Sequential Logic Design

- Latches and Flip-Flops
- Synchronous Logic Design
- Finite State Machines
- Timing of Sequential Logic
- Parallelism

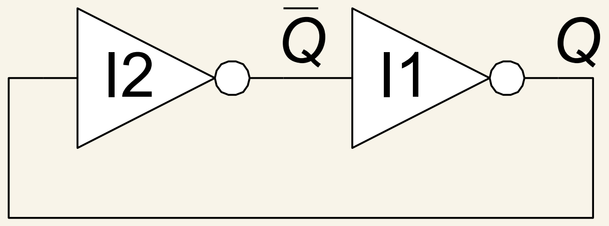
Chap 3

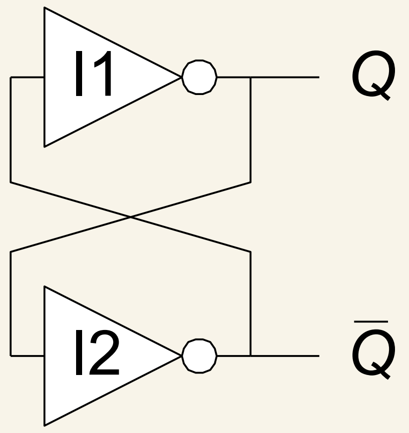
Sequential Circuits

- Outputs of sequential logic depend on current and prior input values – it has memory
  - Use feedback from output to input to store information
- Some definitions:
  - State : all the information about a circuit necessary to explain its future behavior
  - Latches and flip-flops : state elements that store one bit of state
    - » Bistable circuit
    - » SR Latch
    - » D Latch
    - » D Flip-flop
  - Synchronous sequential circuits : combinational logic followed by a bank of flip-flops

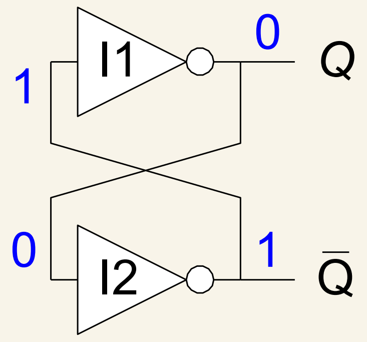
Bistable Circuit

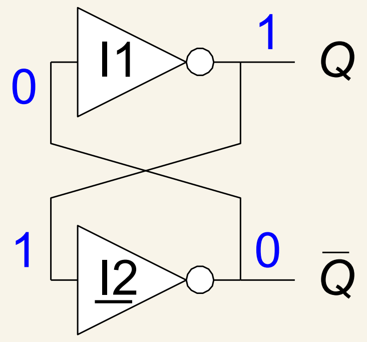
- Fundamental building block of other state elements
- Two outputs: Q, $\overline{Q}$
- No inputs to control the state





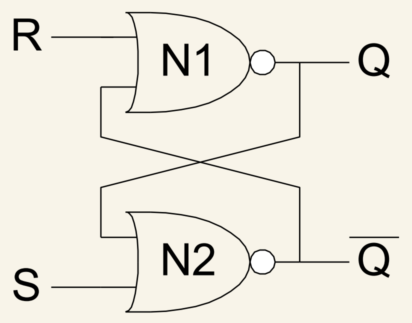
- Consider the two possible cases :
  - Q = 0:
    - » then Q = 0, $\overline{Q}$ = 1 (consistent)
  - Q = 1:
    - » then Q = 1, $\overline{Q}$ = 0 (consistent)

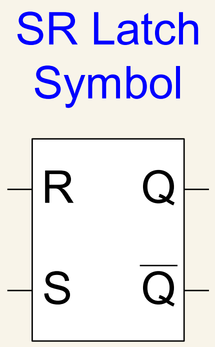


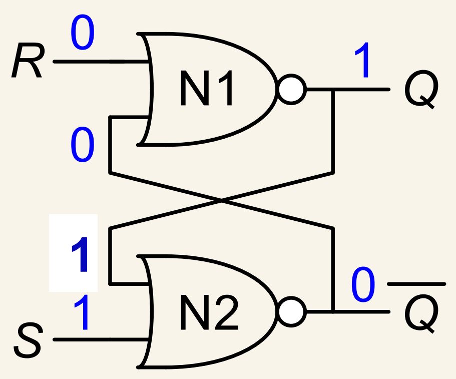


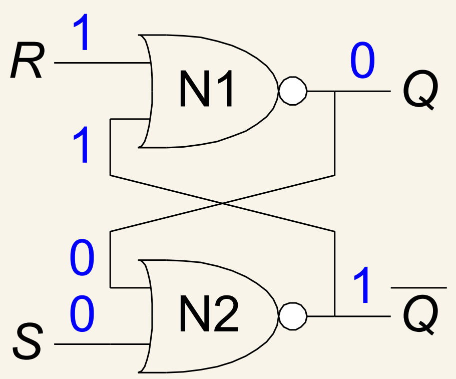
SR (Set/Reset) Latch

- SR Latch

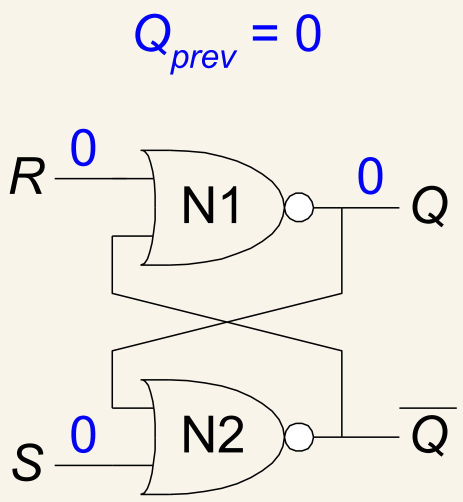


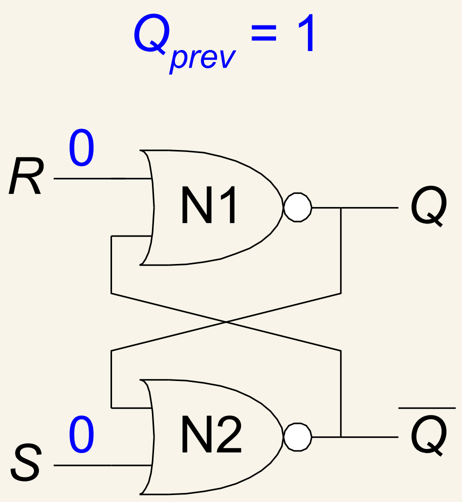


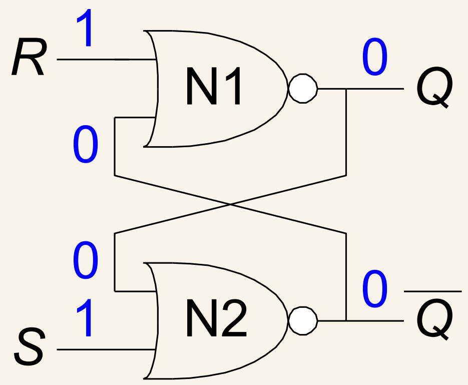




- Consider the four possible cases :
  - S = 1, R = 0
    - » Set the output
  - S = 0, R = 1
    - » Reset the output
  - S = 0, R = 0
    - » Q = Q_{prev} : Memory
  - S = 1, R = 1
    - » Q = 0, $\overline{Q}$ = 0 : Invalid state



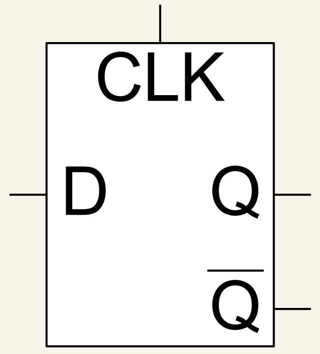


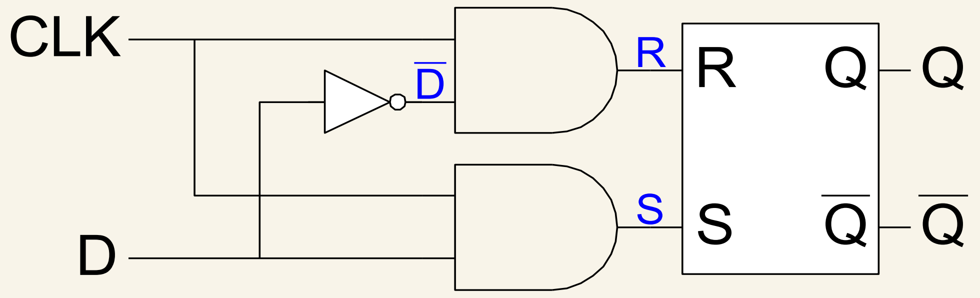


D Latch

- Two inputs : CLK, D
  - CLK: controls when the output changes
  - D (the data input): controls what the output changes to
- Function
  - When CLK = 1,
    - » D passes through to Q (transparent)
  - When CLK = 0,
  - Q holds its previous value (opaque)
- Avoids invalid case

