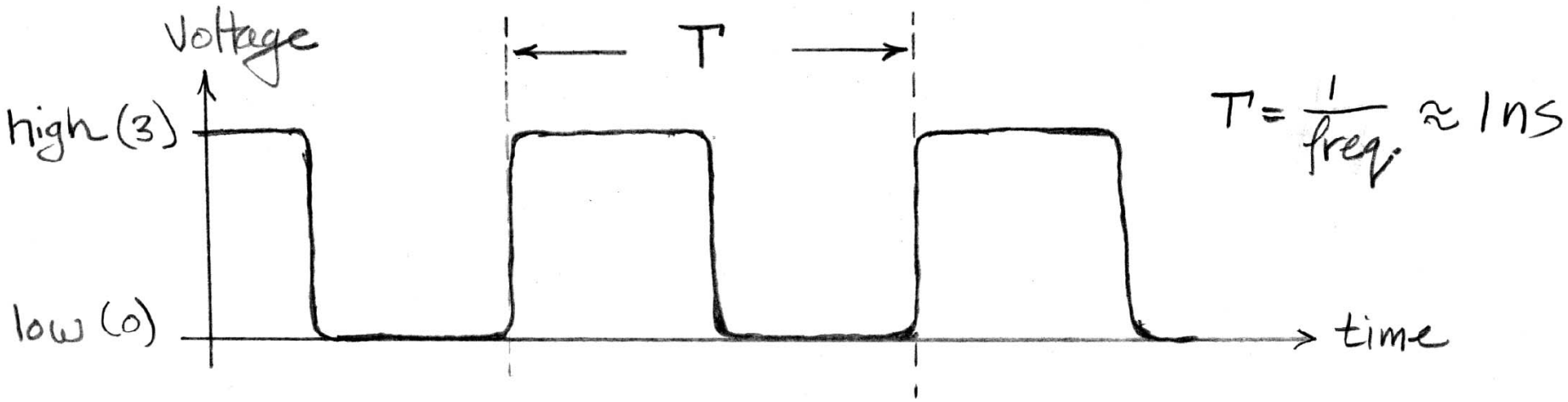
Monday February 5, 2024

Outline

Digital Signals

We digitalize an analog (voltage) signal to encode binary.

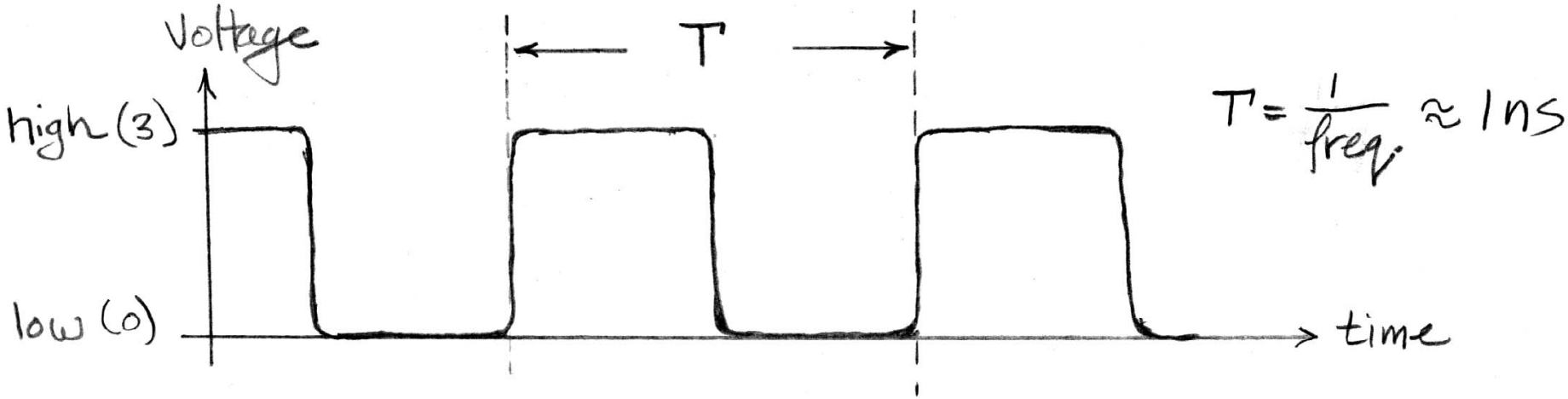
- "High" voltage ⇒ 1.
- "Low" voltage ⇒ 0.



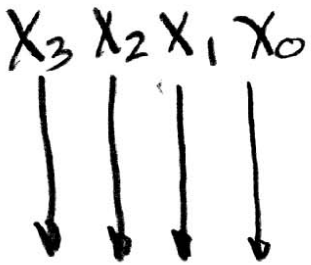
Transmitting Digital Signals

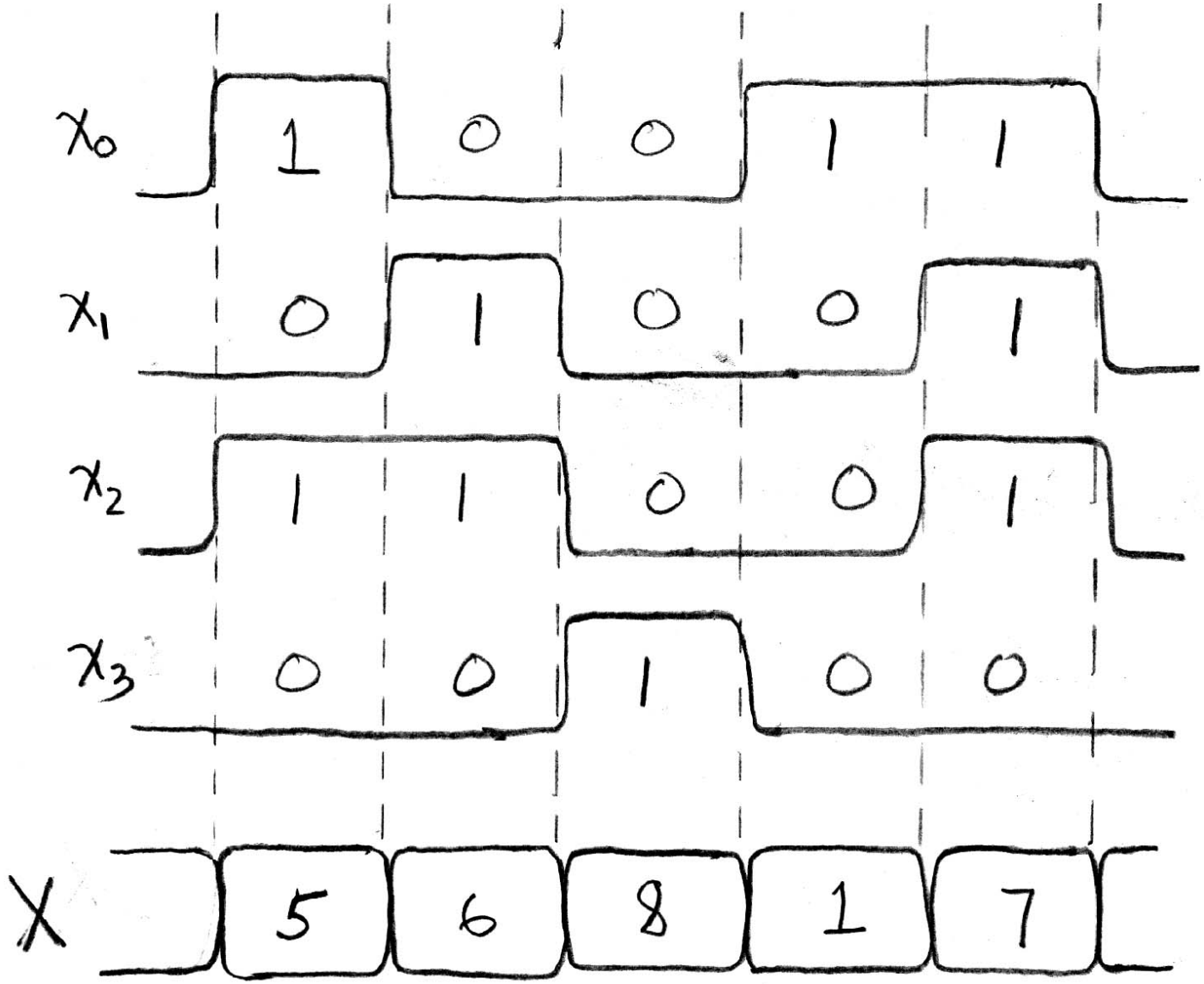
For our purposes:

- Transmission is continuous. There's always something on the wire.
- Transmission/switching is effectively instantaneous.



Grouping Signals To Encode Many Bits

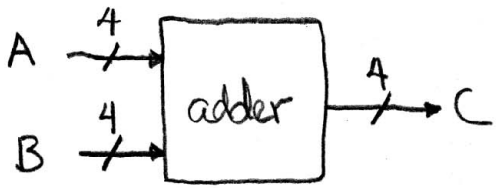




Signals and Circuits

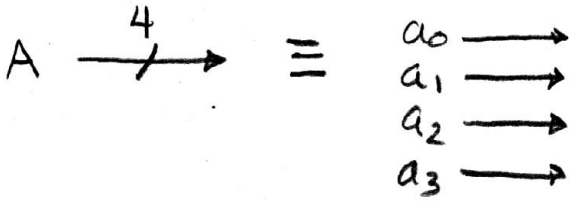
Unfortunately for us, combinational circuits cause **propagation delay**.

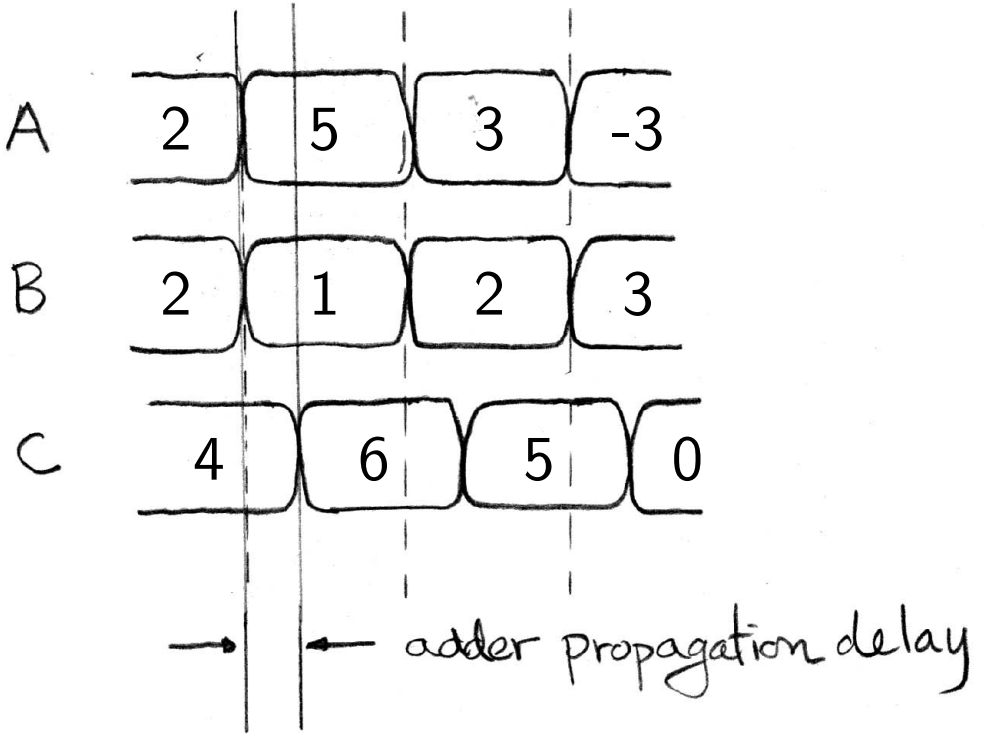
- The more complex the circuit the longer the delay.
- Every gate adds some delay.



$A = [a_3, a_2, a_1, a_0]$

$B = [b_3, b_2, b_1, b_0]$





Dealing with Delay

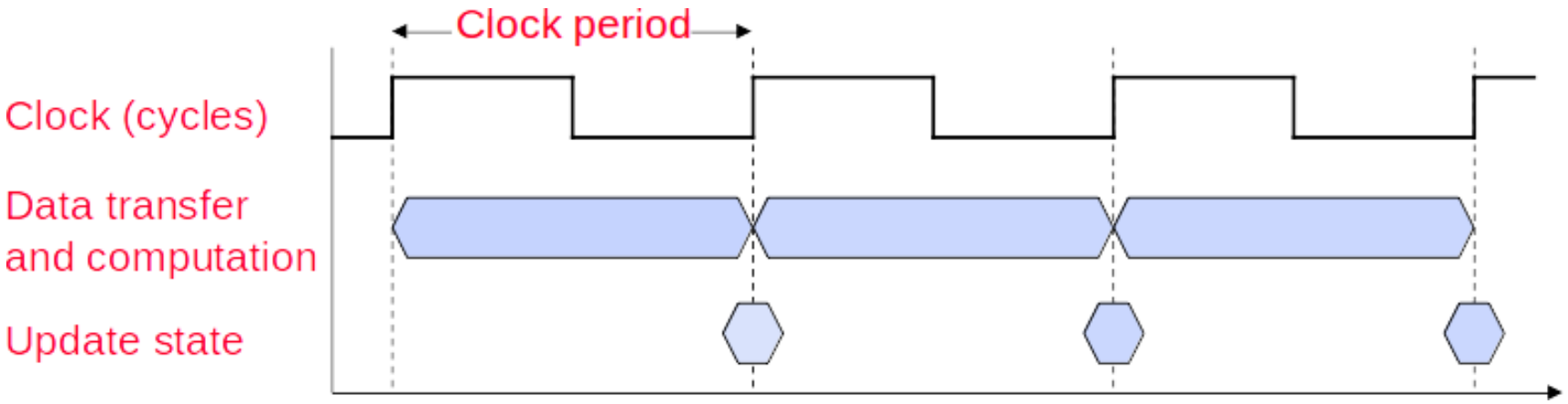
Problems with propagation delay:

- Inputs transmit (change) instantaneously, but *output* does not.
- When can the next circuit read the output and ensure it is getting the correct value?

Synchronize the circuits via a **clock**.

Outline

The Clock Signal



The clock is a digital signal which has a precise timing for switching between 1/0.

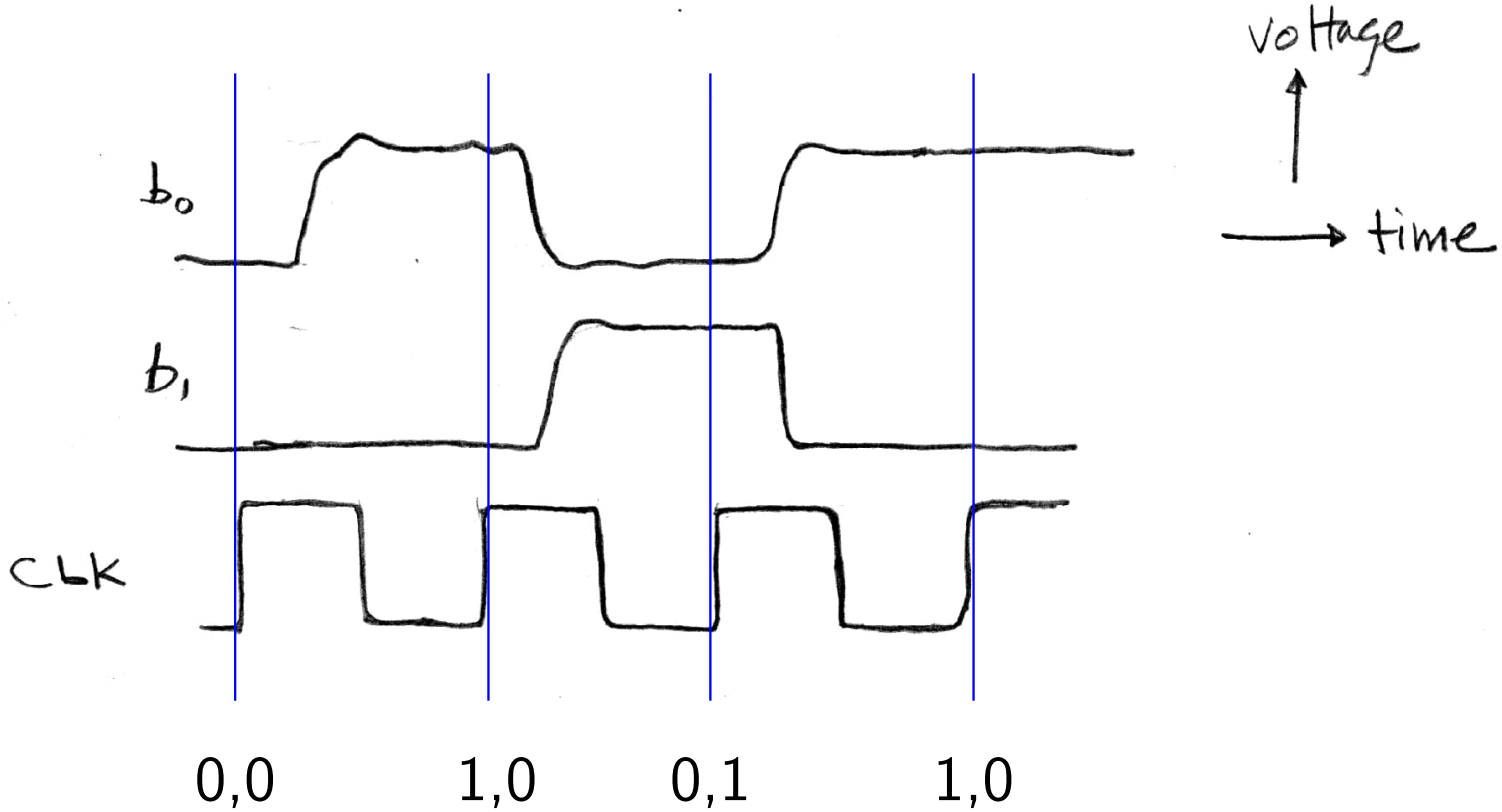
Synchronous circuits use the clock to sync their executions, decide when to read inputs/outputs.

↳ Heartbeat of a synchronous system.

How to Synchronize

Circuits usually synchronize to the **rising edge** of the clock.

- The transition from 0 to 1.
- Depending on the system, can instead sync on the **falling edge**.



Clock Multipliers

We know that CPU and memory operate at difference speeds. So how do they synchronize?

- One central clock used.
- Central clock is as slow as the slowest component.
- Faster components use a **clock multiplier**.

A clock multiplier multiplies the central clock frequency so that a component has many **internal cycles** for a single clock cycle of the *entire system*.

Note: this is simply a technicality of implementation. Generally, we still discuss speeds based on frequency the CPU experiences. Our old metrics still work as they always have.

Outline

Circuits that Remember

Sometimes values on a wire (i.e. a bit) cannot be maintained indefinitely on that wire. Values must change.

- Computer memory is circuits which *remember*.
- Circuits implement memory but are also used within other circuits to hold state.
  - ↳ Modular design.

Flip-flop: a circuit which implements a single bit of memory.