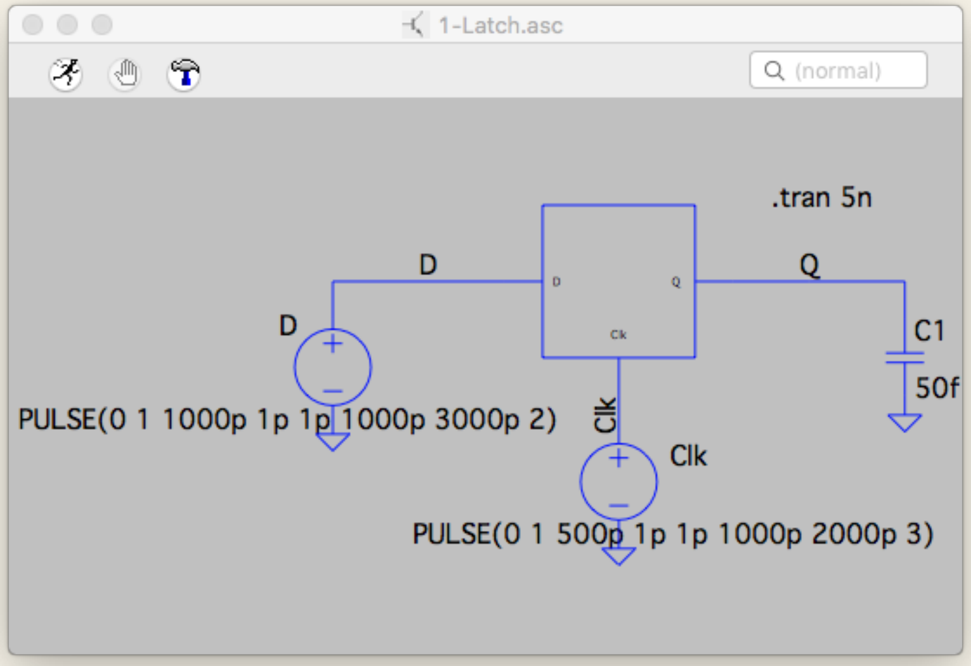
D Latch Symbol

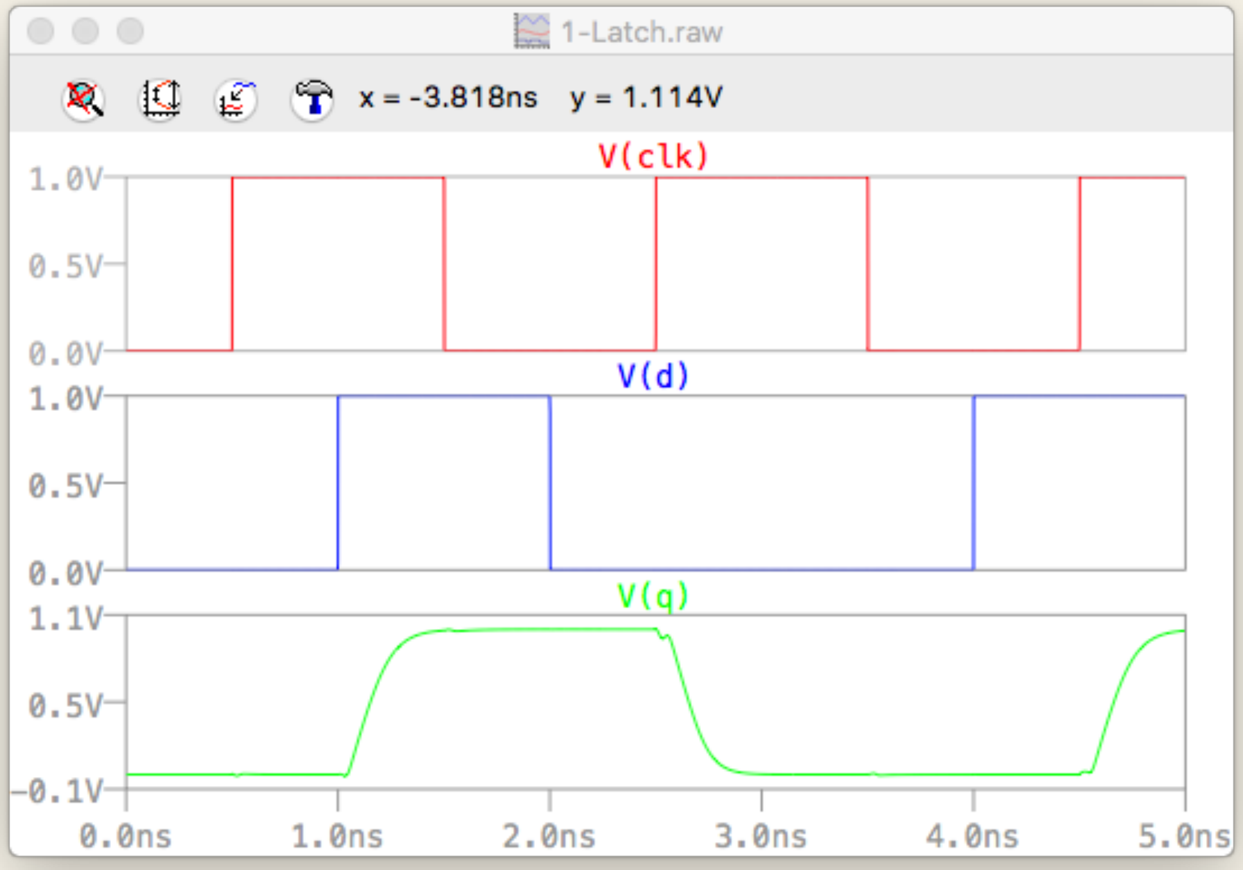


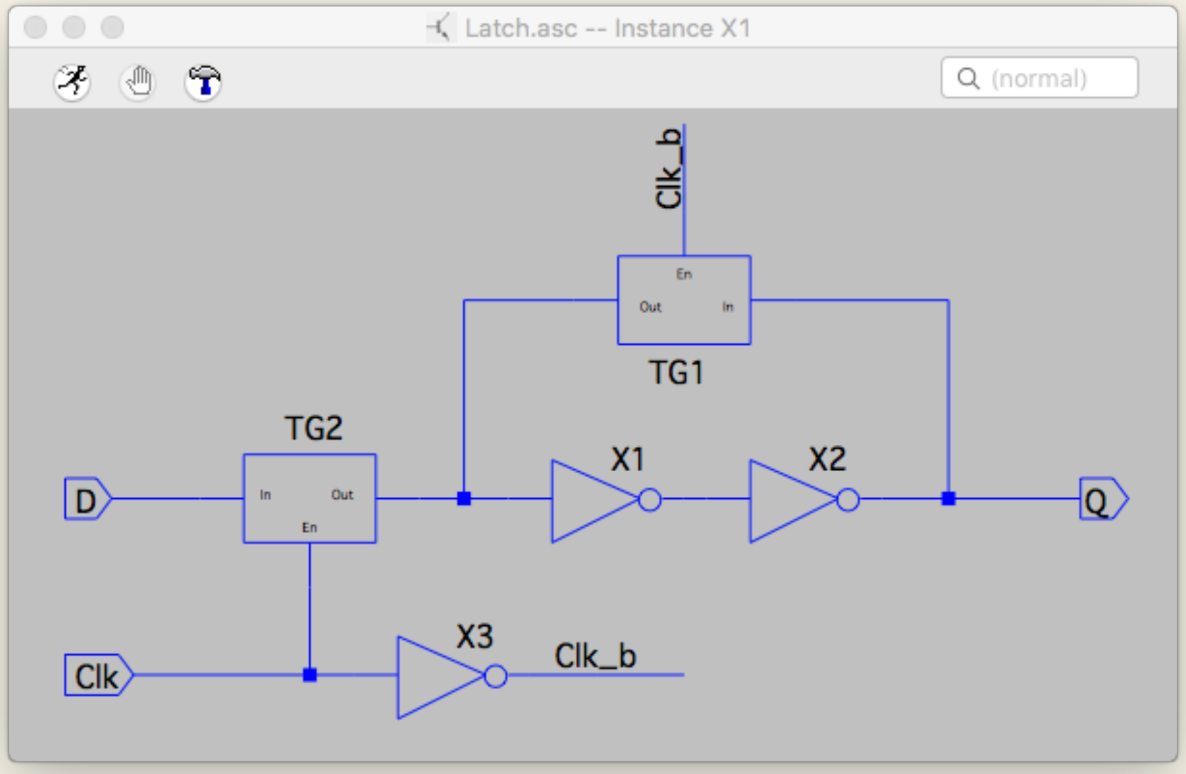


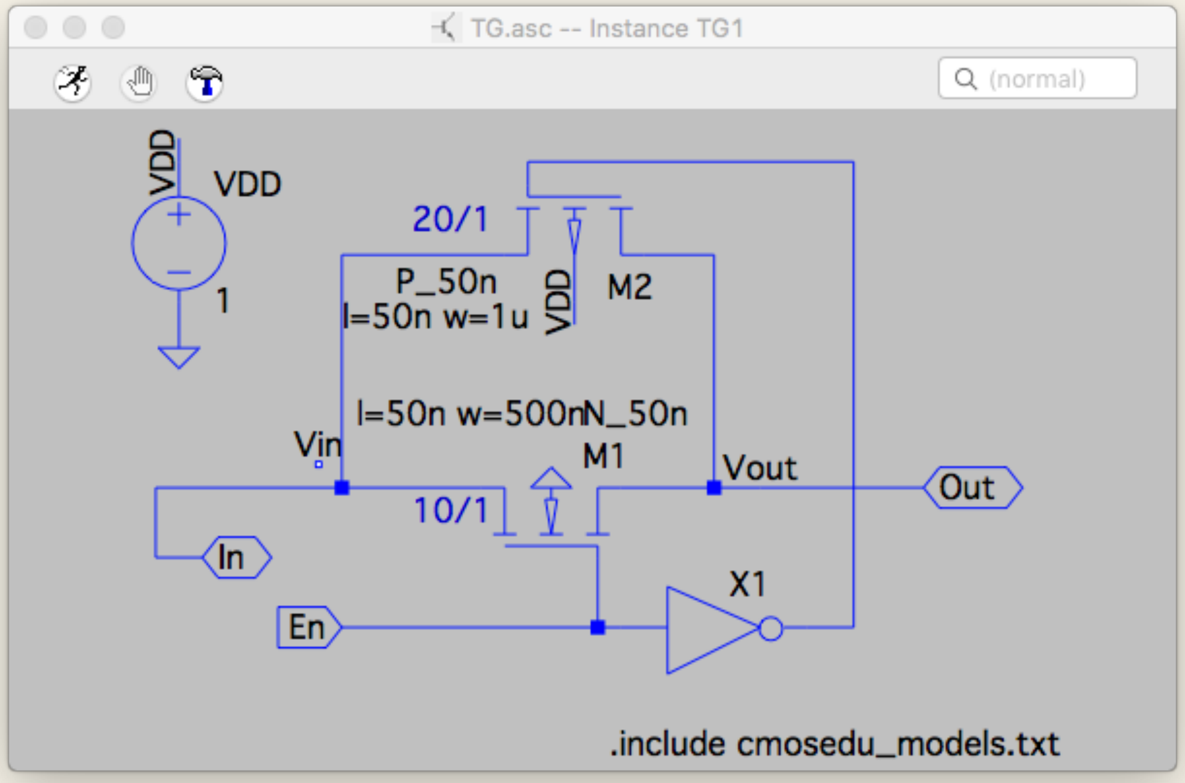
CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

- D Latch CMOS Implementation


1-Latch.asc
(normal)
.tran 5n
D
Q
D
Q
Ck
D
C1
50f
PULSE(0 1 1000p 1p 1p 1000p 3000p 2)
Clk
Clk
PULSE(0 1 500p 1p 1p 1000p 2000p 3)



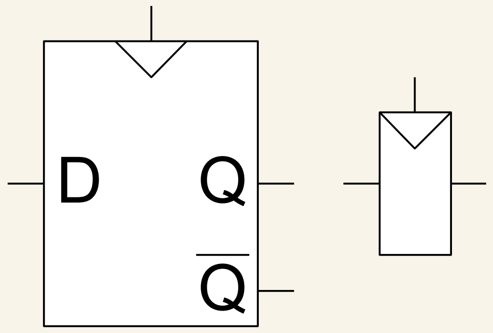




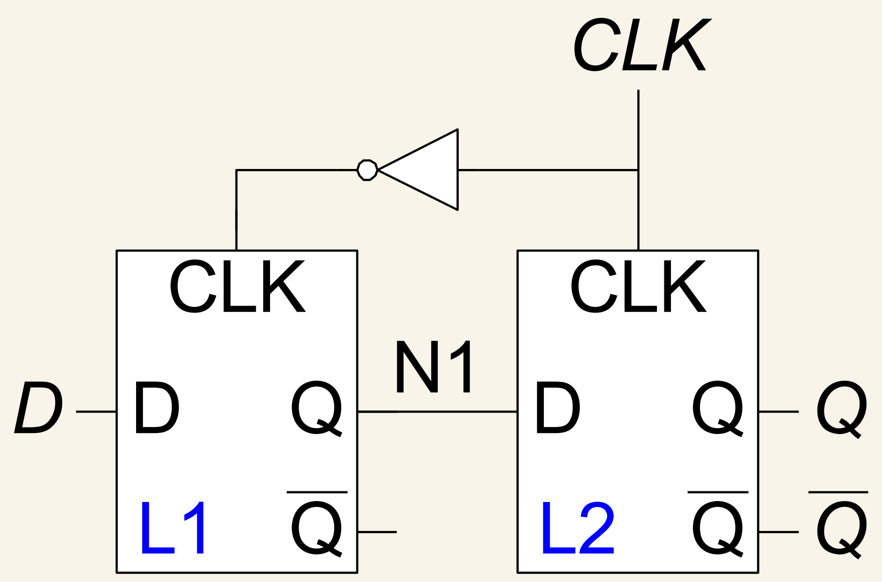
D Flip-Flop

- Inputs: CLK, D
- Function
  - Samples D on rising edge of CLK
    - When CLK rises from 0 to 1, D passes through to Q
    - Otherwise, Q holds its previous value
  - Q changes only on rising edge of CLK
- Called edge-triggered
- Activated on the clock edge