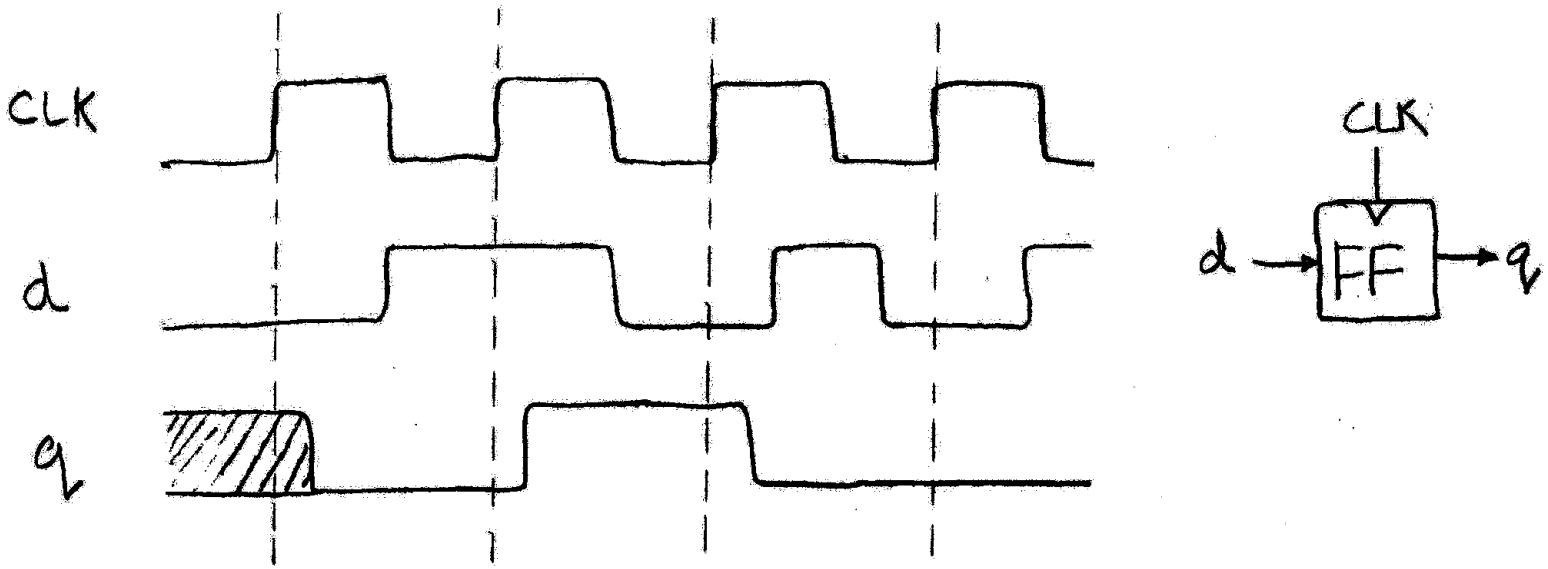
- ↳ All flip-flops based on a simple design: inputs, combined with current state, give next state.
- ↳ Essentially, the implementation of static RAM (SRAM).

Register: a storage for multiple bits of memory.

Edge-Triggered Flip-Flop

A flip-flop which looks at its input on the edge of clock.

- Rising edge or positive edge (usually), or
- Falling edge or negative edge.



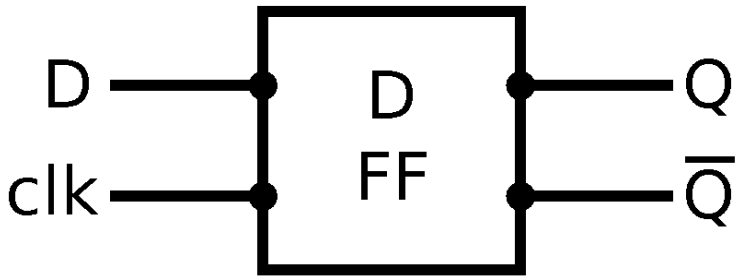
This is a *delay* flip-flop

D Flip-Flop

Delay flip-flop: takes input and, with some delay, sets output equal to the input.

- Simplest (conceptually) flip-flop.
- Requires constant updating to maintain state.
- Grabs input on rising edge and outputs that until next clock cycle.
- Current state does not affect next state.

D	Q	Q_{next}
0	-	0
1	-	1



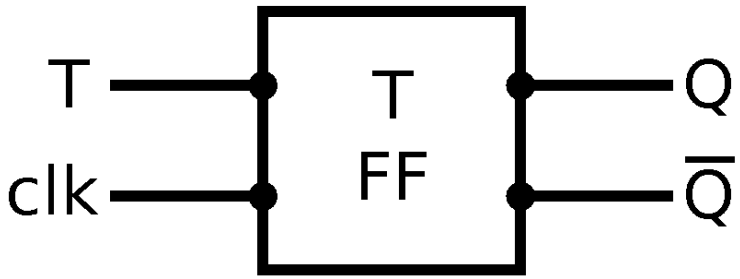
Flip-flops usually produce next state and negation of next state simultaneously.

T Flip-Flop

Toggle flip-flop: if input is 1, toggle current state.

- Uses current state to determine next state.
- $T \equiv 0 \Rightarrow$ "Hold". Next state is same as current.
- $T \equiv 1 \Rightarrow$ "Toggle". Next state is opposite of current.

T	Q	Q_{next}
0	0	0
0	1	1
1	0	1
1	1	0

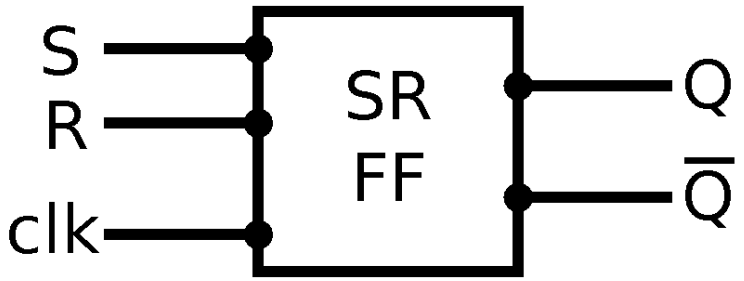


SR Flip-Flop

Set-Reset flip-flop

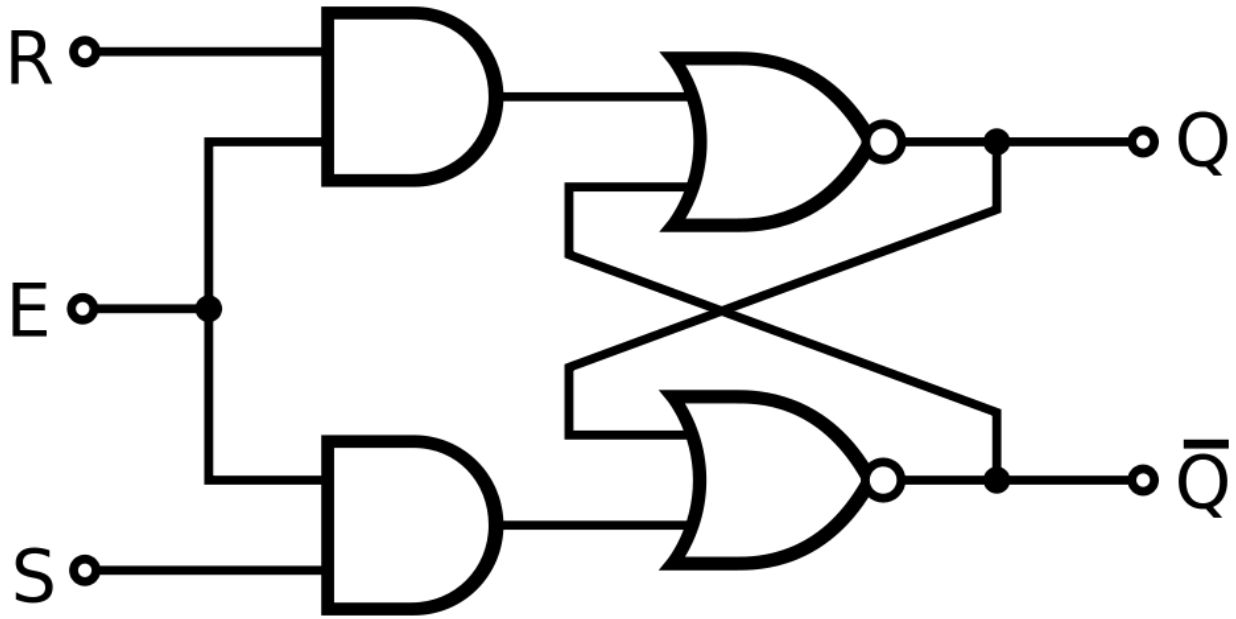
- Two inputs, S (set), R (reset), synchronized by a clock.
- $S \equiv 1 \Rightarrow$ "Set". Next state is 1.
- $R \equiv 1 \Rightarrow$ "Reset". Next state is 0.
- $S \equiv 0 \wedge R \equiv 0 \Rightarrow$ "Hold".

S	R	Q	Q_{next}
0	0	-	Q
0	1	-	0
1	0	-	1
1	1	-	✗



Can **not** have both S and R set to 1...

SR Technicalities



E "enable" $\iff$ clock

$$S \equiv R \equiv E \equiv 1 \implies \overline{1 + \overline{Q}} \equiv 0 \equiv \overline{1 + Q} \implies Q \equiv \overline{Q} \text{ ???}$$

We get undefined behaviour. This is weird and can destabilize the system.

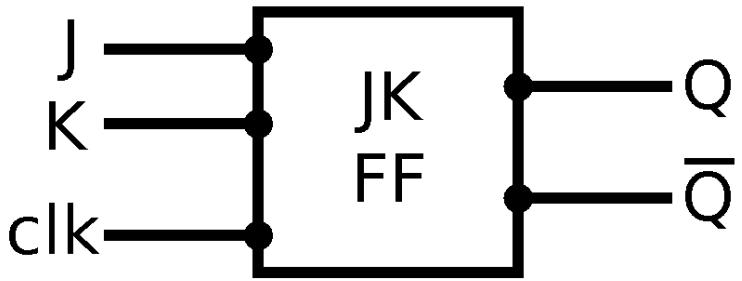
JK Flip-Flop

JK flip-flop

- Two inputs, J (set), K (reset), synchronized by a clock.
- Same as SR except with toggle.
- $J \equiv 1 \wedge K \equiv 1 \Rightarrow$ "Toggle".

set reset.

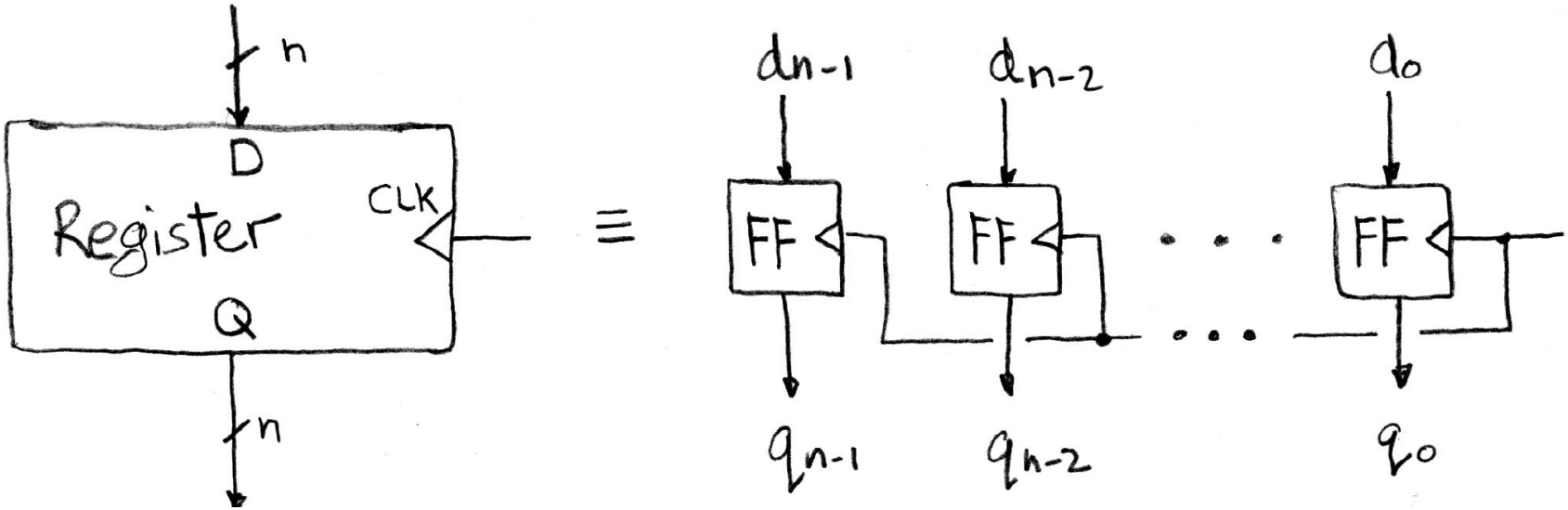
J	K	Q	Q_{next}
0	0	-	Q ← keep
0	1	-	0
1	0	-	1
1	1	-	$\overline{Q}$ ← toggle.



Registers

A register is just a collection of flip-flops.

- Technically, this is a *shift register*.
- n-bits $\implies$ n flip-flops.
- Clock pulse connected to all flip-flops.
- Can be encoded using any type of flip-flop.

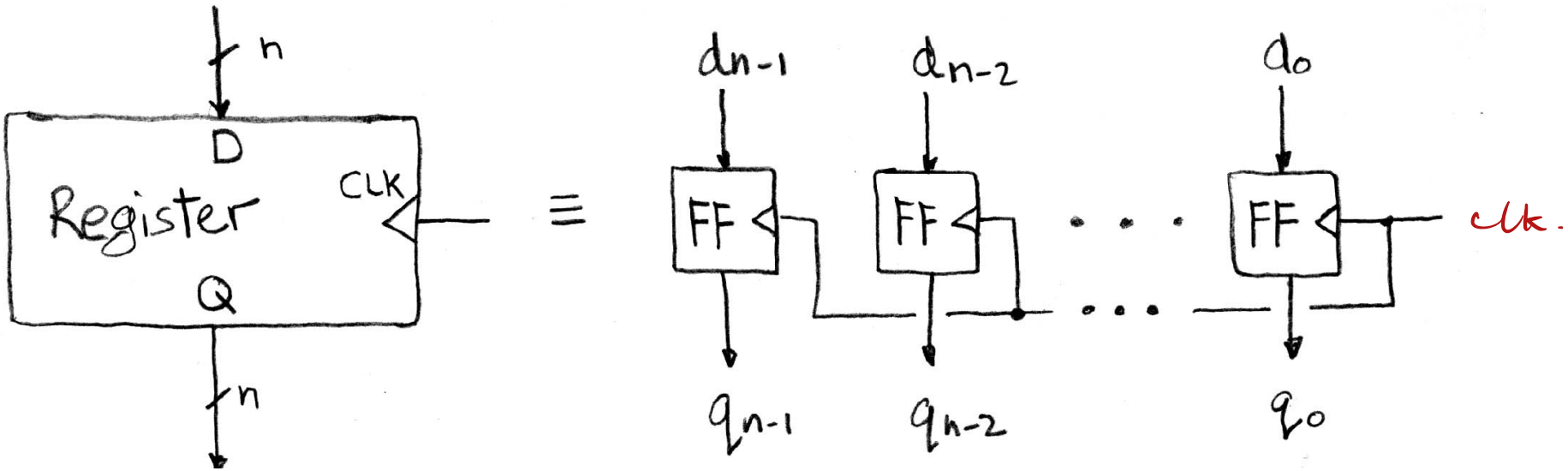


This example is a *parallel in, parallel out* register.

PIPO Registers

Parallel In, Parallel Out Register: All inputs bits come in in parallel, and output bits get output in parallel.

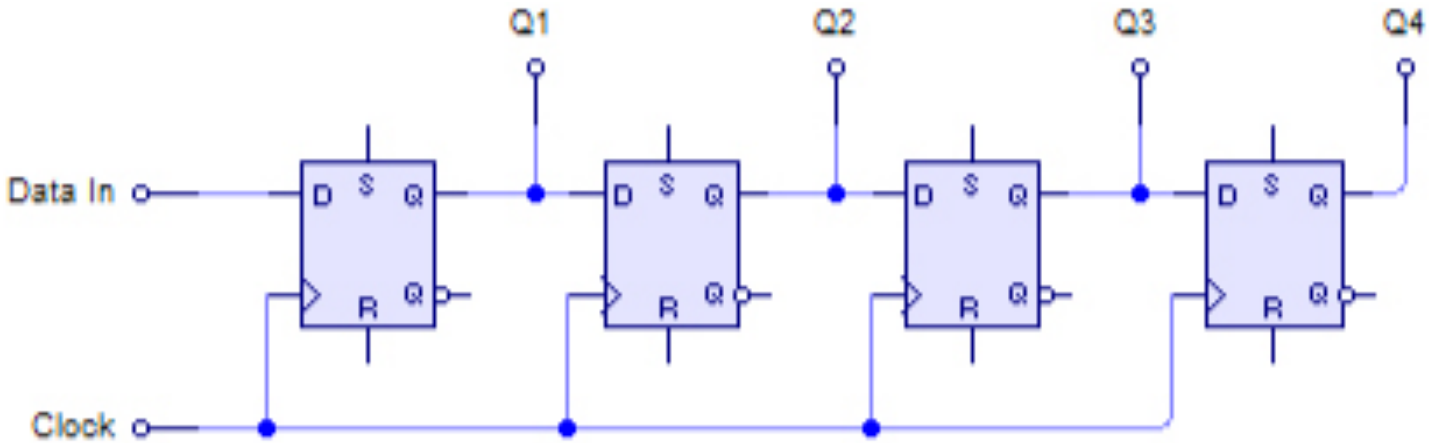
- Most common.
- Input/output of each flip-flop is independent.
- Can be encoded using any type of flip-flop.



SIPO Registers

Serial In, Parallel Out Register: One input bit at a time, output all bits at once.

- Input bit moves through chain of flip-flops.
- Transitions at each clock.



- This example uses D flip-flops.
- Sometimes it is useful to clear the entire register without waiting n cycles for n bits of data to shift out.
- Additional control signals can be used to set all flip-flops to 1 (S) or all flip-flops to 0 (R).

SISO/PISO Registers

Serial In, Serial Out Register: A linear chain of flip-flops.

- Output of one flip-flop is the input of the next.
- One input bit and one output bit.
- Kind of like a conveyor belt of bits.

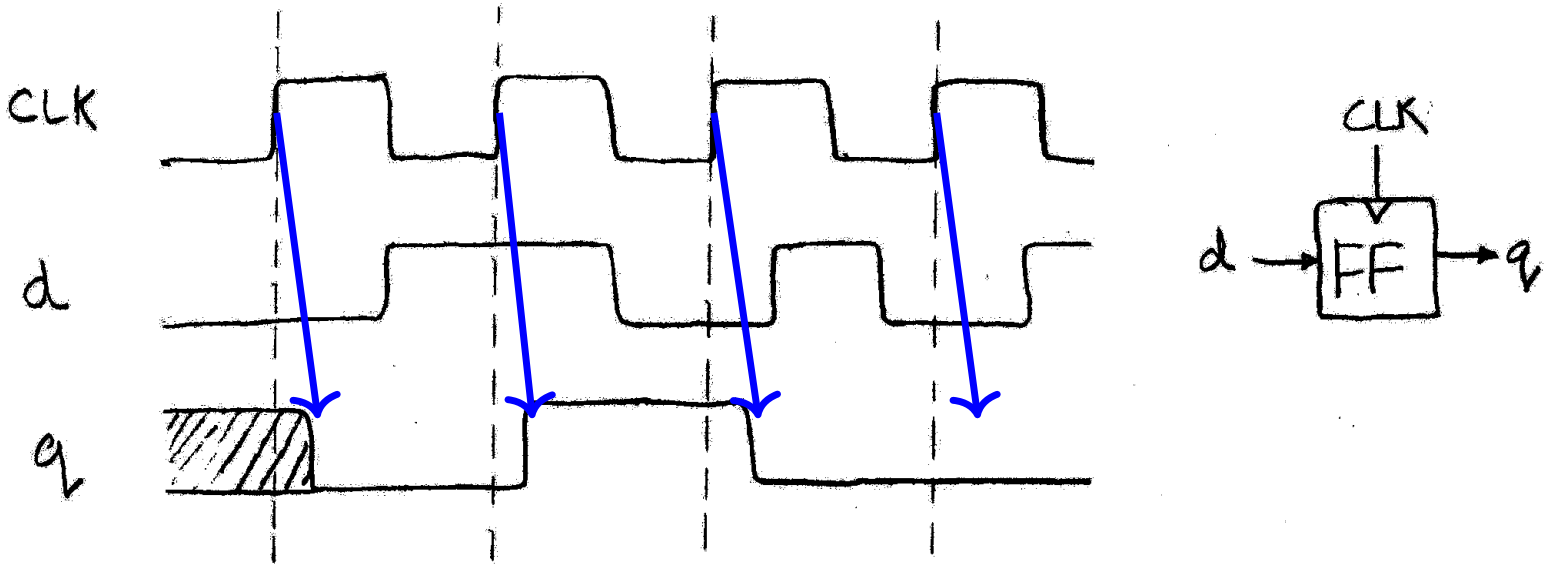
Parallel In, Serial Out Register: A linear chain of flip-flops + control circuits.