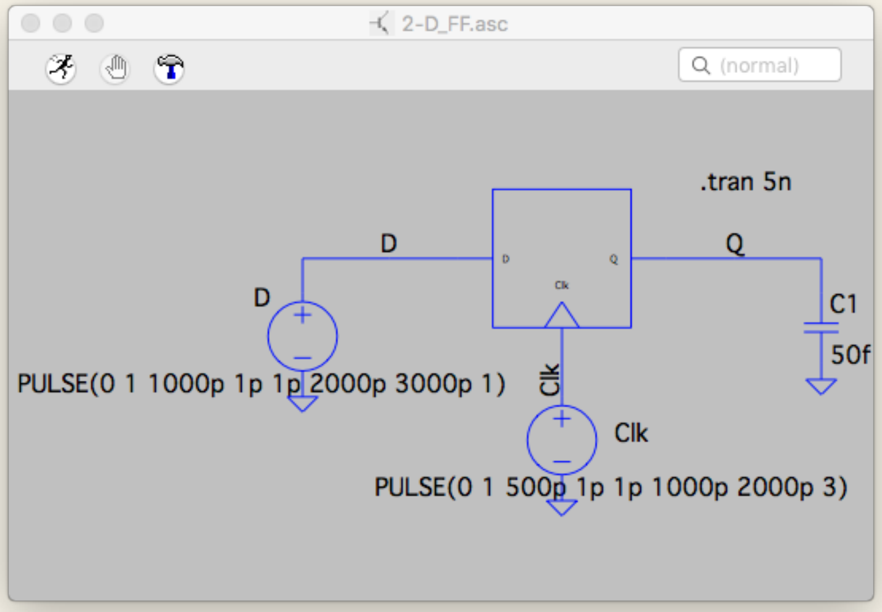
D Flip-Flop Symbols

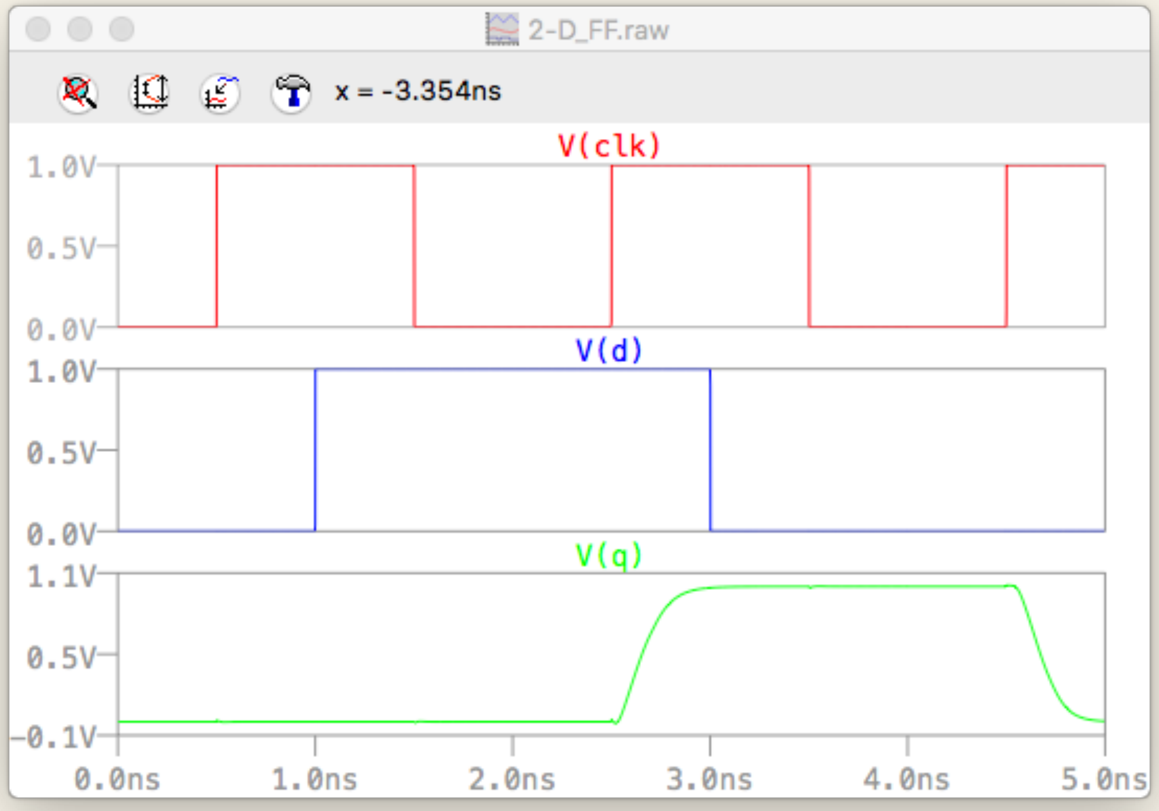


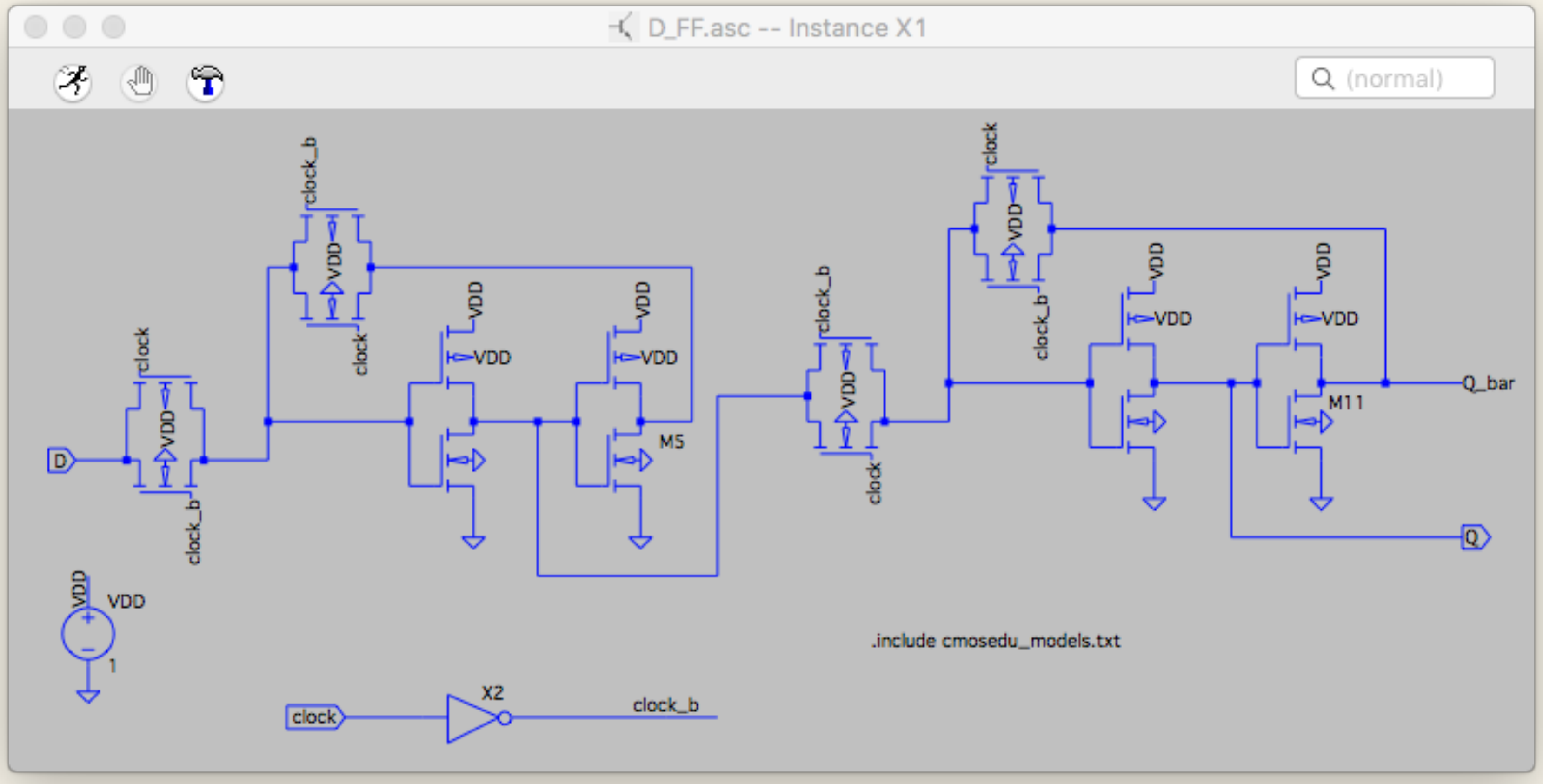
- D Flip-Flop Internal Circuit
  - Two back-to-back latches (L1 and L2) controlled by complementary clocks
    - » When CLK = 0
      - L1 is transparent
      - L2 is opaque
      - D passes through to N1
    - » When CLK = 1
      - L2 is transparent
      - L1 is opaque
      - N1 passes through to Q
  - Thus, on the edge of the clock (when CLK rises from 0   1)
  - D passes through to Q



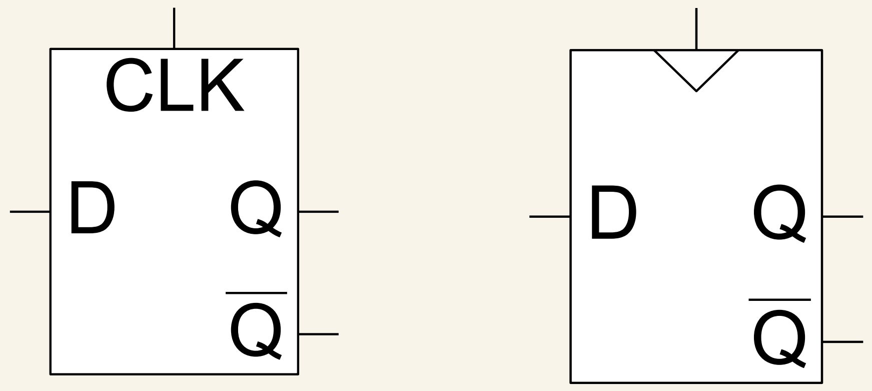
- D Flip-Flop CMOS Implementation

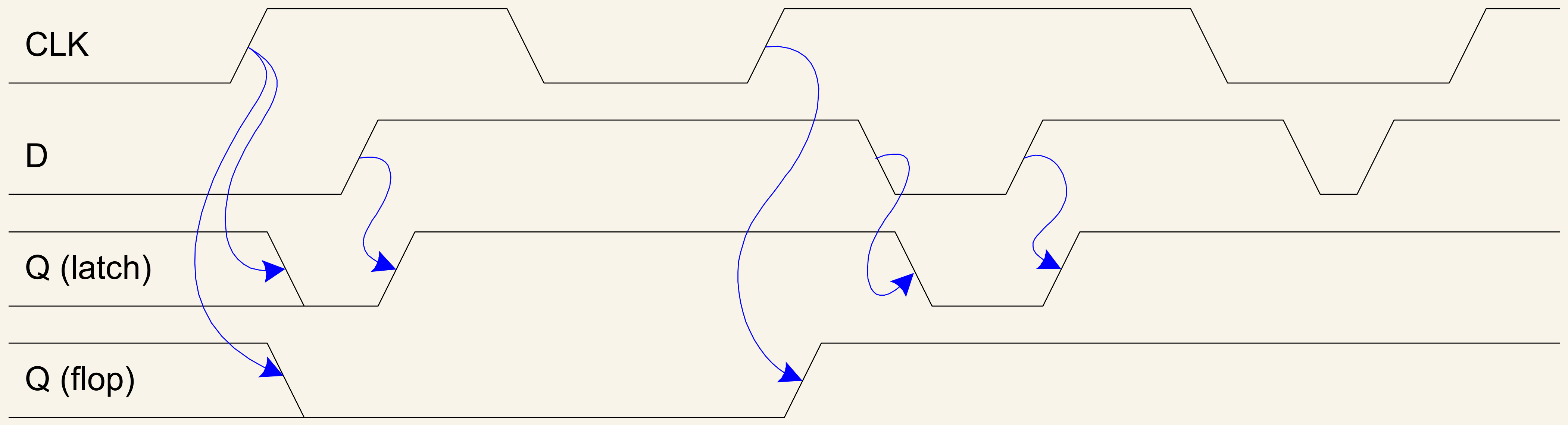

2-D_FF.asc
(normal)
.tran 5n
D
Q
D
C1
50f
PULSE(0 1 1000p 1p 1p 2000p 3000p 1)
Clk
Clk
PULSE(0 1 500p 1p 1p 1000p 2000p 3)



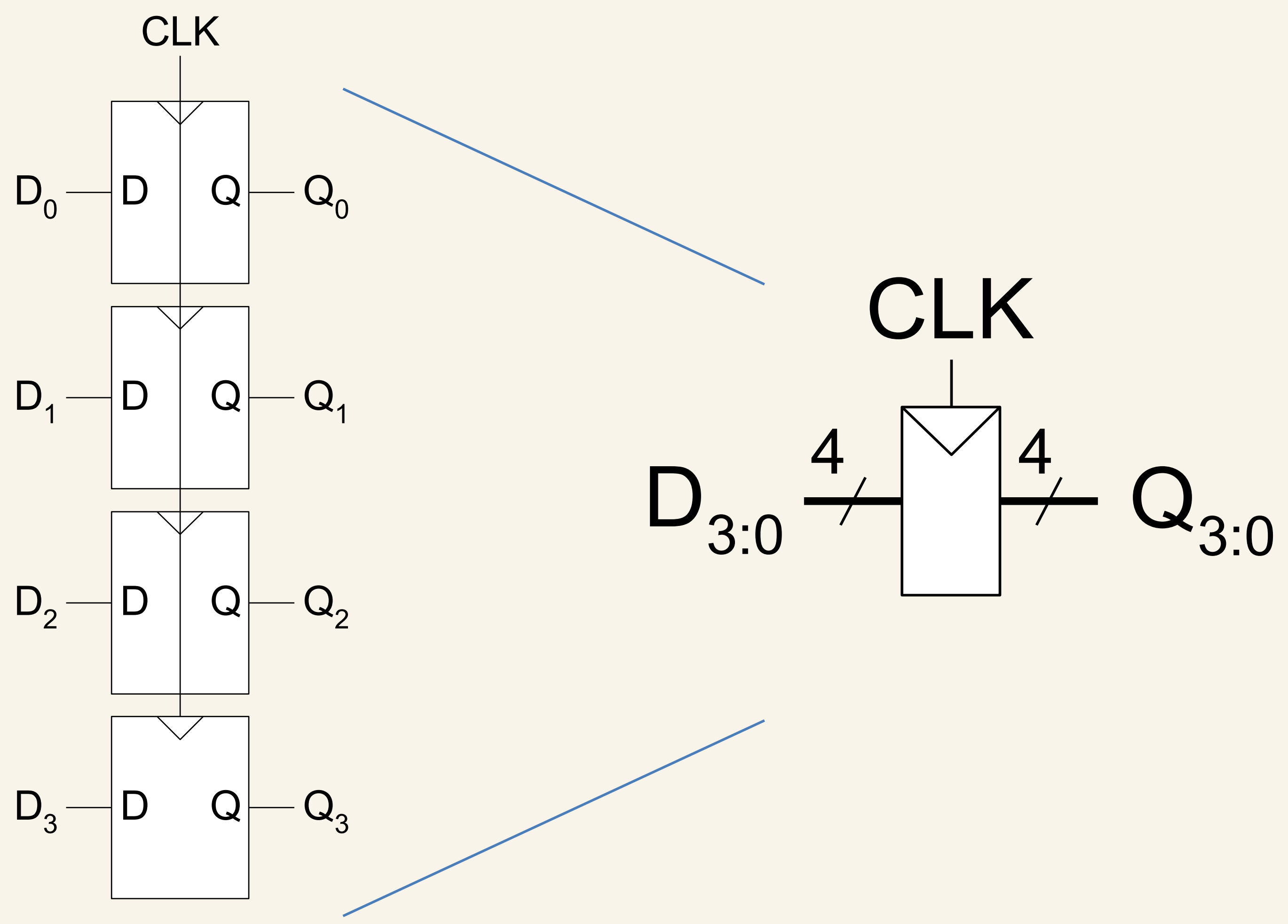


D Latch vs. D Flip-Flop



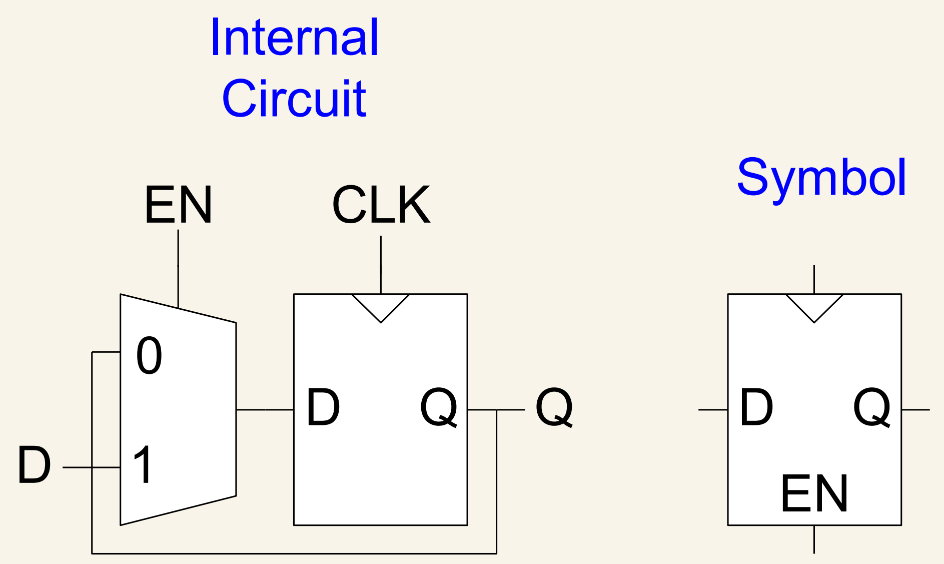


Registers: Multi-bit Flip-Flop



Enabled Flip-Flops

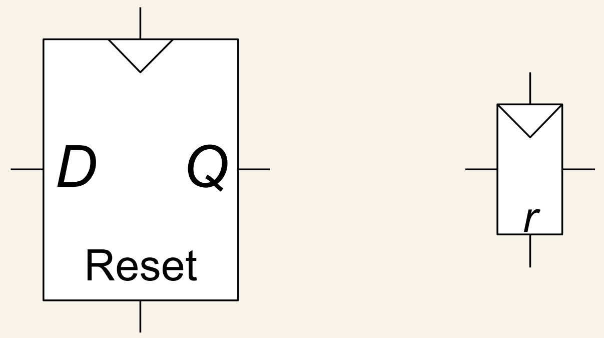
- Inputs: CLK, D, EN
  - The enable input (EN) controls when new data (D) is stored
- Function
  - EN = 1 : D passes through to Q on the clock edge
  - EN = 0 : the flip-flop retains its previous state