- Data *loaded* in parallel: n flip-flops load n bits at once.
- Data *output* in serial: Acts as SISO for output.
  - ↳ Output one bit at a time.
  - ↳ Bits are shifted one over on each output.
- Requires clock and additional write/shift control signal.

Timing a Flip-Flop

All gates/circuits introduce propagation delay.

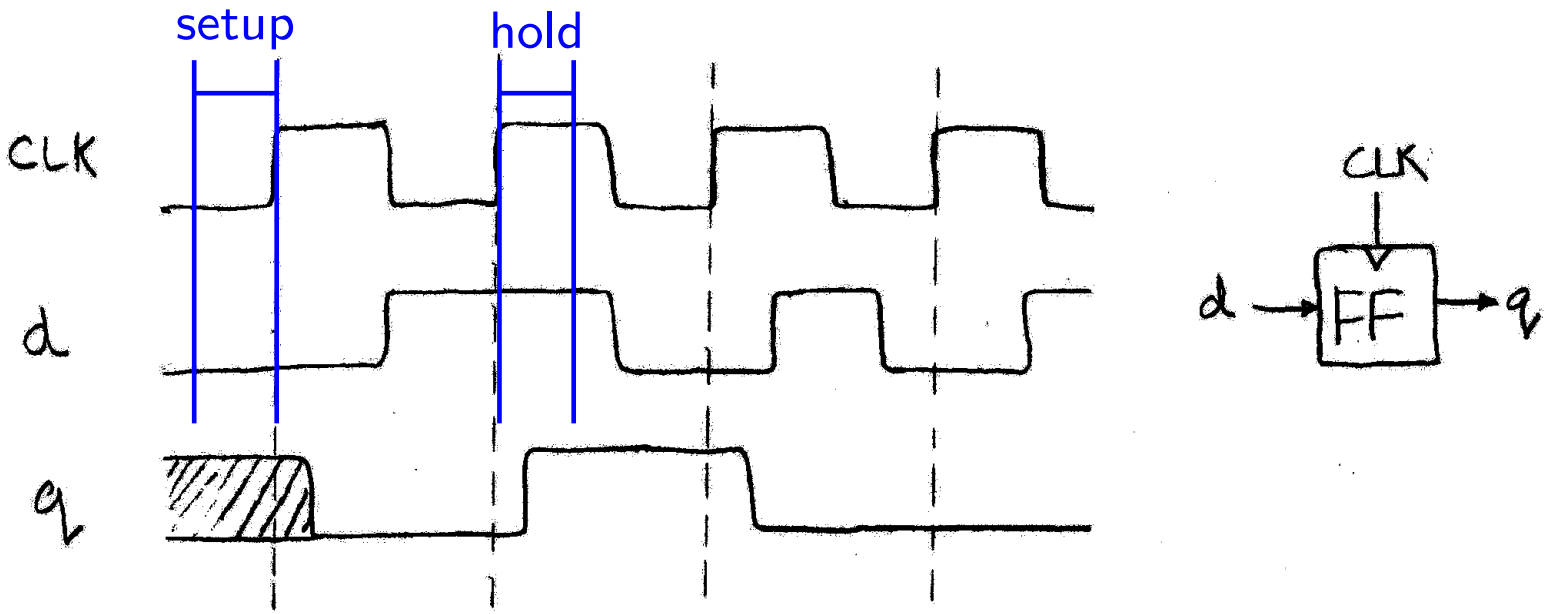
For flip-flops this propagation delay is called **clk-to-q delay**.



Timing a Flip-Flop: Data Stability

Input to a flip-flop must have a stable value around the rising edge of the clock.

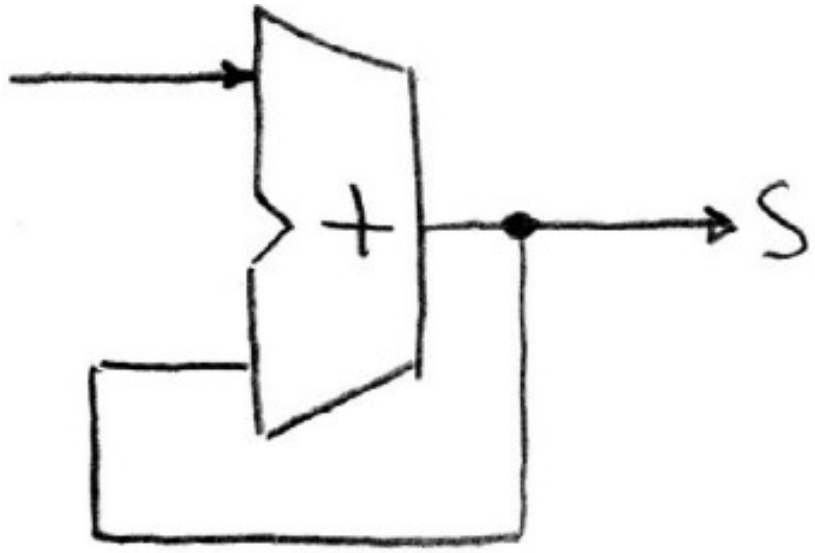
- Before the rising edge: **setup time**.
- After the rising edge: **hold time**.



Despite how it's shown here, hold time is less than clk-to-q delay.

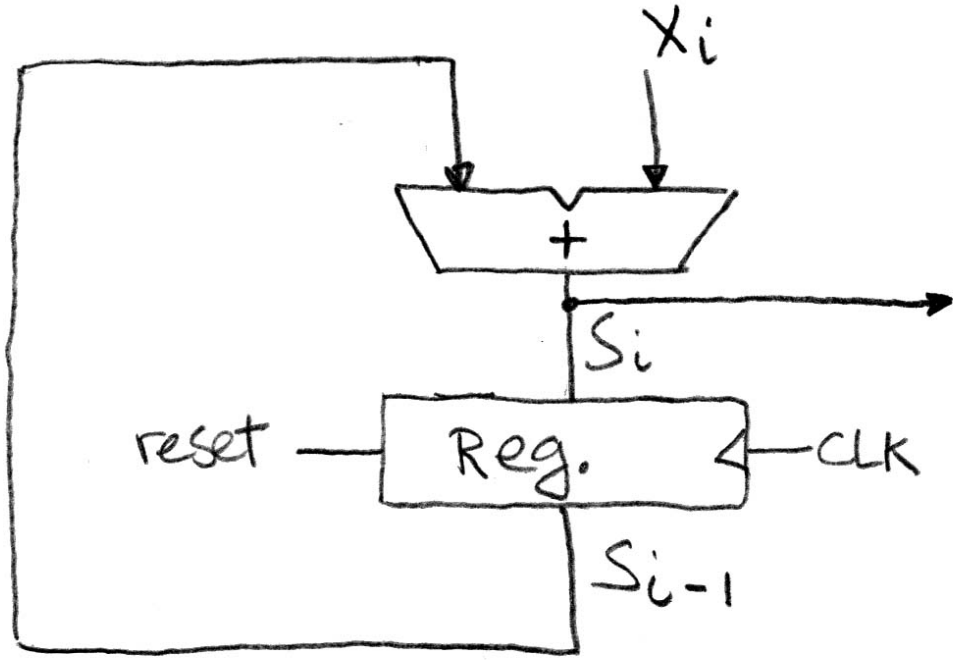
Putting it all Together: Accumulator

An accumulator: continually adds input value to its stored value.



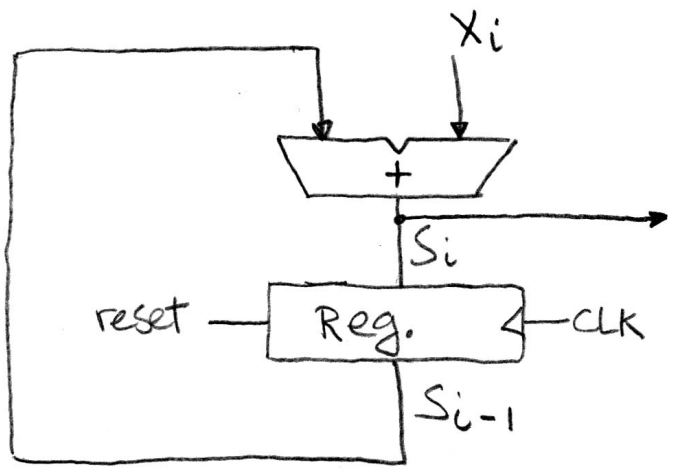
- This doesn't work.
- Would spin once per circuit's propagation delay, not once per input.
- Need clock to **synchronize** reading from input.