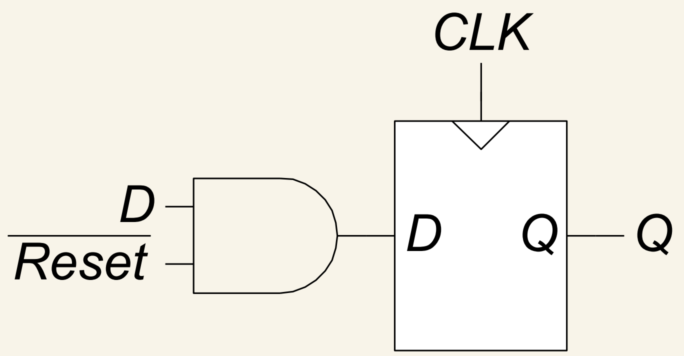
Resettable Flip-Flops

- Inputs : CLK, D, Reset
  - Reset = 1 : Q is forced to 0
  - Reset = 0 : flip-flop behaves as ordinary D flip-flop
- Two types :
  - Asynchronous : resets immediately when Reset = 1
    - » requires changing the internal circuitry of the flip-flop
  - Synchronous : resets at the clock edge only

Symbols



Internal Circuit

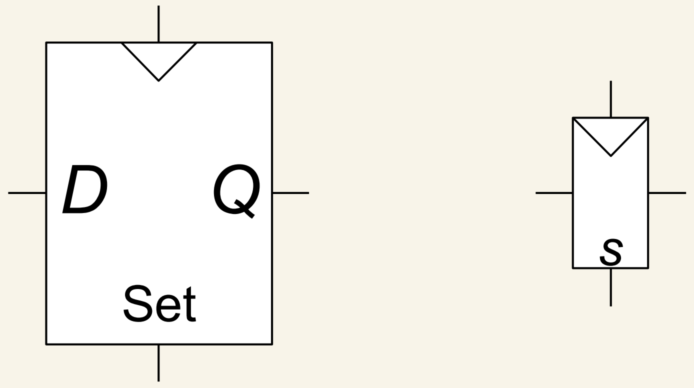


Settable Flip-Flops

- Inputs : CLK, D, Set

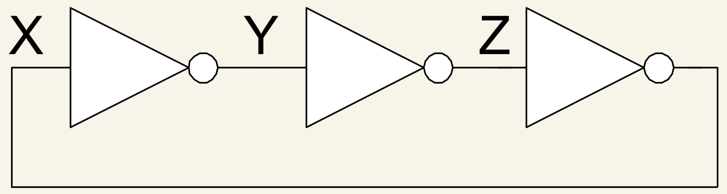
- Function :
  - Set = 1: Q is set to 1
  - Set = 0: the flip-flop behaves as ordinary D flip-flop

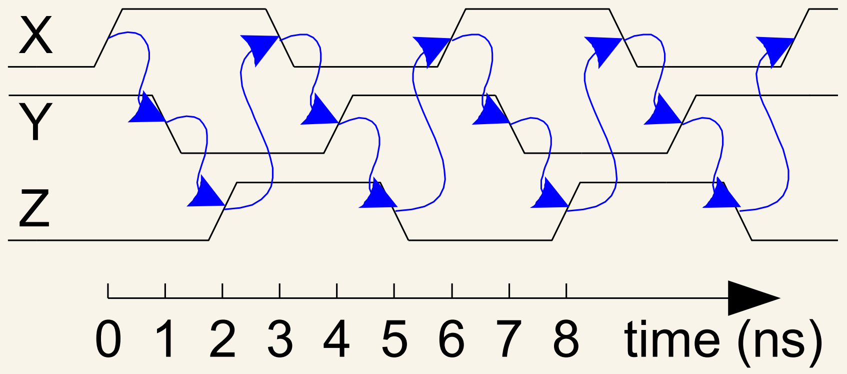
Symbols



Sequential Logic

- Sequential circuits: all circuits that aren’t combinational
- A problematic circuit:





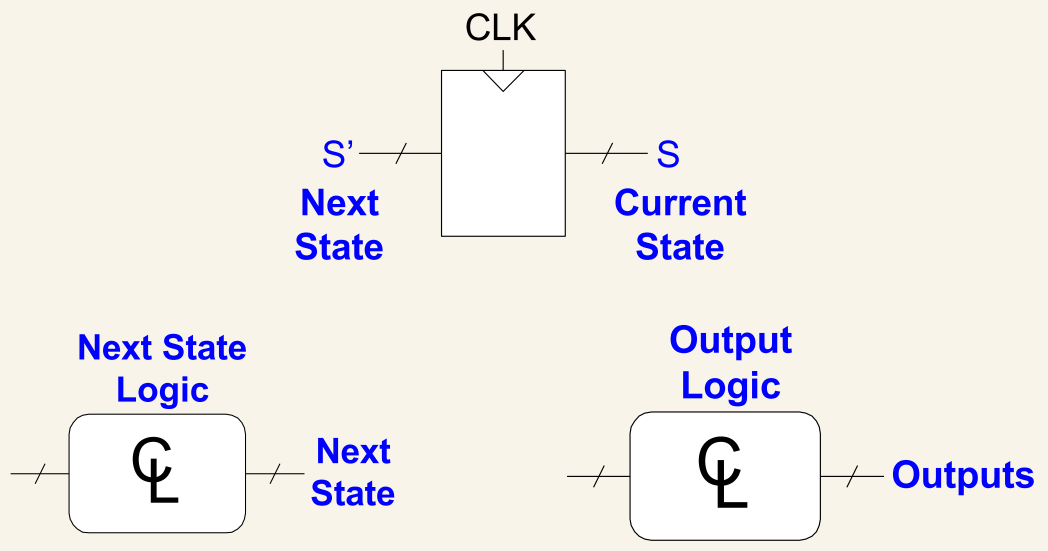
  » No inputs and 1-3 outputs
  » Astable circuit, oscillates
  » Period depends on inverter delay
  » It has a cyclic path: output fed back to input

Synchronous Sequential Logic Design

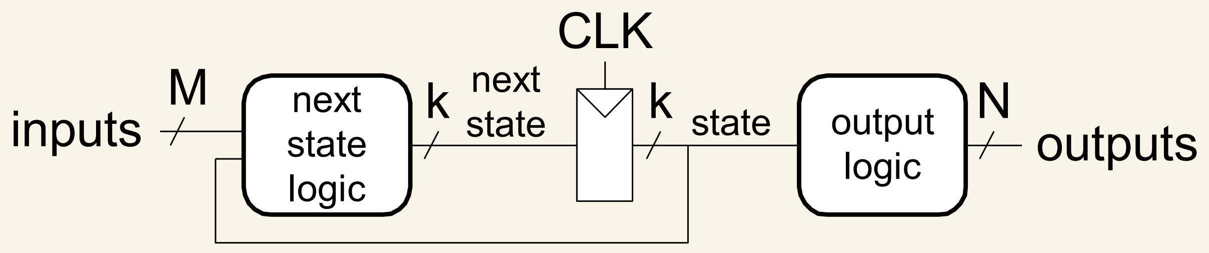
- Breaks cyclic paths by inserting registers
- Registers contain state of the system
- State changes at clock edge : system synchronized to the clock
- Rules of synchronous sequential circuit composition
  - » Every circuit element is either a register or a combinational circuit
  - » At least one circuit element is a register
  - » All registers receive the same clock signal
  - » Every cyclic path contains at least one register
- Two common synchronous sequential circuits
  - » Finite State Machines (FSMs)
  - » Pipelines

Finite State Machine (FSM)

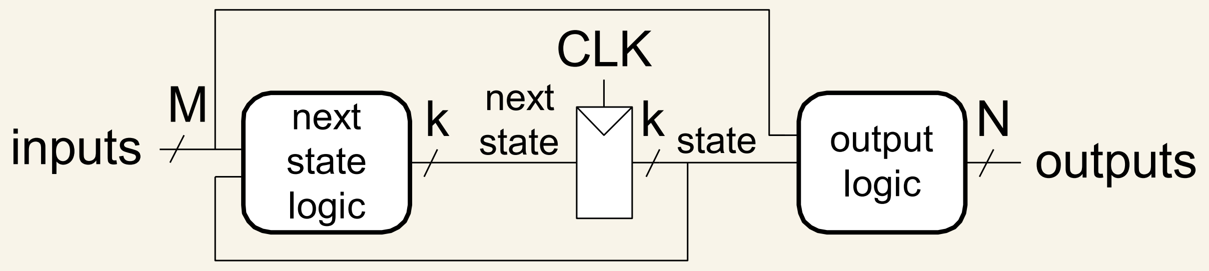
- Consists of:
  - State register
    - » Stores current state
    - » Loads next state at clock edge
  - Combinational logic
    - » Computes the next state
    - » Computes the outputs
- Next state determined by current state and inputs
- Two types of finite state machines differ in output logic:
  - » Moore FSM : outputs depend only on current state
  - » Mealy FSM : outputs depend on current state and inputs



Moore FSM



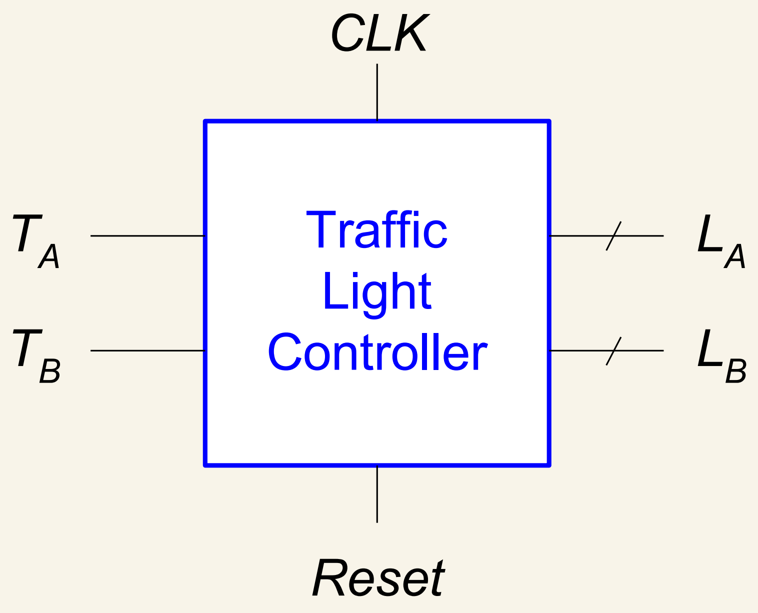
Mealy FSM

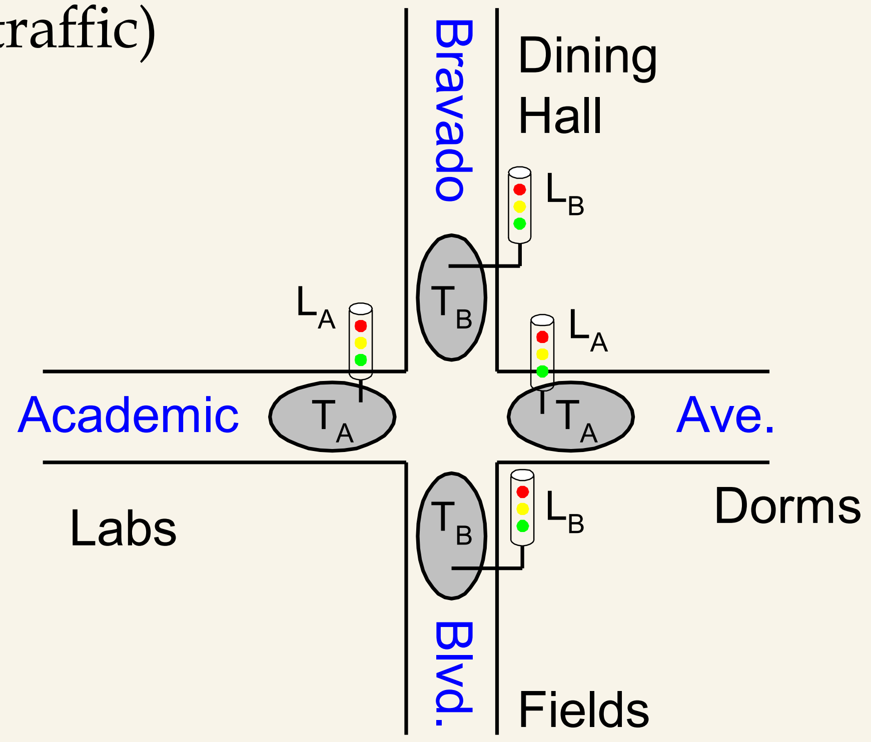


FSM Example

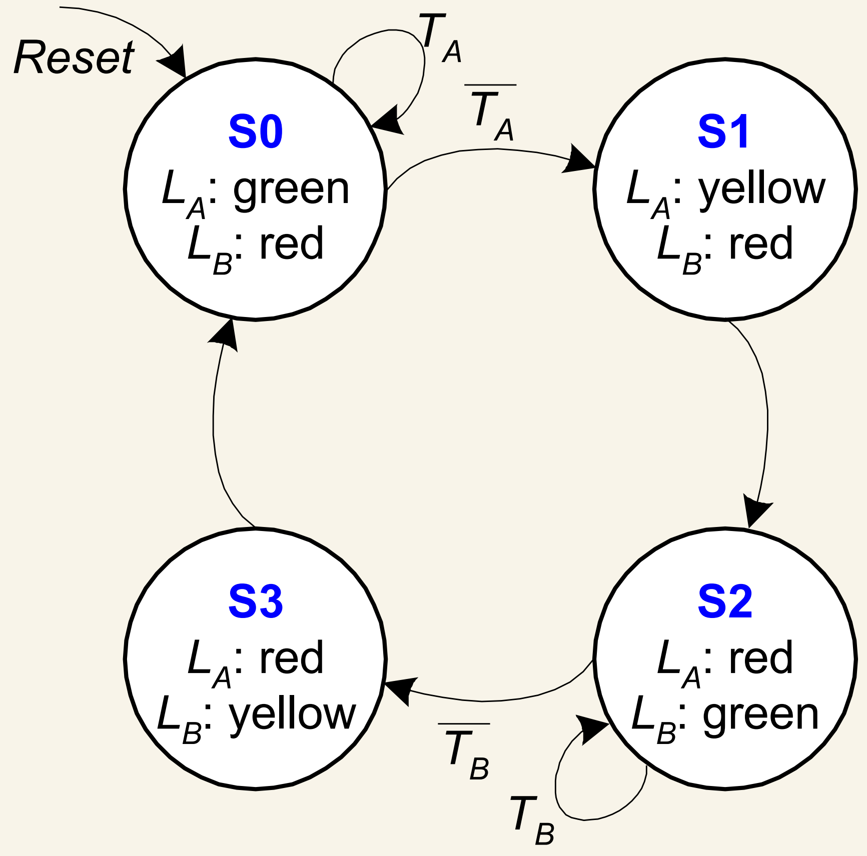
- Traffic light controller
  - » Traffic sensors: T_A, T_B (TRUE when there's traffic)
  - » Lights: L_A, L_B

- FSM Black Blox





- FSM State Transition Diagram
  - » Moore FSM: outputs labeled in each state
  - » States : Circles
  - » Transitions : Arcs



- FSM State Transition Diagram

Current State	Inputs		Next State
S	T_A	T_B	*S'*
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

- FSM Encoded State Transition Table

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

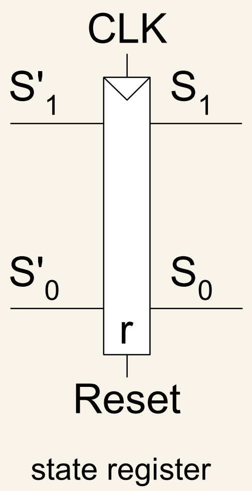
- FSM Output Table

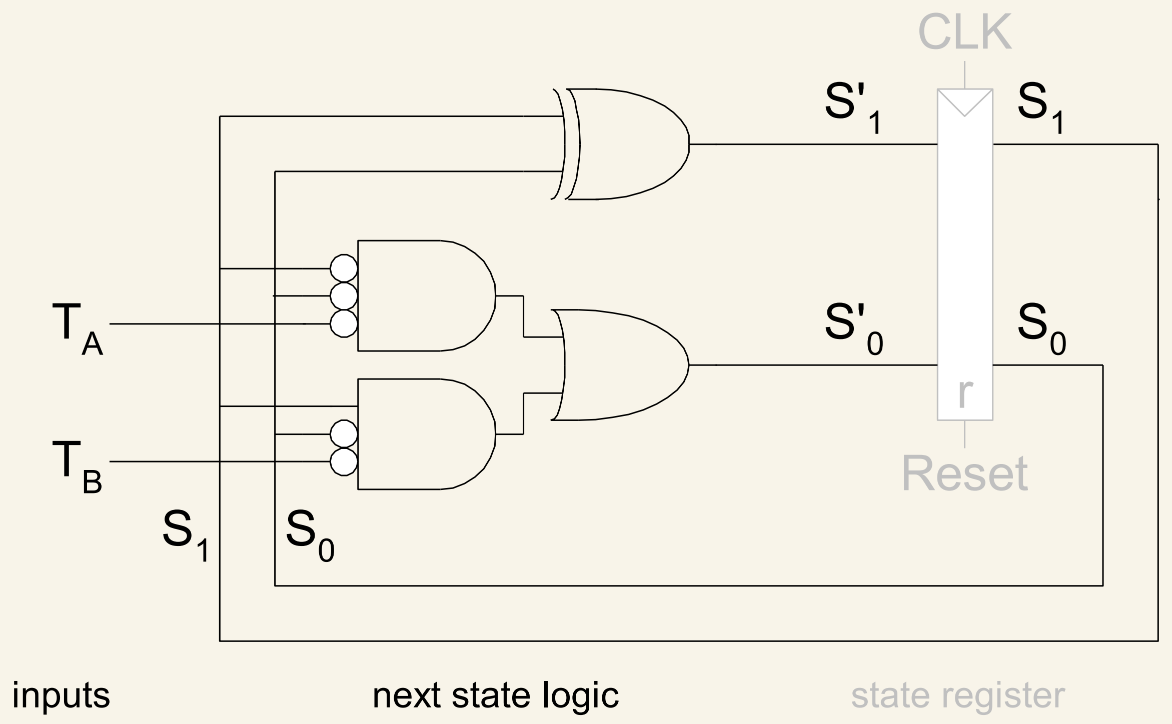
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

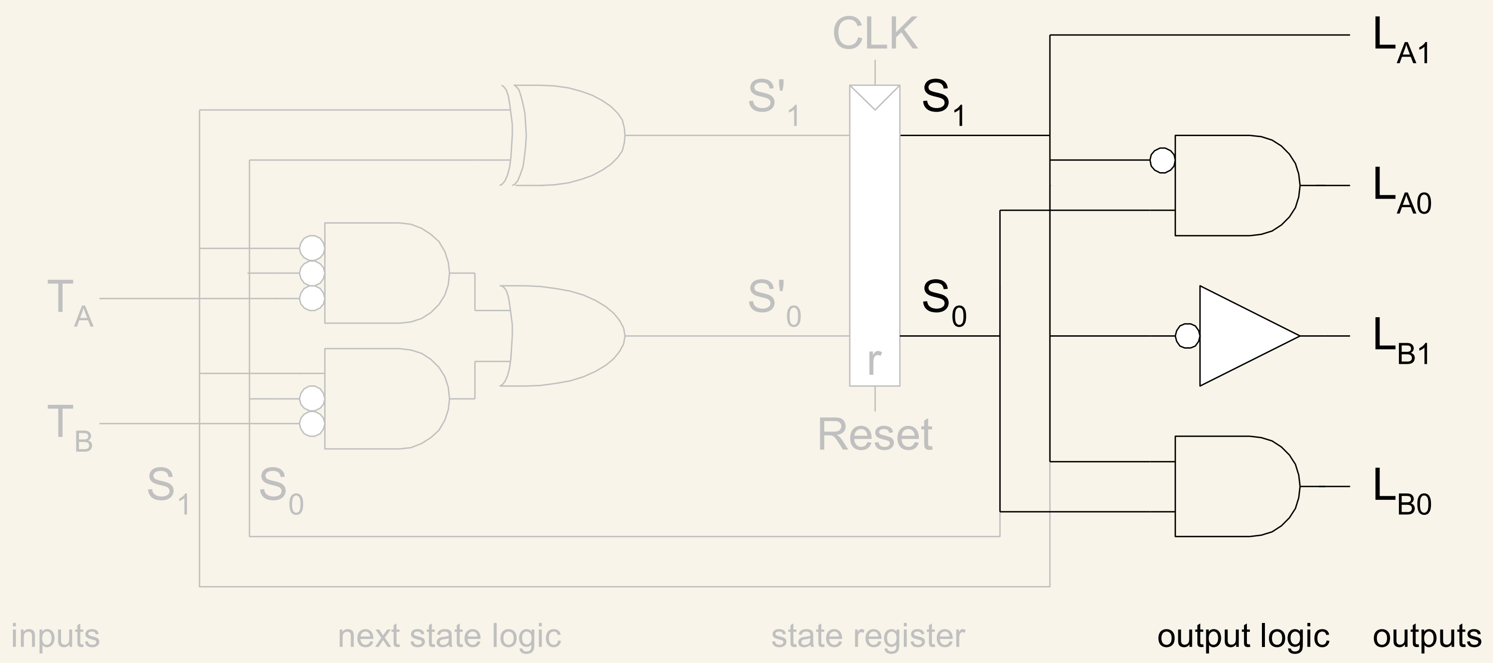
Output	Encoding
green	00
yellow	01
red	10

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1}S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1S_0$$

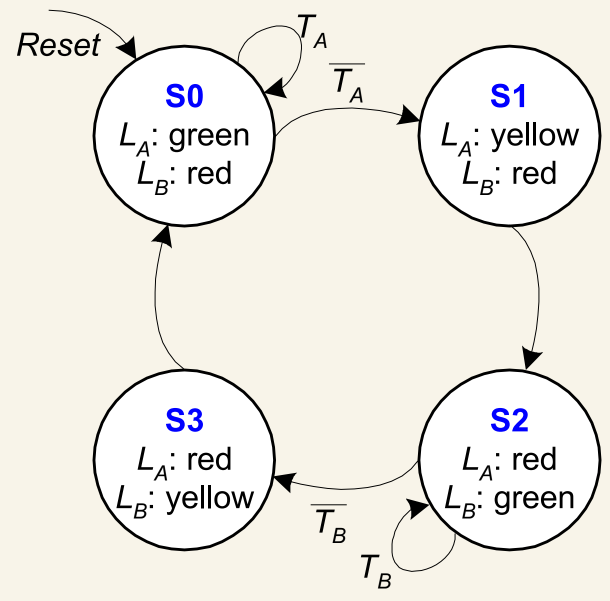
- FSM Schematic
  - State Register
  - Next State Logic
  - Output Logic

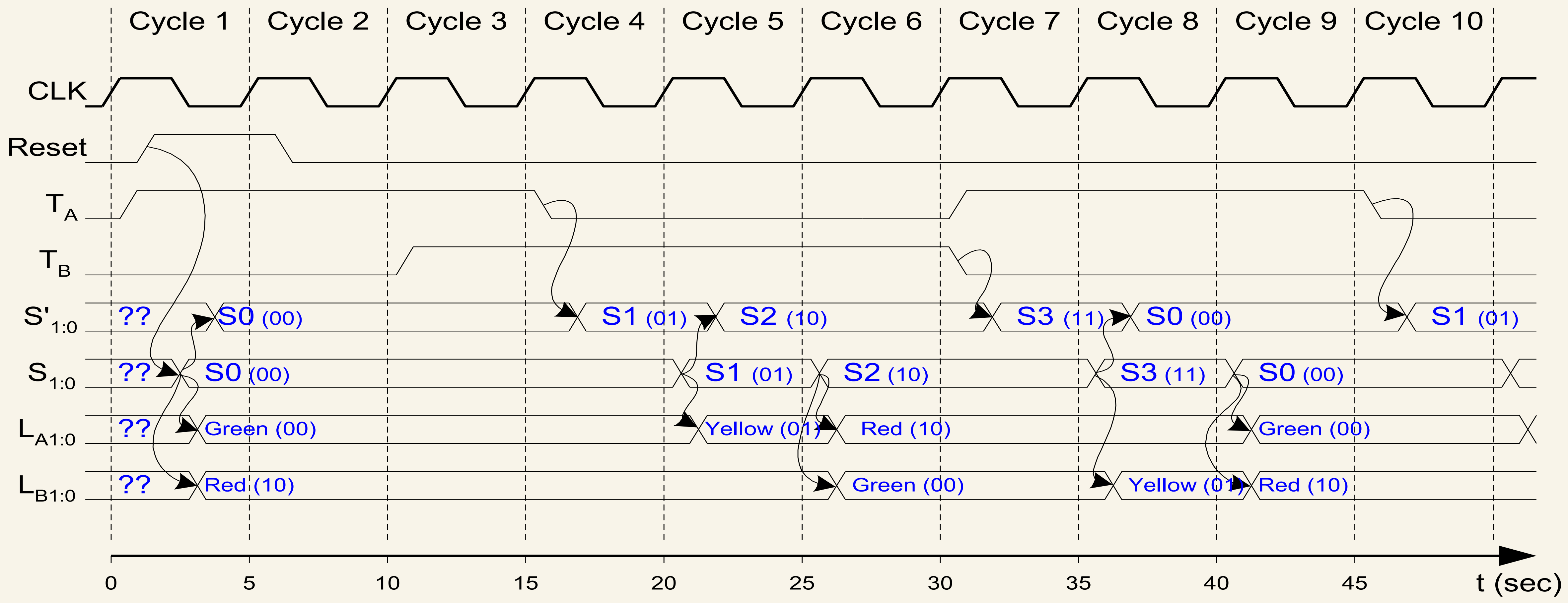






- FSM Timing Diagram



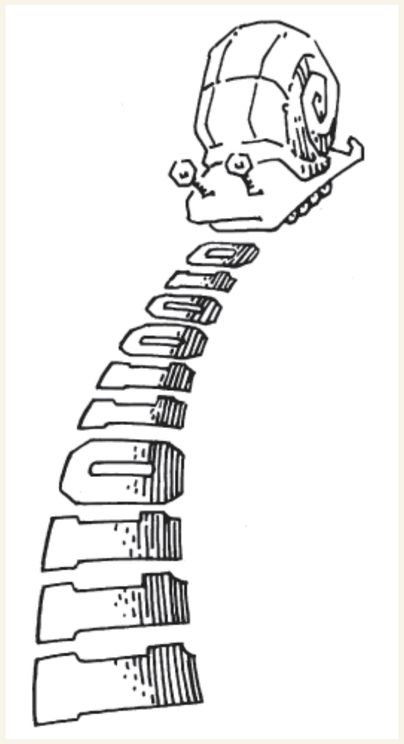


FSM State Encoding

- Binary encoding:
  - i.e., for four states, 00, 01, 10, 11

- One-hot encoding
  - One state bit per state
  - Only one state bit HIGH at once
    - » i.e., for 4 states, 0001, 0010, 0100, 1000
  - Requires **more flip-flops**
  - Often next state and output **logic is simpler**

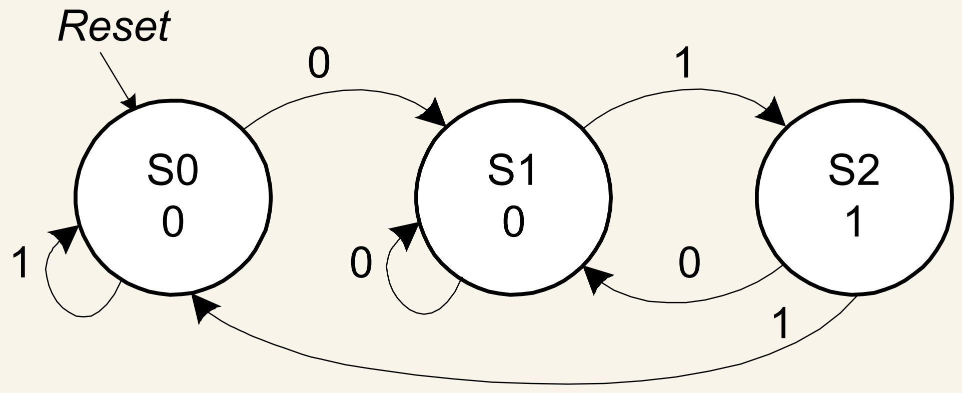
Moore vs. Mealy FSM



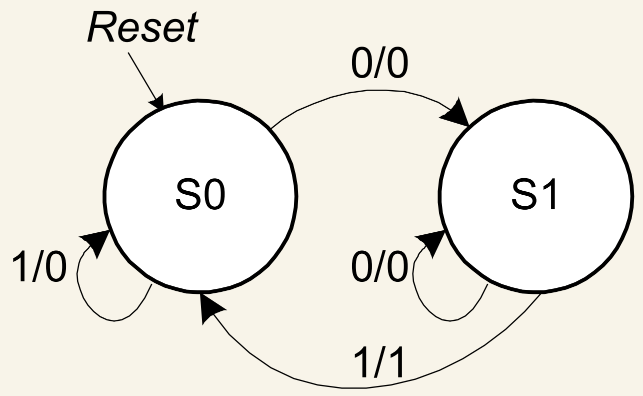
- Example
  - » You have a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.