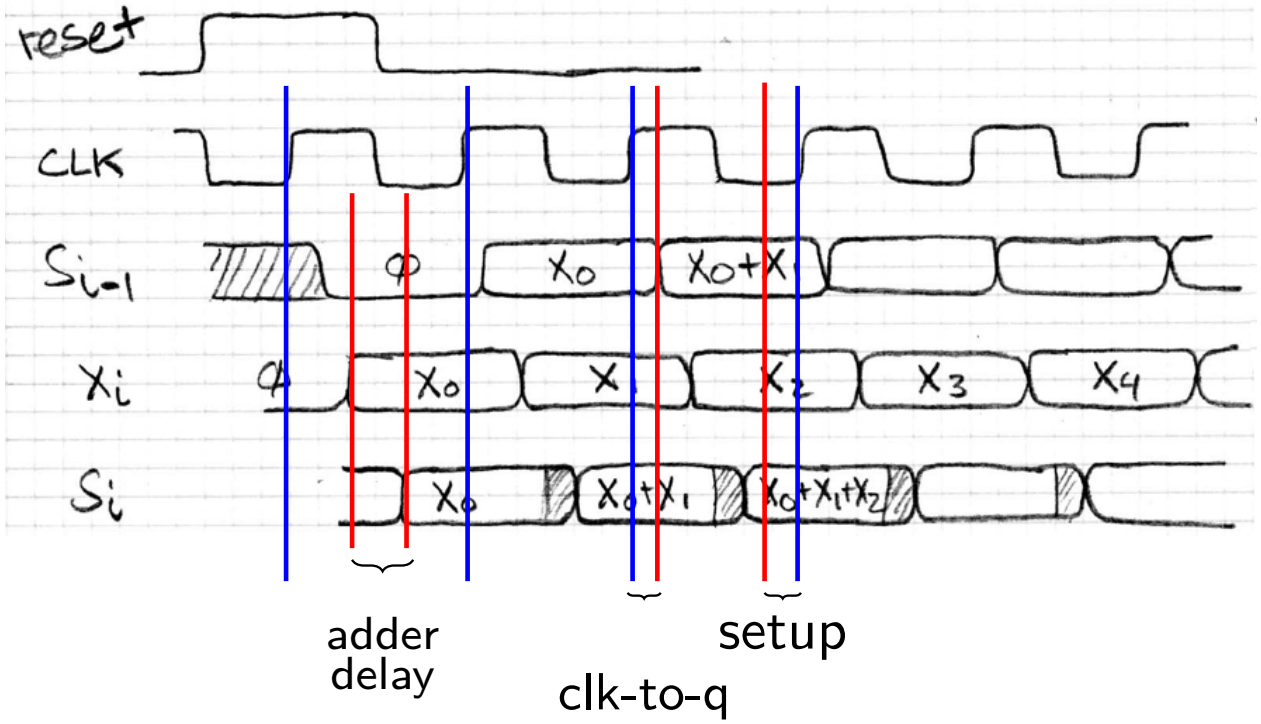
Clocked Accumulator



- Insert register to store output.
- Only need to clock the register, not the combinational circuit.
- Clock on register determines when output of circuit actually gets stored.

Timing the Accumulator

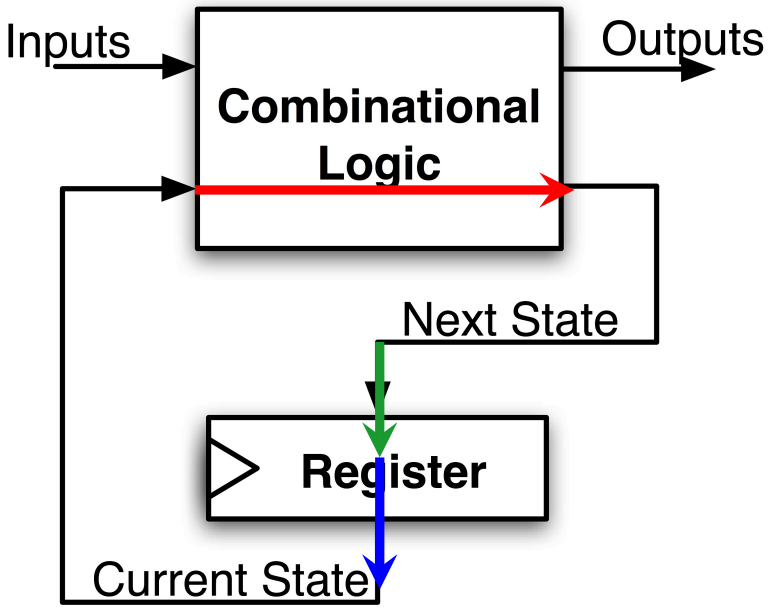




Clock must be slow enough to include:

- Adder delay,
- Clk-to-q,
- Setup time.

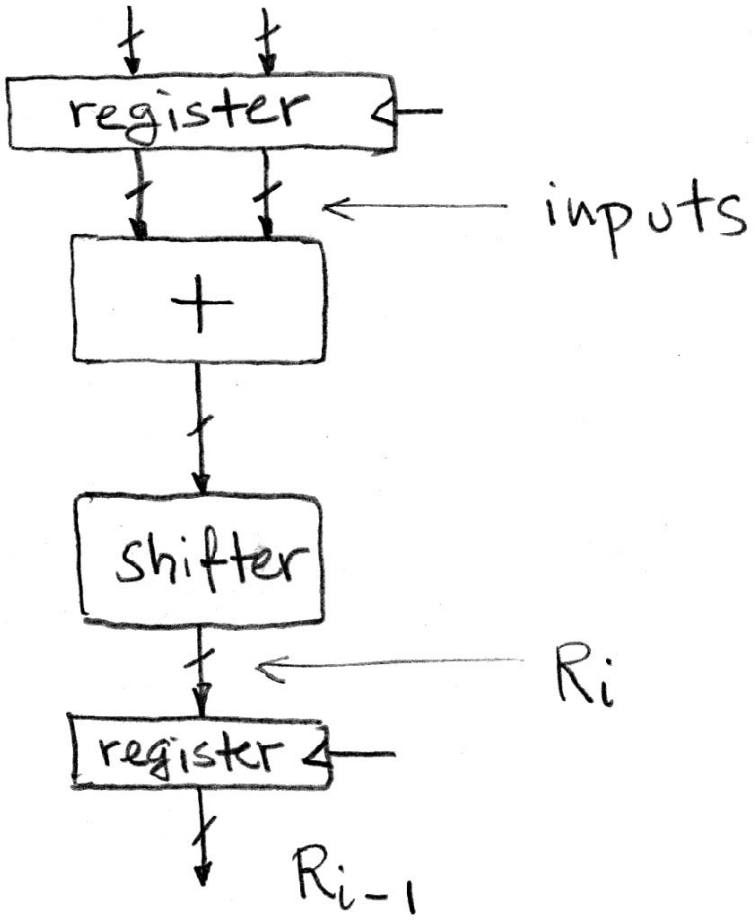
Synchronous Circuits: Clock Frequency

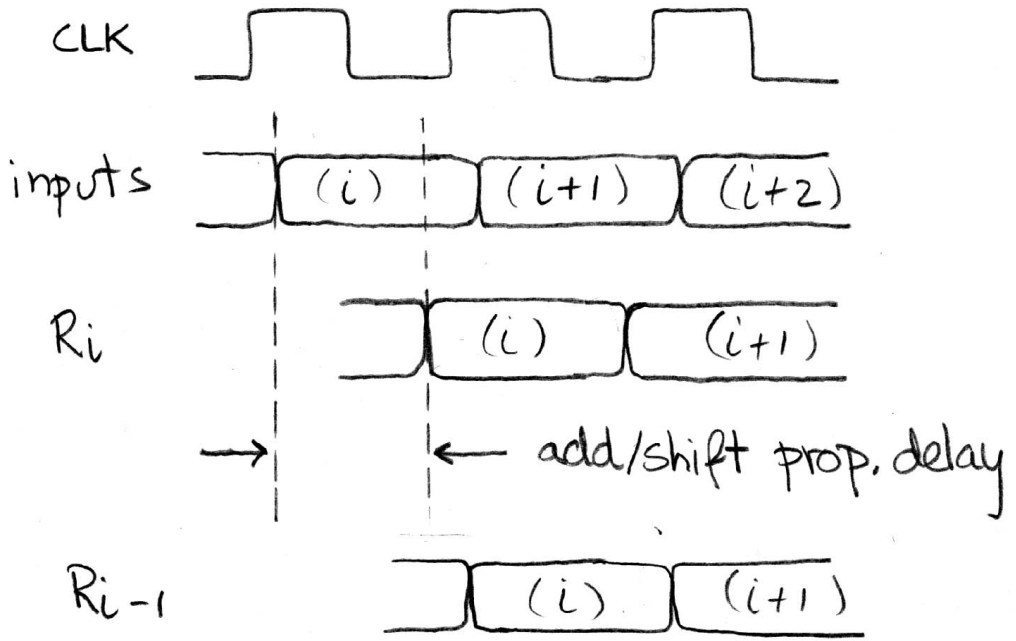


(Max Clock Freq.)