- State Transition Diagrams

Moore FSM



Mealy FSM



- Moore Tables
  - » Transition Table

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

  - » Output Table

Current State		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

$$Y = S_1$$

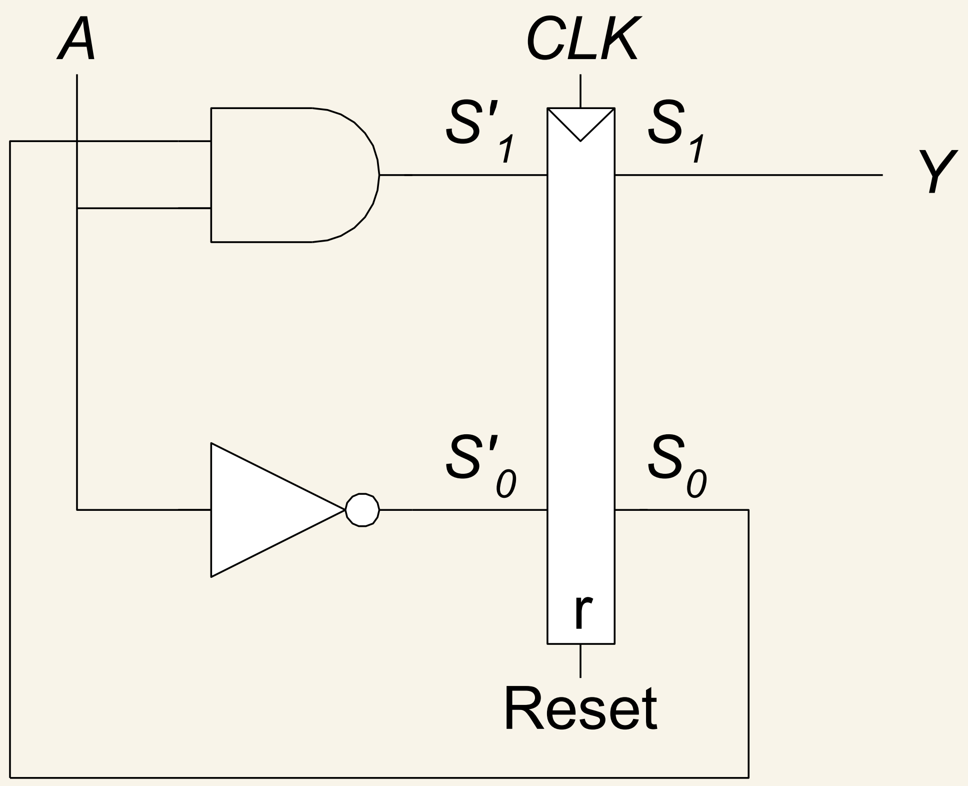
- Mealy Tables
  - » Transition Table
  - » Output Table

Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

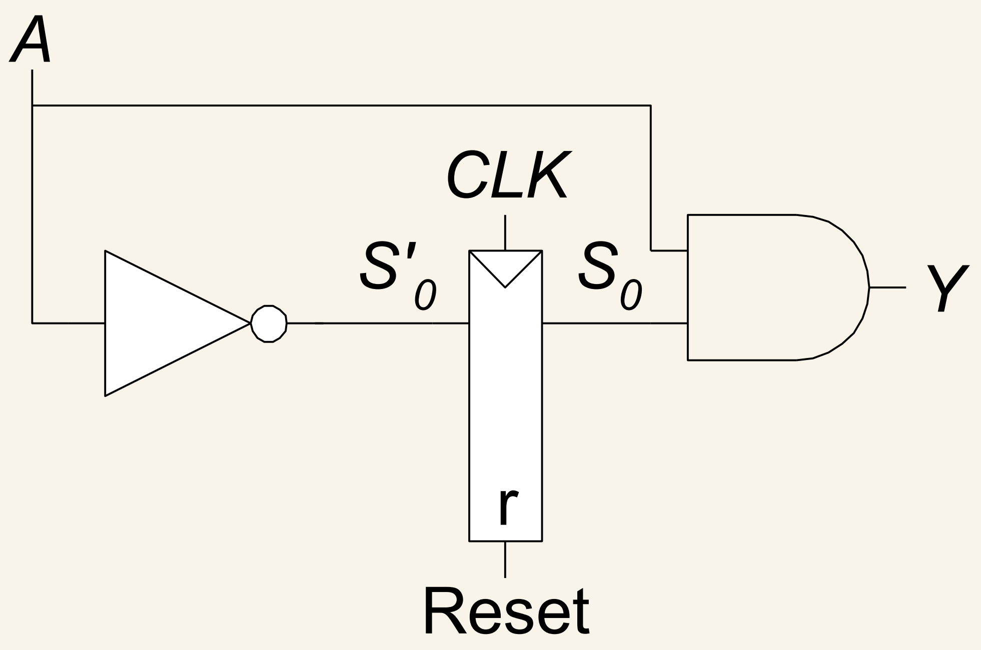
State	Encoding
S0	00
S1	01

$$S_0' = \overline{A}$$
$$Y = S_0 A$$

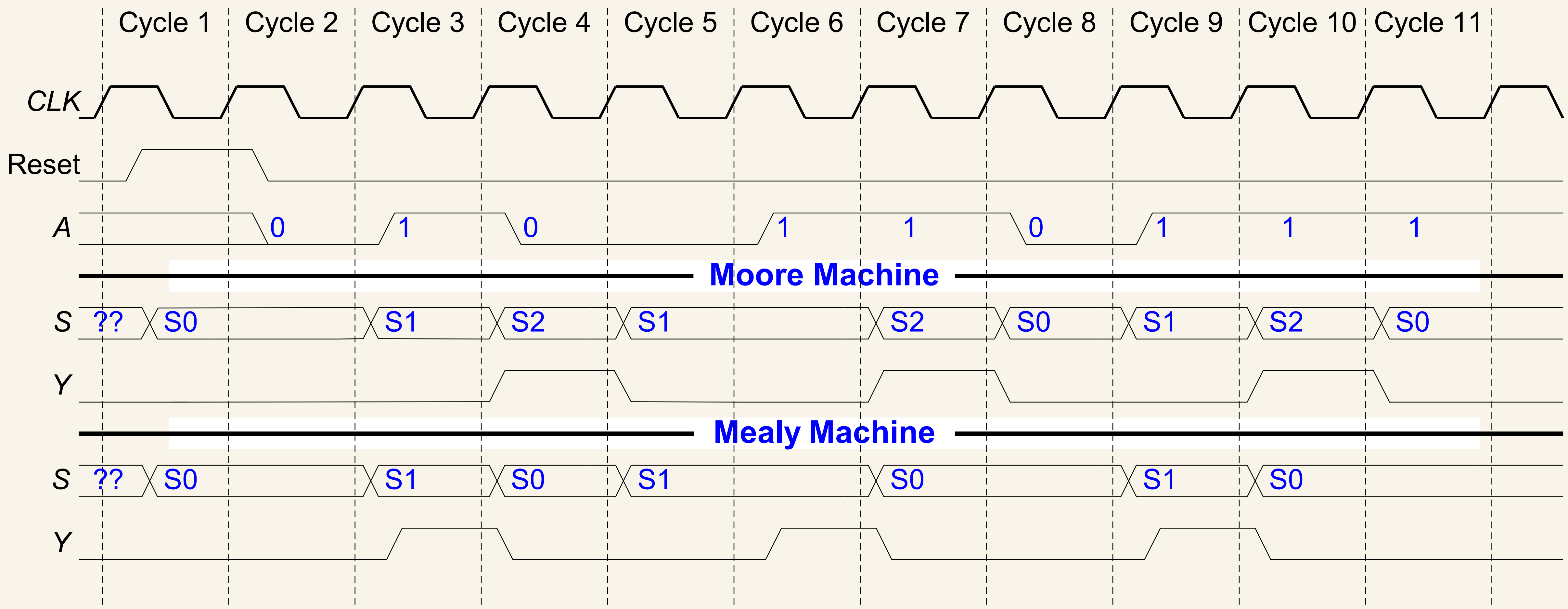
- Moore Schematic

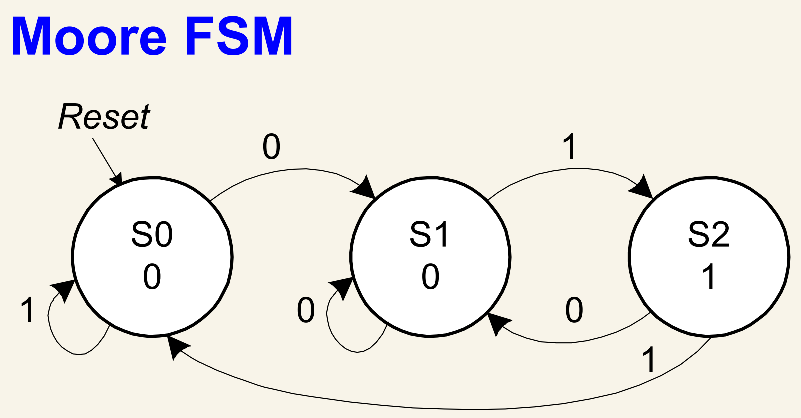


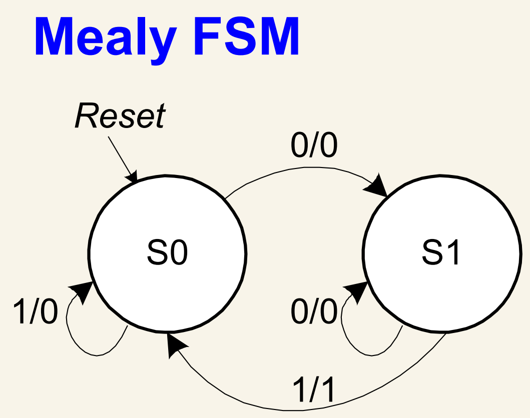
- Mealy Schematic



- Moore & Mealy Timing Diagram

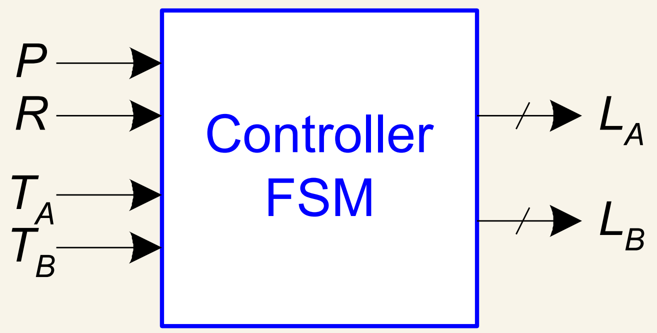




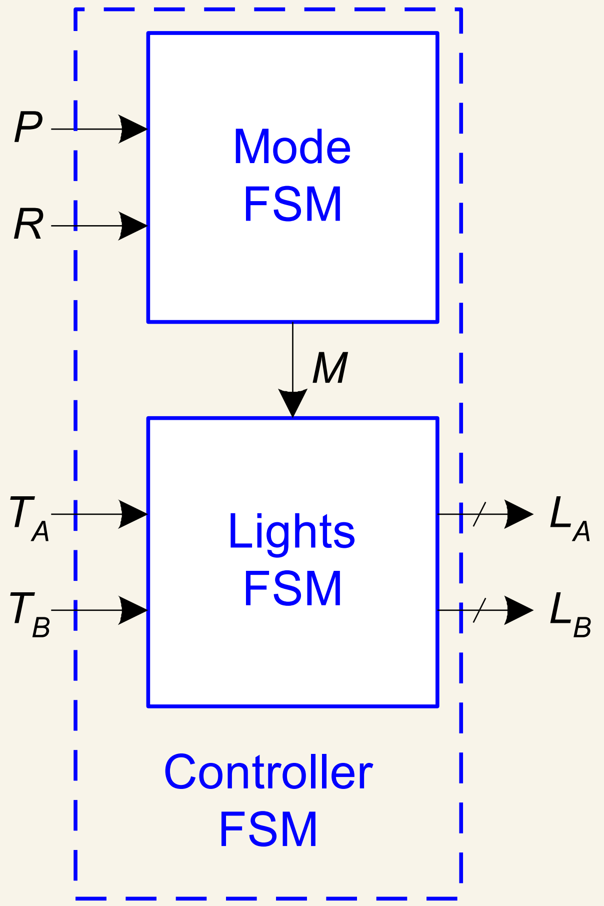


Factoring FSM

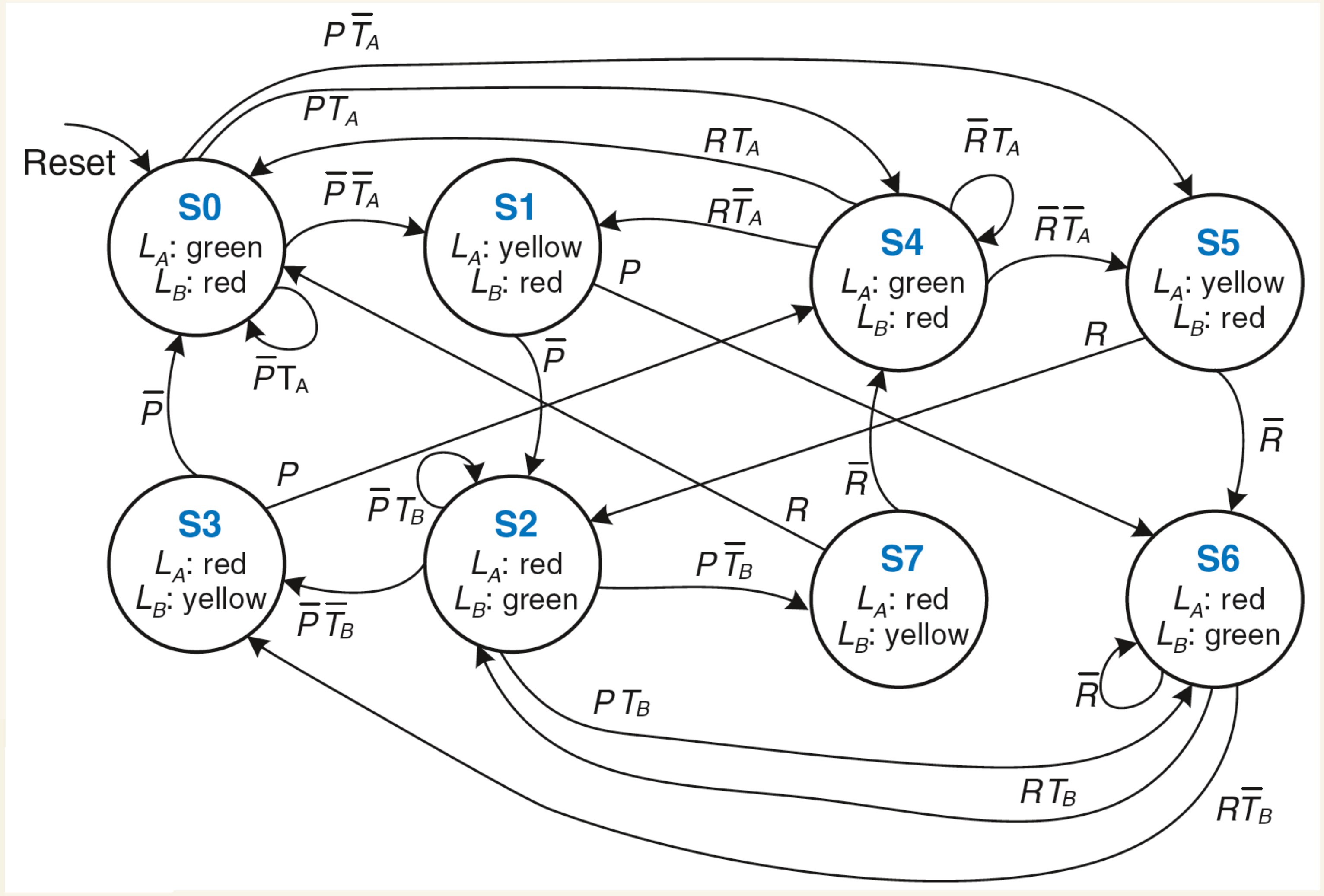
- Break complex FSMs into smaller interacting FSMs
- Example: Modify traffic light controller to have Parade Mode.
  - Two more inputs: P, R
  - When P = 1, enter Parade Mode & Bravado Blvd light stays green
  - When R = 1, leave Parade Mode



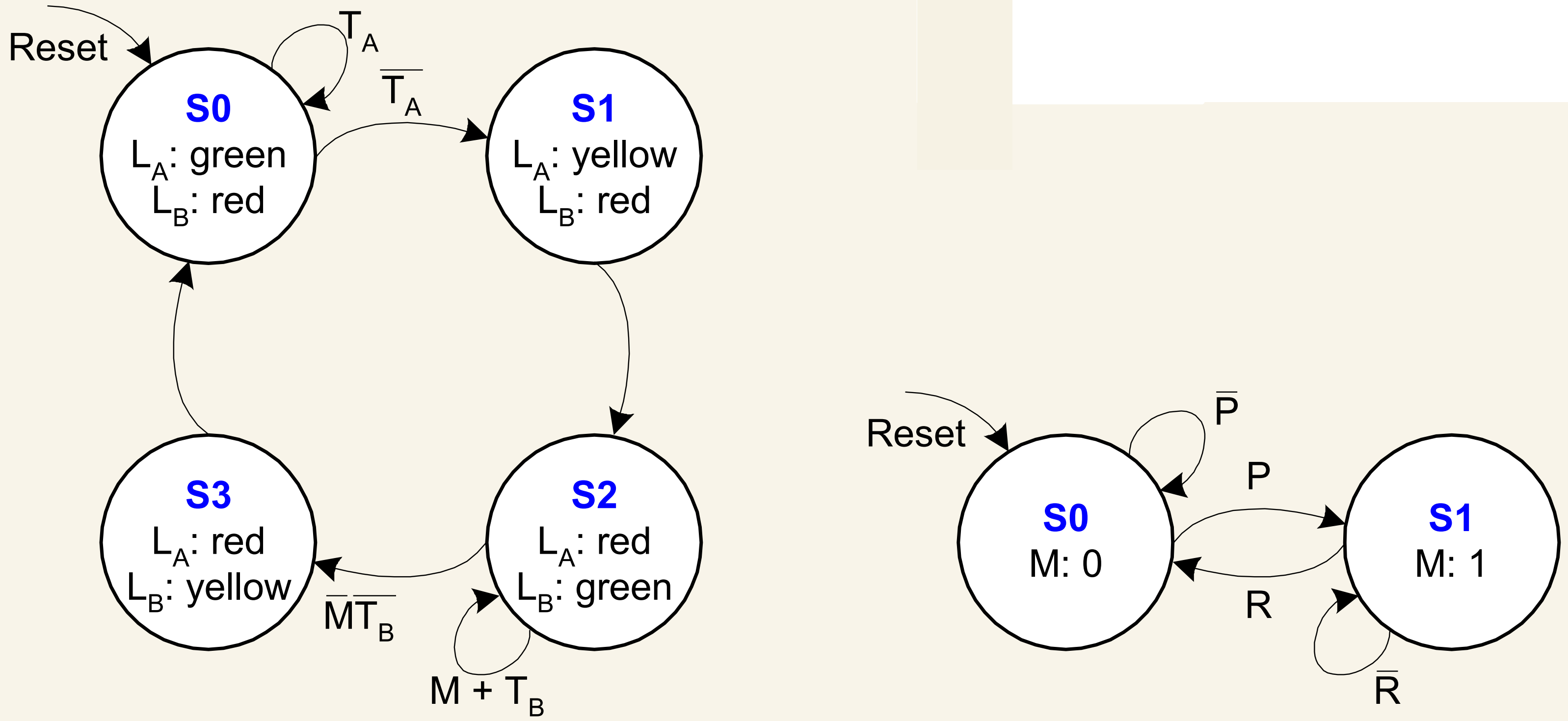
Unfactored FSM



Factored FSM



Unfactored FSM



Lights FSM

Factored FSM

Mode FSM

FSM Design Procedure

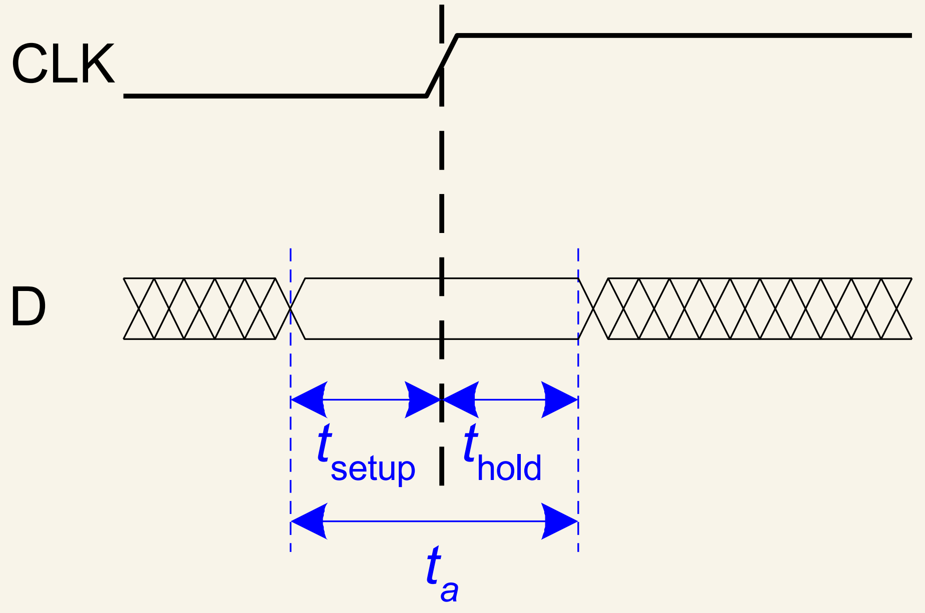
1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. For Moore machine:
   - Rewrite state transition table with state encodings
   - Write output table
6. For a Mealy machine:
   - Rewrite combined state transition and output table with state encodings
7. Write Boolean equations for next state and output logic
8. Simplify Boolean equations
9. Sketch the circuit schematic

Timing

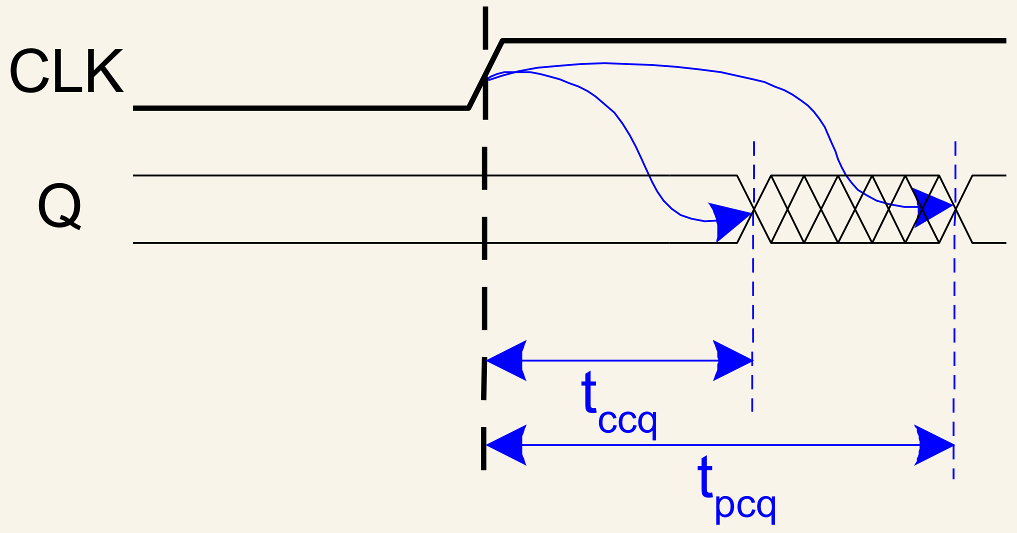
- Flip-flop samples D at clock edge
- D must be stable when sampled
- Similar to a photograph, D must be stable around clock edge
- If not, metastability can occur

Timing Constraints

- Input Constraints
  - » Setup time: t_{setup} = time before clock edge data must be stable (i.e. not changing)
  - » Hold time: t_{hold} = time after clock edge data must be stable
  - » Aperture time: t_a = time around clock edge data must be stable ($t_a = t_{setup} + t_{hold}$)

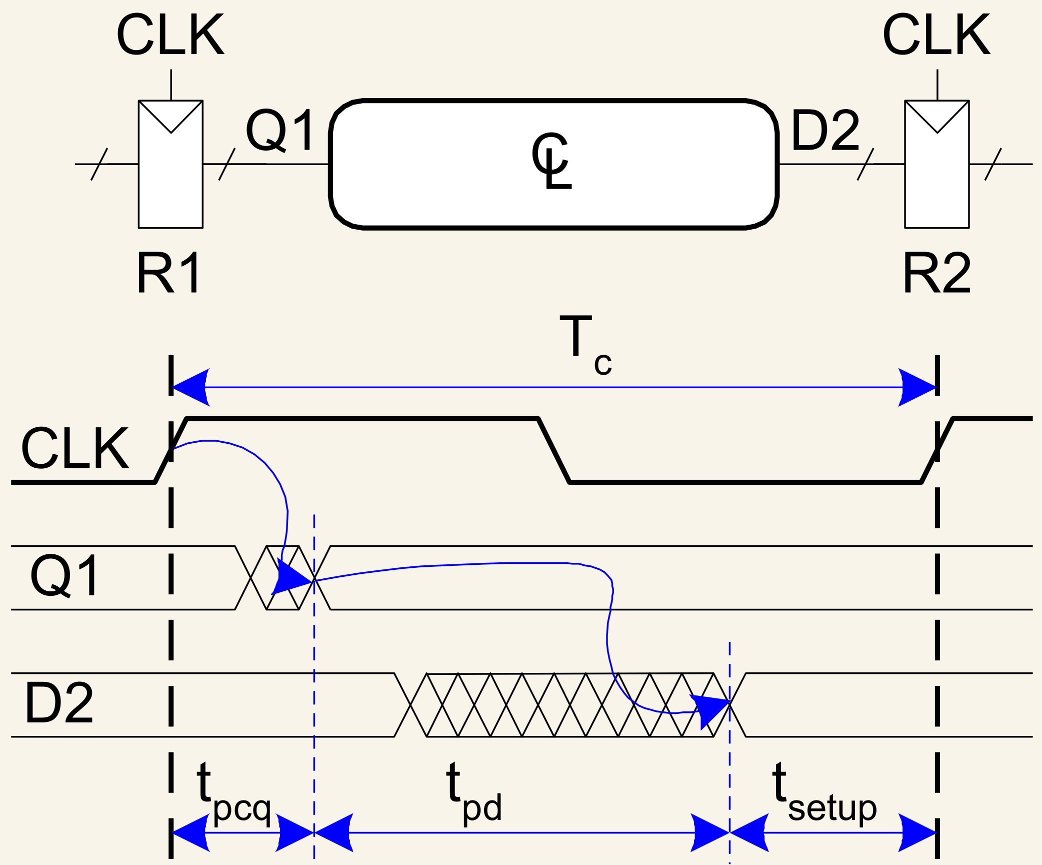


- Output Constraints
  - » Propagation delay: t_{pcq} = time after clock edge that the output Q is guaranteed to be stable (i.e., to stop changing)
  - » Contamination delay: t_{ccq} = time after clock edge that Q might be unstable (i.e., start changing)



Setup Time Constraint

- Depends on the maximum delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge

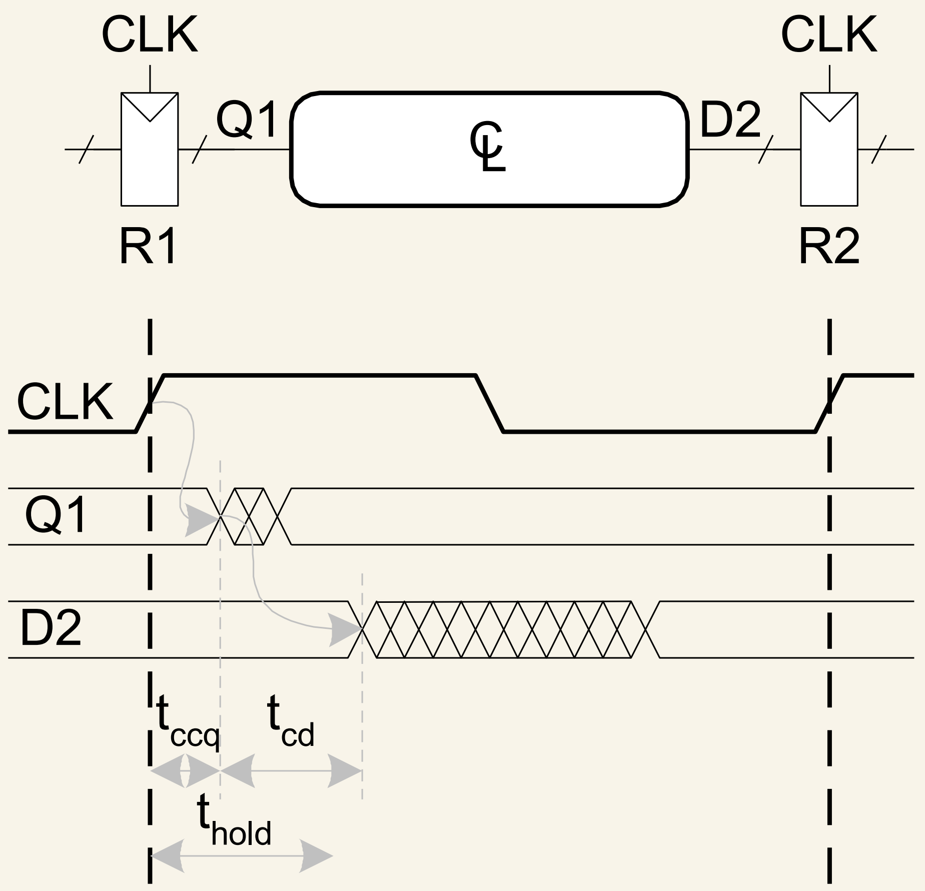


$$\boxed{\begin{aligned} T_c &\geq t_{pcq} + t_{pd} + t_{setup} \\ t_{pd} &\leq T_c - (t_{pcq} + t_{setup}) \end{aligned}}$$

$(t_{pcq} + t_{setup})$: *sequencing overhead*

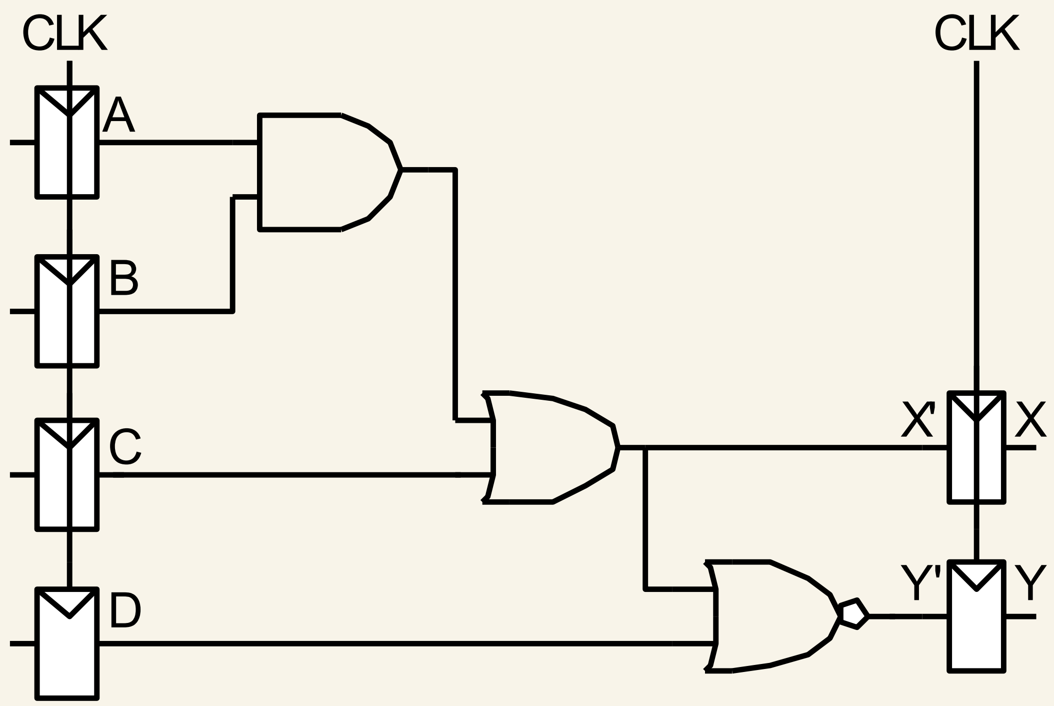
Hold Time Constraint

- Depends on the minimum delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



$$t_{hold} < t_{ccq} + t_{cd}$$
$$t_{cd} > t_{hold} - t_{ccq}$$

Timing Analysis



Timing Characteristics

t_{ccq}	= 30 ps
t_{pcq}	= 50 ps
t_{setup}	= 60 ps
t_{hold}	= 70 ps

per gate

t_{pd}	= 35 ps
t_{cd}	= 25 ps

t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 25 ps

Setup time constraint:

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

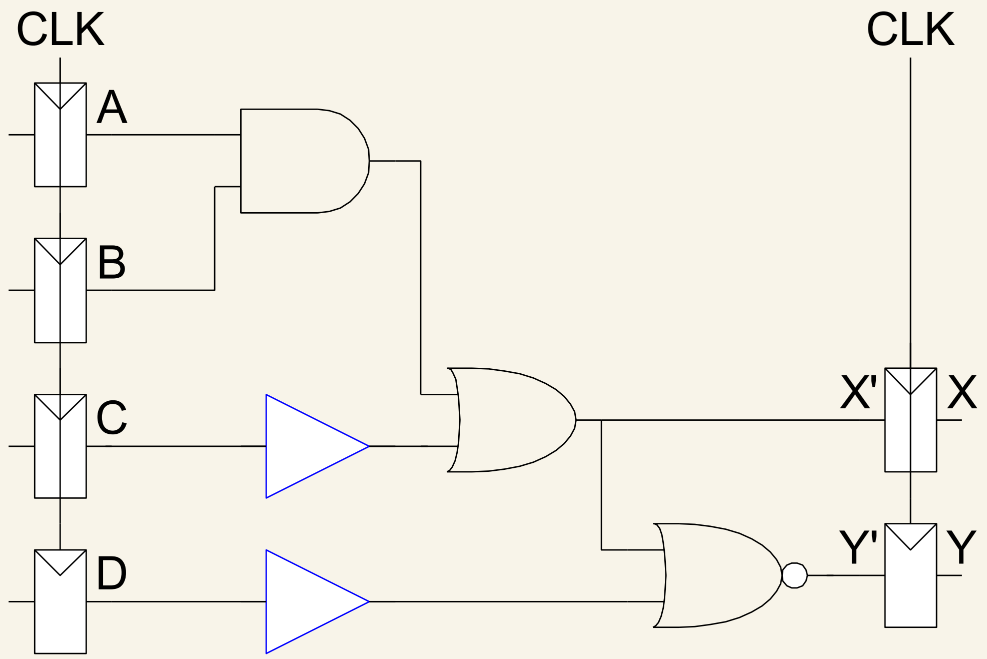
$f_c = 1/T_c$ = 4.65 GHz

Hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?

(30 + 25) ps > 70 ps ? **No!**

Add buffers to the short paths:



Timing Characteristics

t_{ccq}	= 30 ps
t_{pcq}	= 50 ps
t_{setup}	= 60 ps
t_{hold}	= 70 ps

per gate

t_{pd}	= 35 ps
t_{cd}	= 25 ps

t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 2 x 25 ps = 50 ps

Setup time constraint:

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

$f_c = 1/T_c = 4.65$ GHz

Hold time constraint:

$t_{ccq} + t_{cd} > t_{\text{hold}}$?

(30 + 50) ps > 70 ps ? **Yes!**

Parallelism

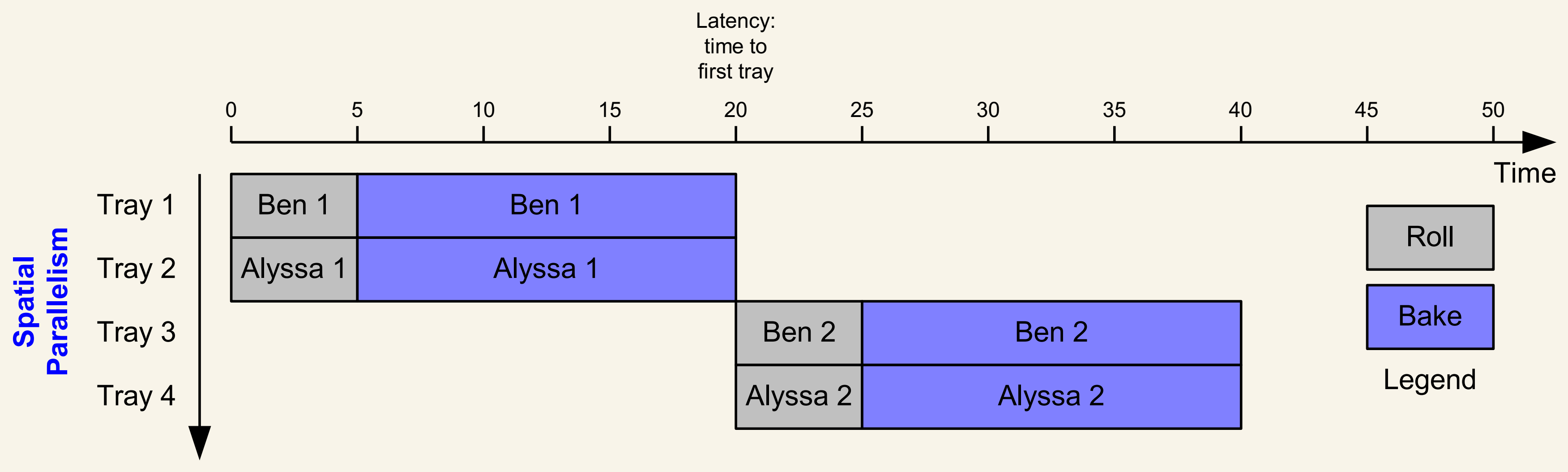
- Two types of parallelism:
  - Spatial parallelism
    - » duplicate hardware performs multiple tasks at once
  - Temporal parallelism
    - » task is broken into multiple stages
    - » also called pipelining
    - » for example, an assembly line

- Some definitions
  - Token : Group of inputs processed to produce group of outputs
  - Latency : Time for one token to pass from start to end
  - Throughput : Number of tokens produced per unit time

- Parallelism increases throughput

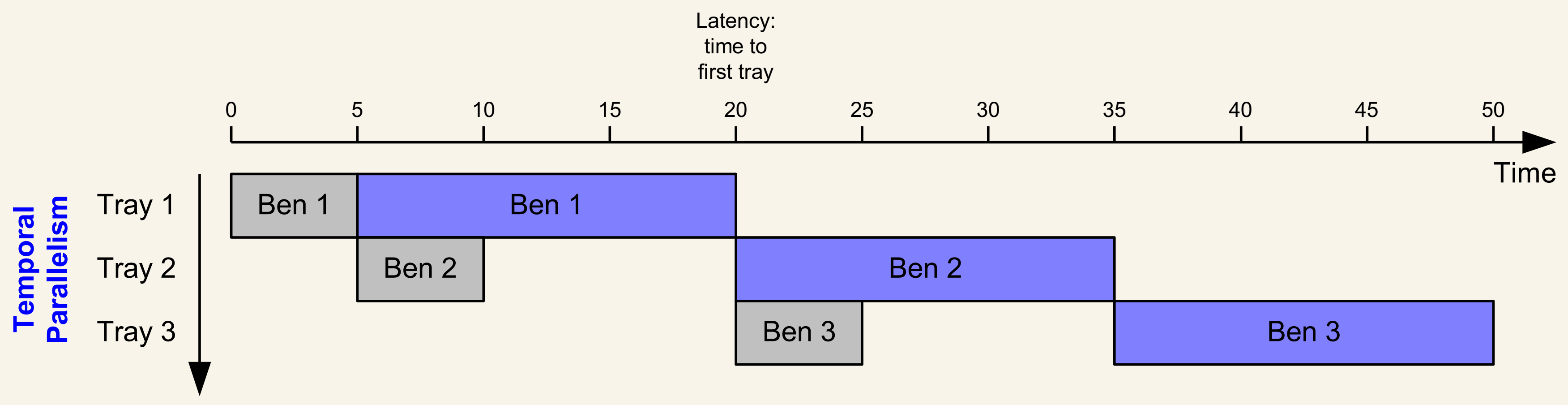
- Example
  - Ben Bitdiddle bakes cookies to celebrate traffic light controller installation
  - 5 minutes to roll cookies
  - 15 minutes to bake
  - What is the latency and throughput without parallelism?
    - » Latency = 5 + 15 = 20 minutes = 1/3 hour
    - » Throughput = 1 tray/ 1/3 hour = 3 trays/hour
  - What is the latency and throughput if Ben uses parallelism?
    - » Spatial parallelism
      - Ben asks Allysa P. Hacker to help, using her own oven
    - » Temporal parallelism:
      - two stages: rolling and baking
      - He uses two trays
      - While first batch is baking, he rolls the second batch, etc.

– Spatial Parallelism



Latency = 5 + 15 = 20 minutes = 1/3 hour
Throughput = 2 trays/ 1/3 hour = 6 trays/hour

– Temporal Parallelism



Latency = 5 + 15 = 20 minutes = 1/3 hour
Throughput = 1 trays/ 1/4 hour = 4 trays/hour