Min. Clock Period = Combinational Circuit Propagation Delay
+ Setup Time
+ Clk-To-Q

Pipeline for Performance (1/2)

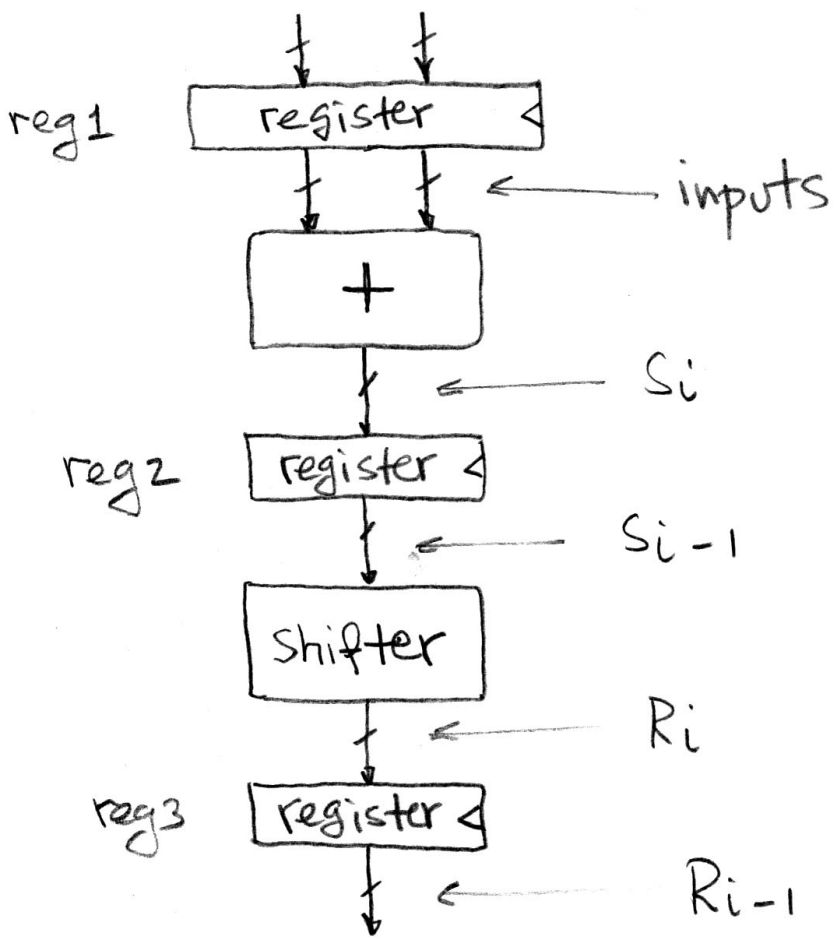


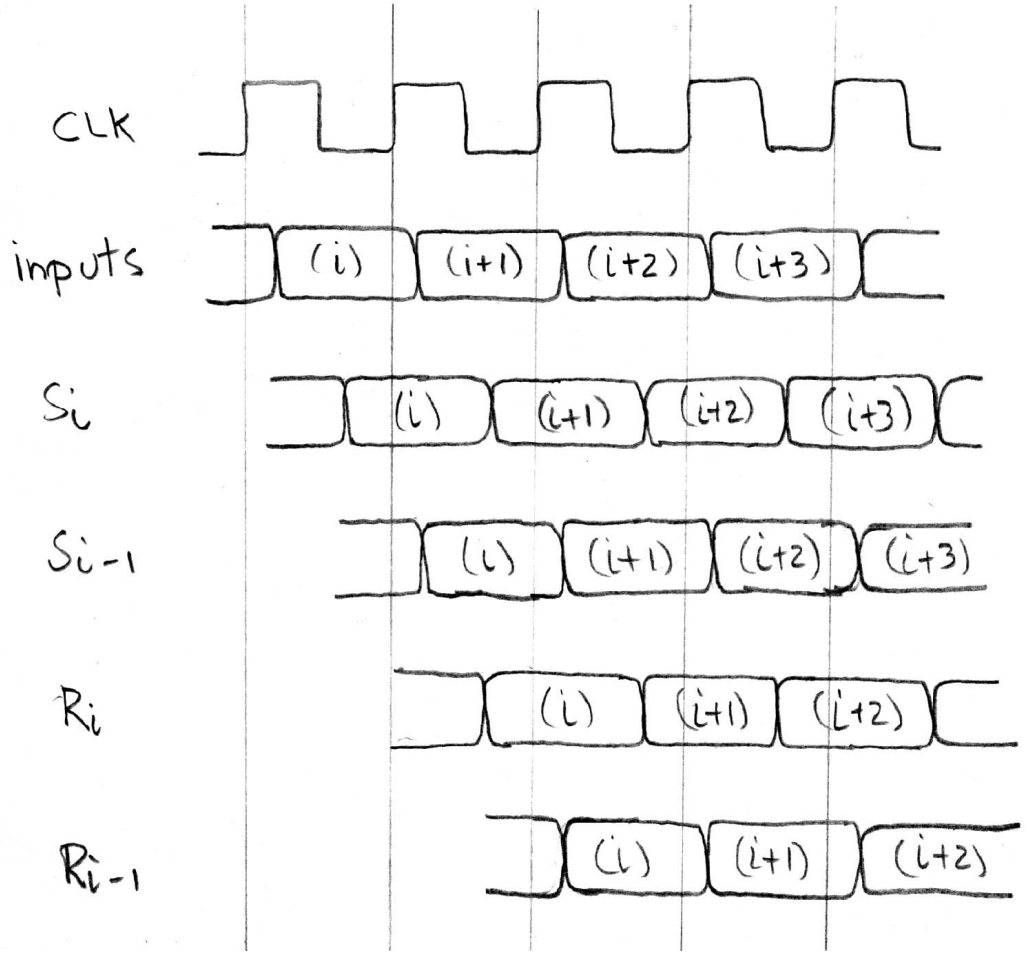


Delay of adder and shifter is very long.

- Forces clock cycle to be very long.
- Slows down other circuits in this synchronous system.

Pipeline for Performance (2/2)

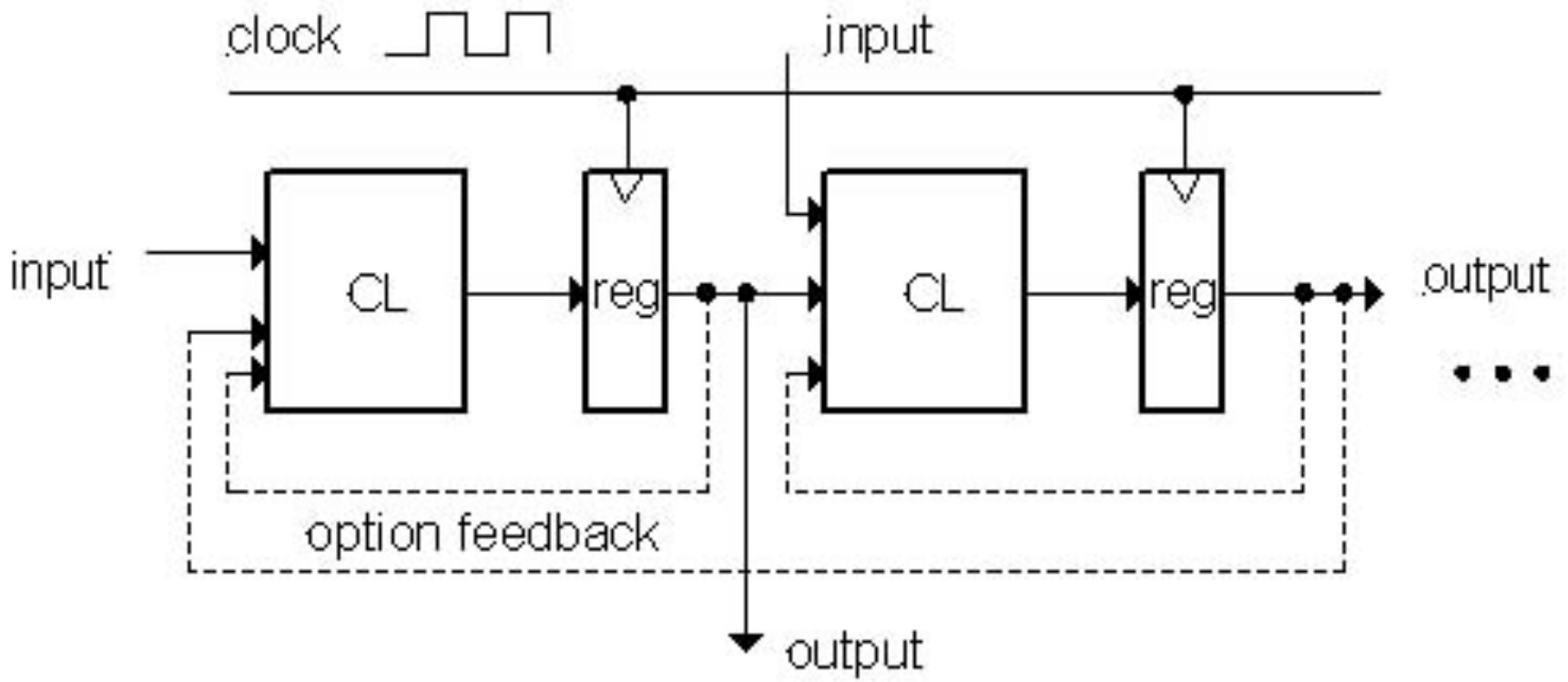




- Split add and shift into two different tasks.
- Insert register between to store results temporarily.
- **Increase clock frequency.**

General Synchronous Systems

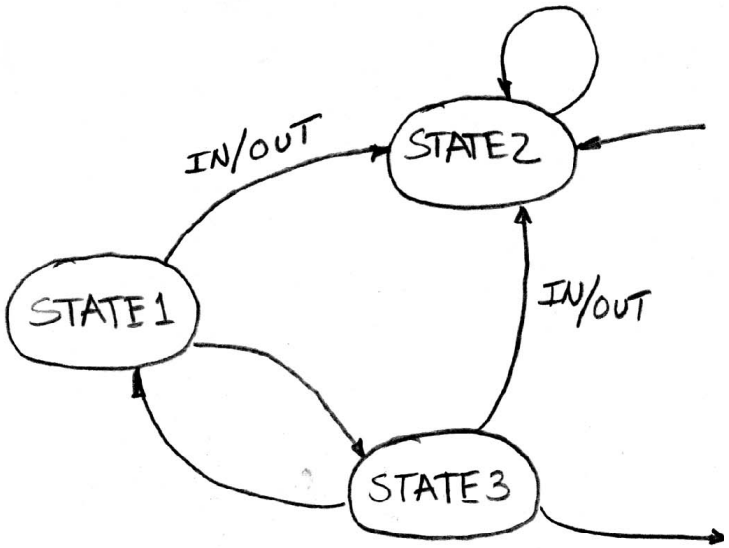
- All systems follow a general pattern:
- A chain of logic circuit blocks, separated by registers, controlled by a single clock.
- Foreshadowing for MIPS pipeline.



Outline

Finite State Machines: Introduction

We know **FSMs** from logic, formal languages, complexity.

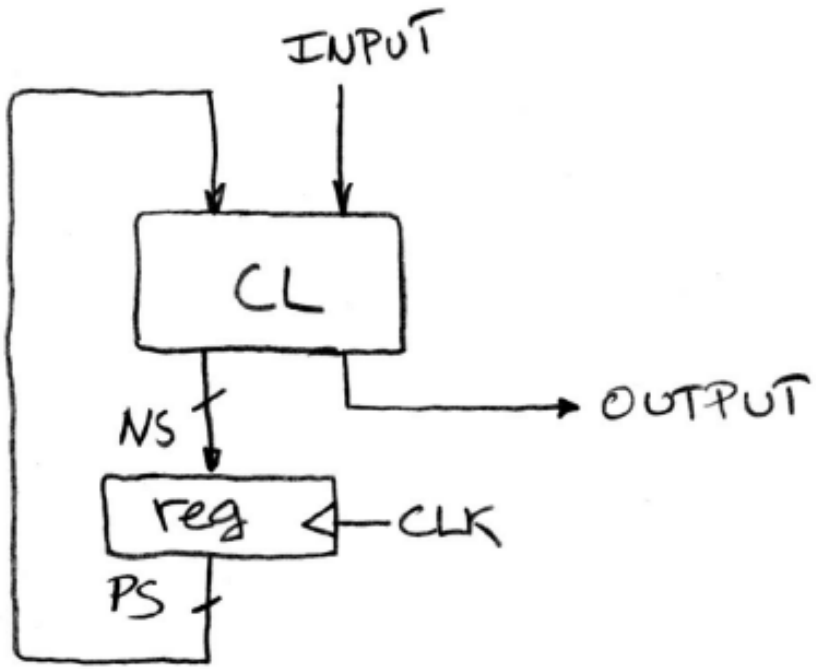


- Each state of the machine is a node.
- Inputs trigger change of state and an output.
- This is a Mealy machine: outputs occur on transitions.
- Moore machines are equivalent.
  - ↳ Output is based on current state.

Finite State Machines: As Circuits

FSMs have three components: state, input, output.

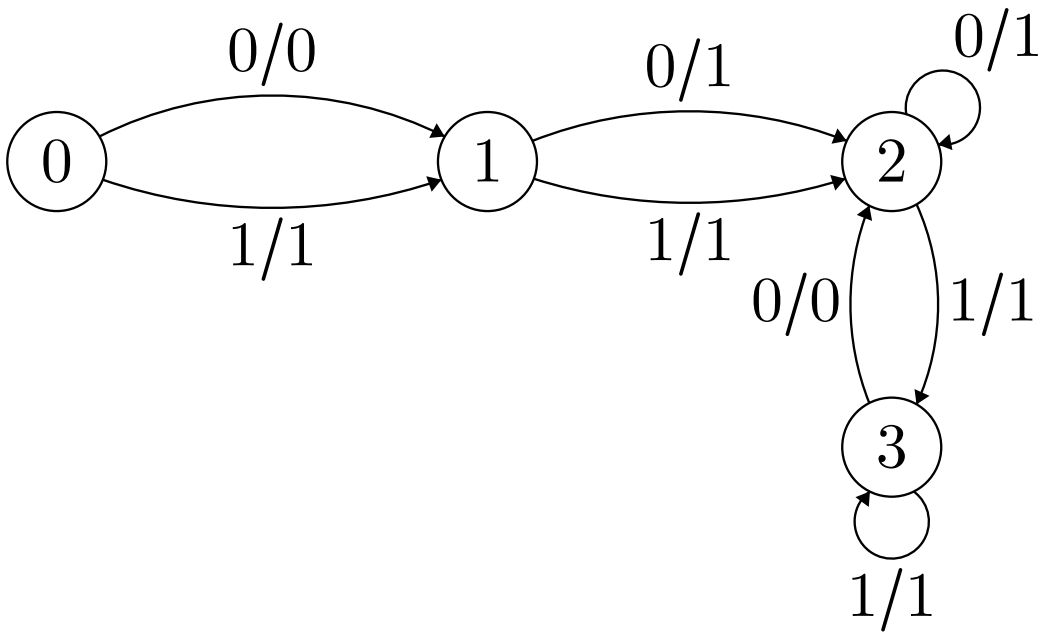
- Just like synchronous circuit.
- Registers, input bits, output bits.
- Clock controls when inputs read ⇒ transitions.
- PS: present state, NS: next state.



Finite State Machines: Implementing The Logic

Next state and output is always just some Boolean combination of input and output. Use our normal 4-step process:

1. Build a truth table,
2. Get canonical form,
3. Simplify,
4. Draw circuit.



PS	In	NS	Out
00	0	01	0
00	1	01	1
01	-	10	1
10	0	10	1
10	1	11	1
11	0	10	0
11	1	11	1

FSM: Implementing The Logic (2/3)

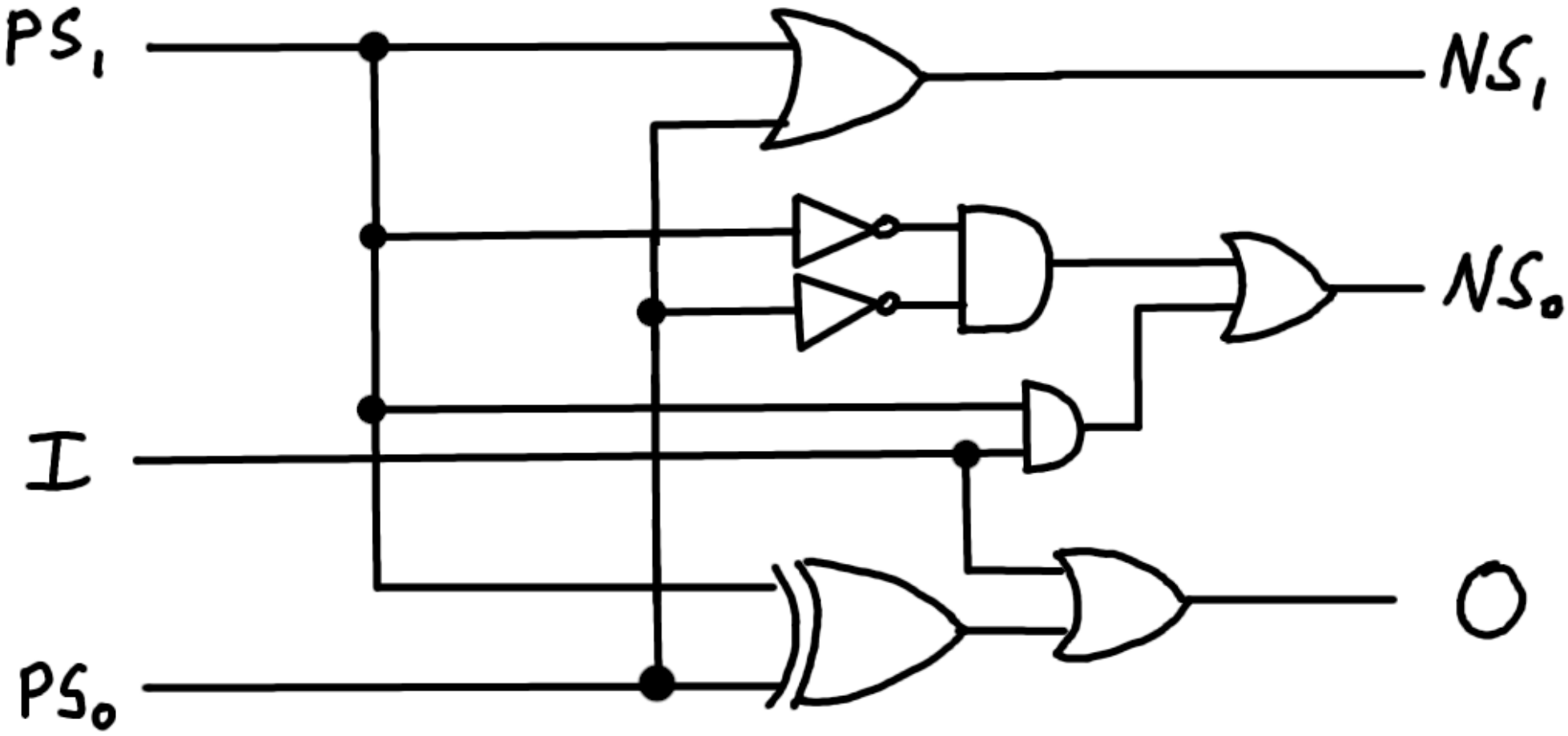
Let PS $= PS_1PS_0$, In $= I$, NS $= NS_1NS_0$, Out $= O$.
From the truth table:

$$
\begin{aligned}
NS_1 &\equiv \Sigma(2,3,4,5,6,7) \\
&\equiv \overline{PS_1}PS_0\overline{I} + \overline{PS_1}PS_0I + PS_1\overline{PS_0I} + PS_1\overline{PS_0}I + PS_1PS_0\overline{I} + PS_1PS_0I \\
&\equiv ... \\
&\equiv PS_1 + PS_0
\end{aligned}
$$

$$
\begin{aligned}
NS_0 &\equiv \Sigma(0,1,5,7) \\
&\equiv \overline{PS_1PS_0I} + \overline{PS_1PS_0}I + PS_1\overline{PS_0}I + PS_1PS_0I \\
&\equiv ... \\
&\equiv \overline{PS_1PS_0} + PS_1I
\end{aligned}
$$

$$
\begin{aligned}
O &\equiv \Sigma(1,2,3,4,5,7) \\
&\equiv \overline{PS_1PS_0}I + \overline{PS_1}PS_0\overline{I} + \overline{PS_1}PS_0I + PS_1\overline{PS_0I} + PS_1\overline{PS_0}I + PS_1PS_0I \\
&\equiv I + \overline{PS_1}PS_0 + PS_1\overline{PS_0} \\
&\equiv I + (PS_1 \oplus PS_0)
\end{aligned}
$$

FSM: Implementing The Logic (3/3)



FSMs are Synchronous Systems are FSMs

- Essentially every synchronous system can be modelled by an FSM.
  - ↳ Would become absurdly large in most circumstances.
- A valid design strategy for integrated circuits and specialized hardware includes:
  1. Turn problem into FSM.
  2. Turn FSM into truth table.
  3. Turn truth table into circuit.
- Full Example: An elevator-controlling circuit.
  - ↳ `https://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf`