Review on Digital Design (Sequential Circuits)

University of South Carolina

Introduction to Computer Architecture
Fall, 2024
Mehdi Yaghouti

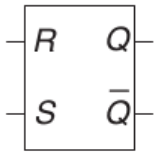
Molinaroli College of Engineering and Computing
UNIVERSITY OF SOUTH CAROLINA

Storage Elements

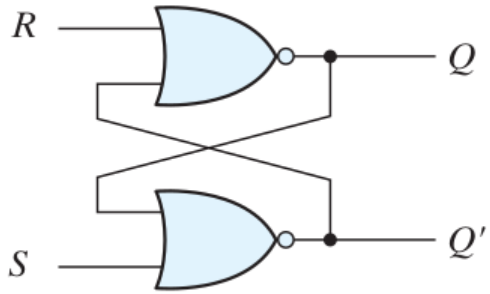
- Maintains a binary state as long as power is on
- Provides control signals to switch the state
- Level sensitive storage elements: Latches
- Transition sensitive storage elements: Flip-Flops
- Latches are the building blocks of Flip-Flops
- Latches are useful for asynchronous designs
- Flip-Flops are the choice for synchronous circuits

Latches

- Latches are level sensitive
- S and R stand for Set and Reset
- $S = 1, R = 0$ set the output
- $S = 0, R = 1$ reset the output
- $S = 0, R = 0$ retains the old value
- $S = 1, R = 1$ causes race condition (forbidden)

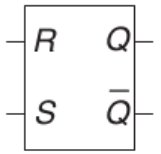


S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)

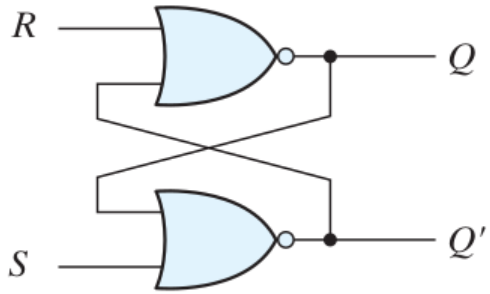


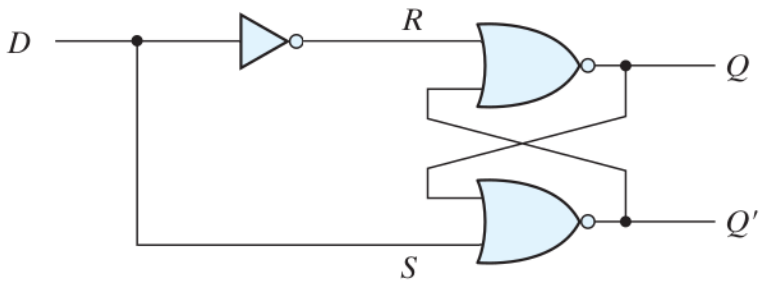
D-Latches

- Latches are level sensitive
- S and R stand for Set and Reset
- $S = 1, R = 0$ set the output
- $S = 0, R = 1$ reset the output
- $S = 0, R = 0$ retains the old value
- $S = 1, R = 1$ causes race condition (forbidden)
- D-Latch eliminates the forbidden status



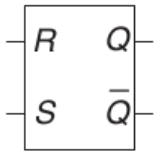
S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)



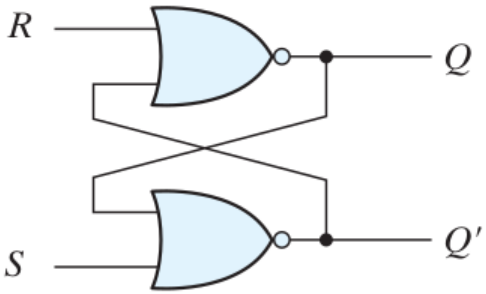


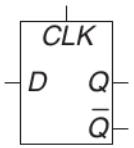
D-Latches

- Latches are level sensitive
- S and R stand for Set and Reset
- $S = 1, R = 0$ set the output
- $S = 0, R = 1$ reset the output
- $S = 0, R = 0$ retains the old value
- $S = 1, R = 1$ causes race condition (forbidden)
- D-Latch eliminates the forbidden status
- Clk determines when change is allowed

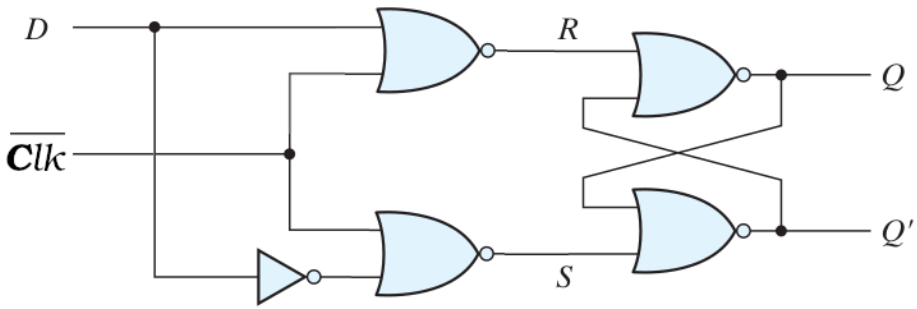


S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)



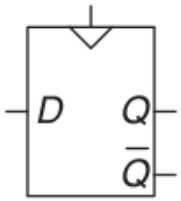


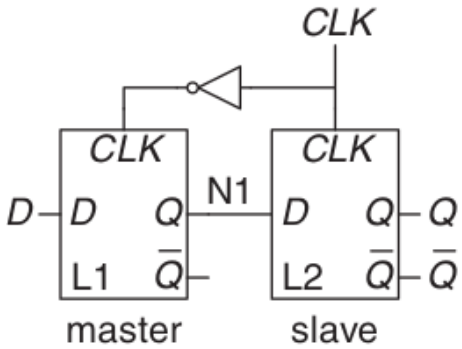
Clk	D	Next state of Q
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

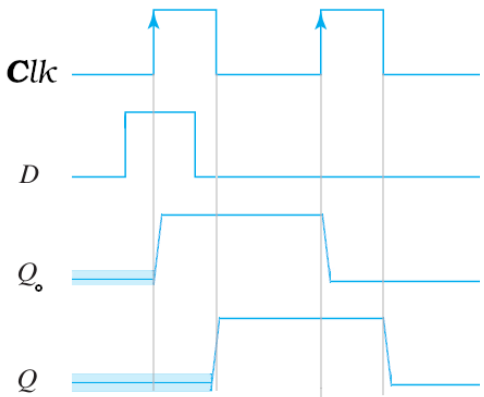


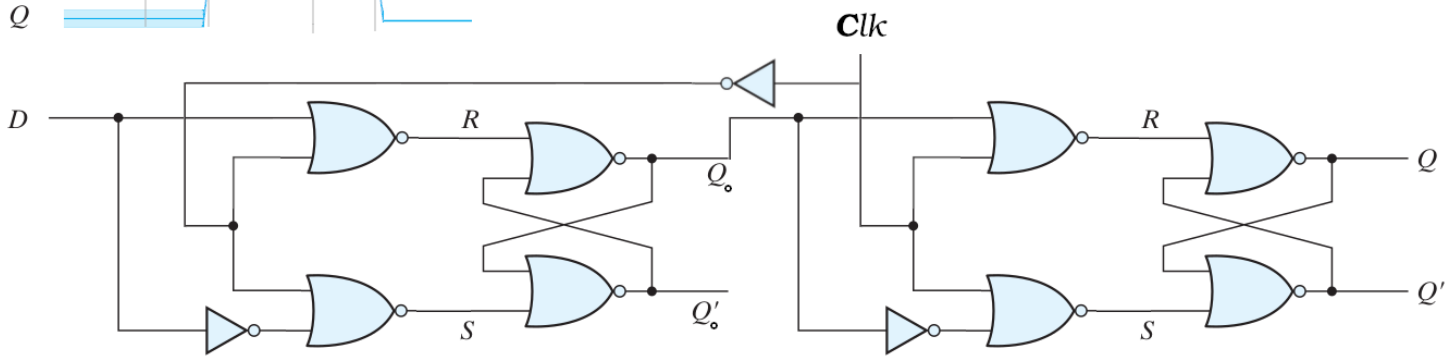
D Flip-Flop

- Master and Slave D-Latches connected back to back
- Master and Slave receives complementary clocks
- At $Clk = 1$, Master Latch is transparent
- At $Clk = 0$, Slave Latch is transparent



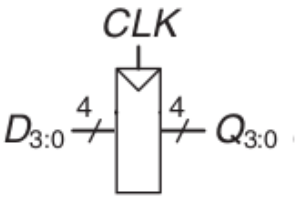


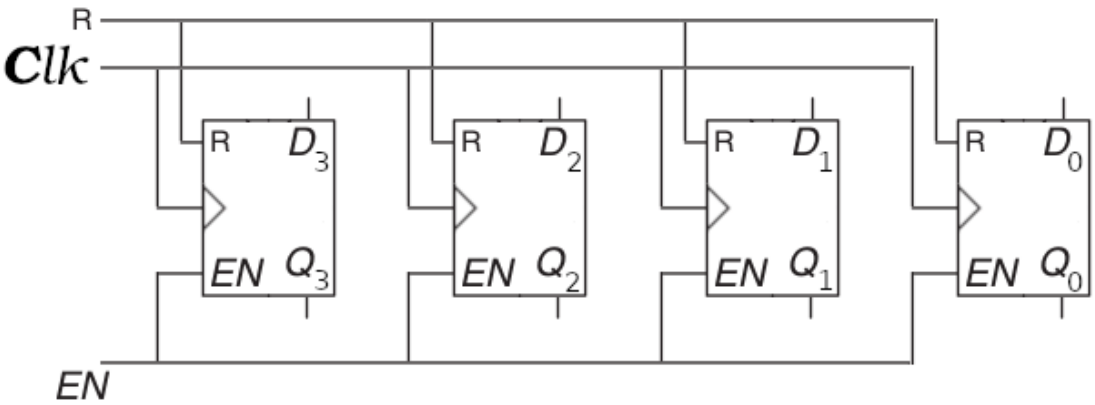


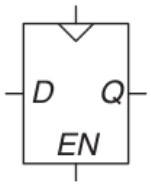


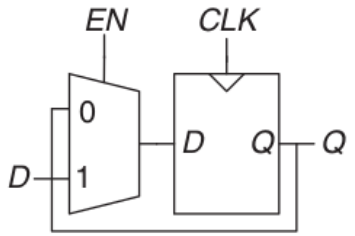
Register

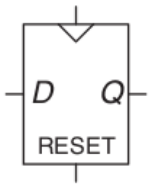
- Enable determines whether data is loaded
- Reset clear the output to 0
- N-bit register is a bank of N Flip-Flops

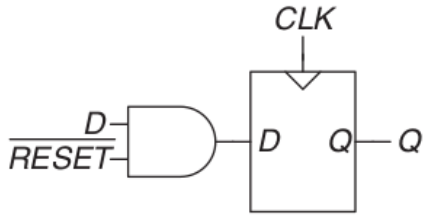


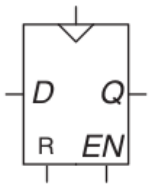


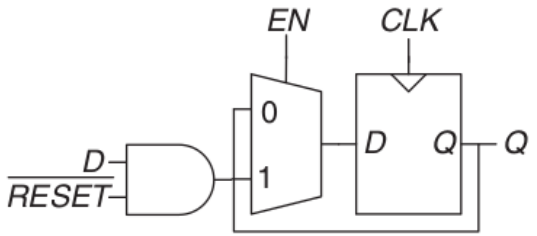






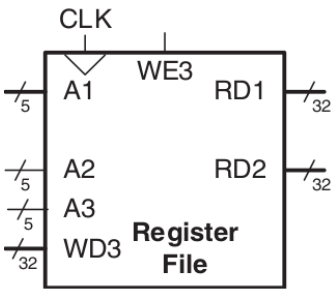


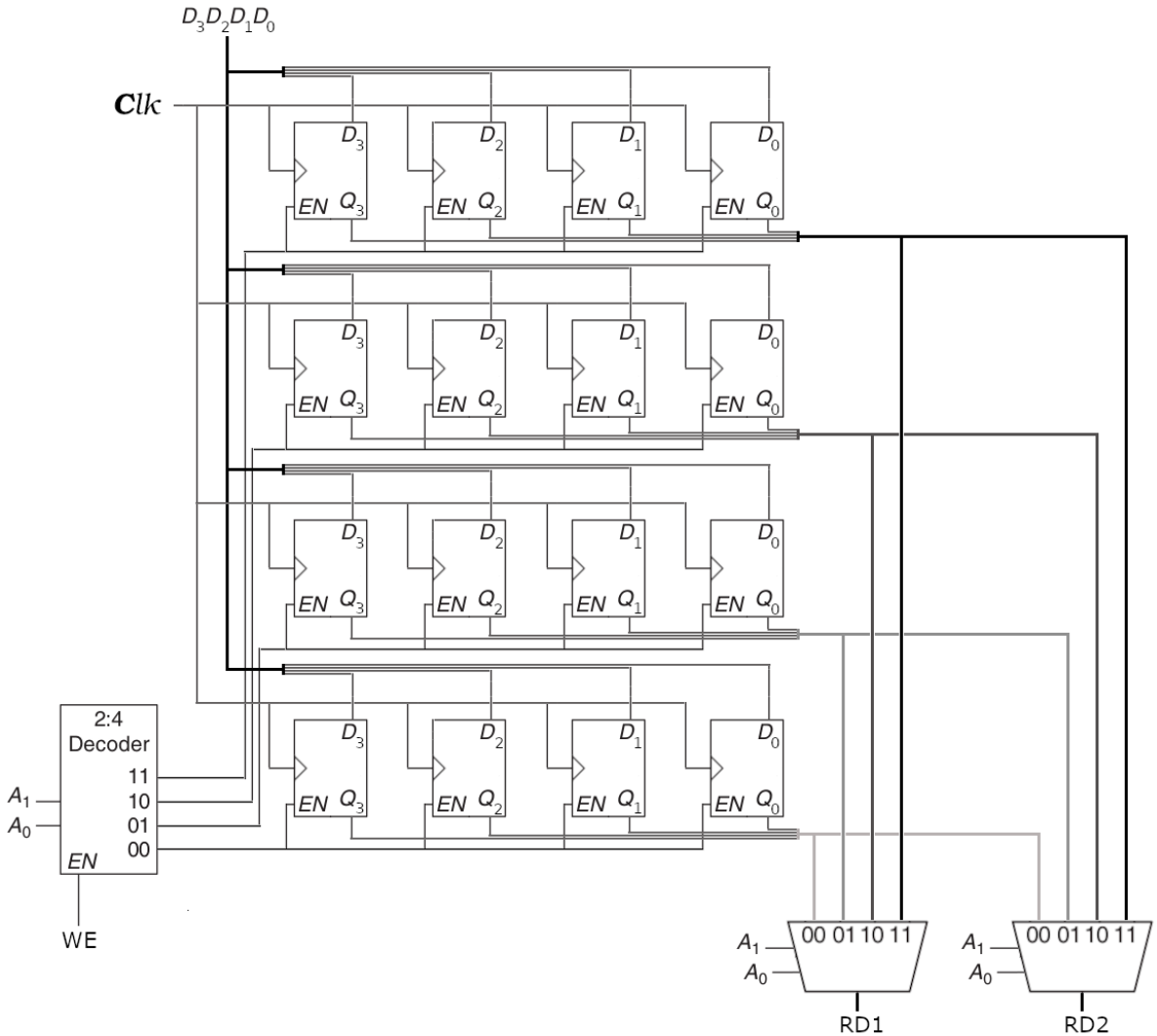




Register File

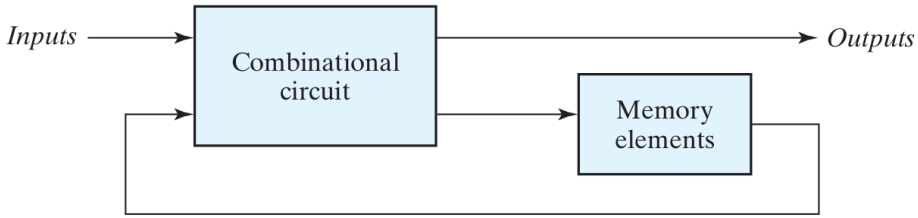
- 4×4-bits register file
- 2 Read ports
- 1 Write port
- $log_2 4$ address bus
- 32×32-bits register file





Synchronous Sequential Circuits

- The outputs and the next state are both a function of the inputs
- The state of the system is hold in flip flops

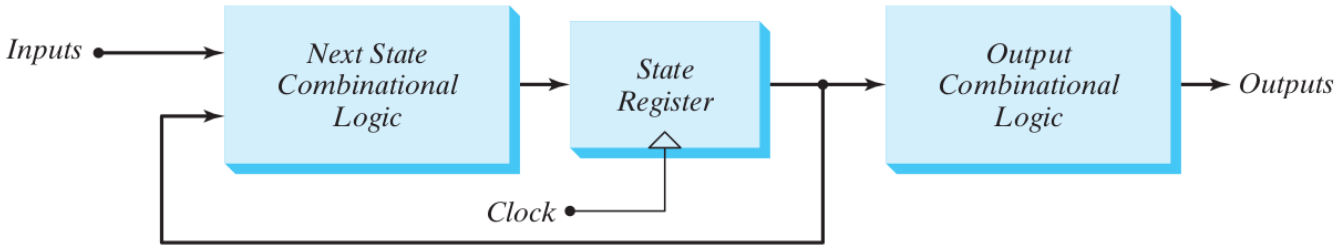


Finite State Machines

- Synchronous sequential circuits can be designed as Finite State Machines
- At each clock both the state and the output of the circuit will be updated
- The next state is computed based on both input and the current state

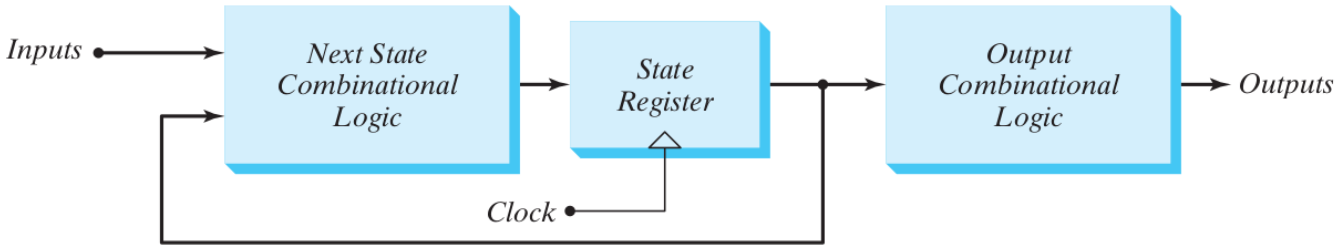
Finite State Machines

- Synchronous sequential circuits can be designed as Finite State Machines
- At each clock both the state and the output of the circuit will be updated
- The next state is computed based on both input and the current state
- There are two type of FSM
  - **Moore machines:** The output depends directly only on the current state

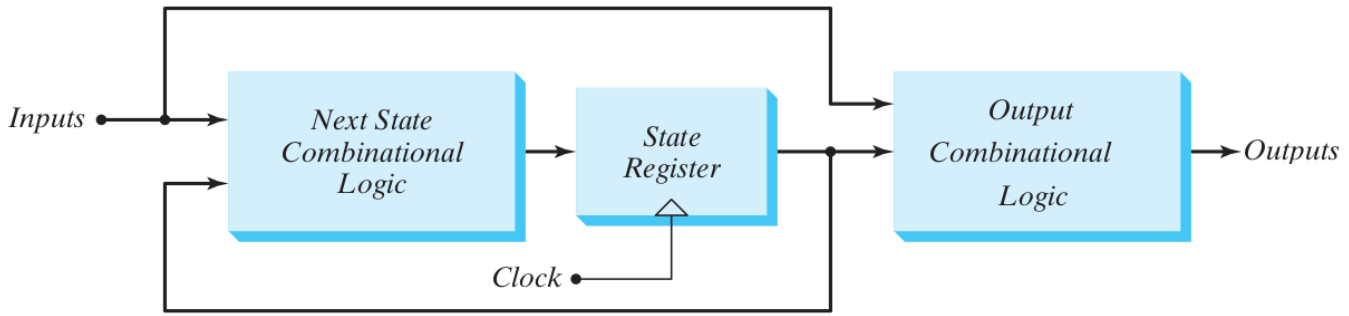


Finite State Machines

- Synchronous sequential circuits can be designed as Finite State Machines
- At each clock both the state and the output of the circuit will be updated
- The next state is computed based on both input and the current state
- There are two type of FSM
  - **Moore machines:** The output depends directly only on the current state



  - **Mealy machines:** The output depends on both current state and the input



State Diagram

- **State Diagram:**

A visual representation that depicts state transitions and outcomes in response to inputs

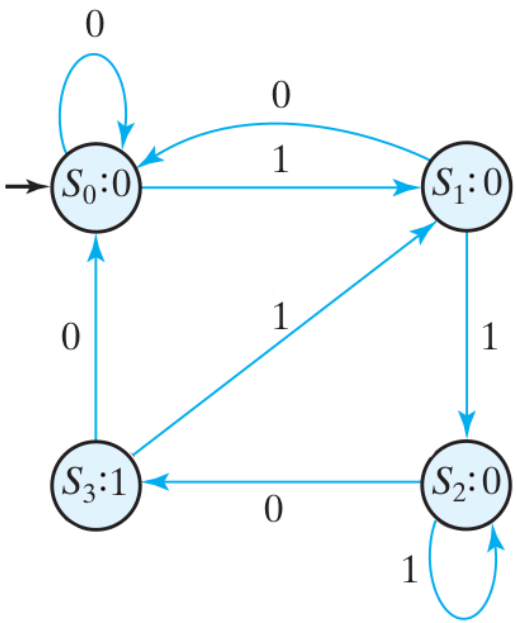
- **States:** Different conditions a system can be in
- **Transitions:** Arrows showing movement between states
- **Inputs:** Triggers for state changes
- **Reset state:** Start and end points of the process
- **State Actions:** Actions within states or during transitions

State Diagram

- **State Diagram:**

A visual representation that depicts state transitions and outcomes in response to inputs

- **States:** Different conditions a system can be in
- **Transitions:** Arrows showing movement between states
- **Inputs:** Triggers for state changes
- **Reset state:** Start and end points of the process
- **State Actions:** Actions within states or during transitions

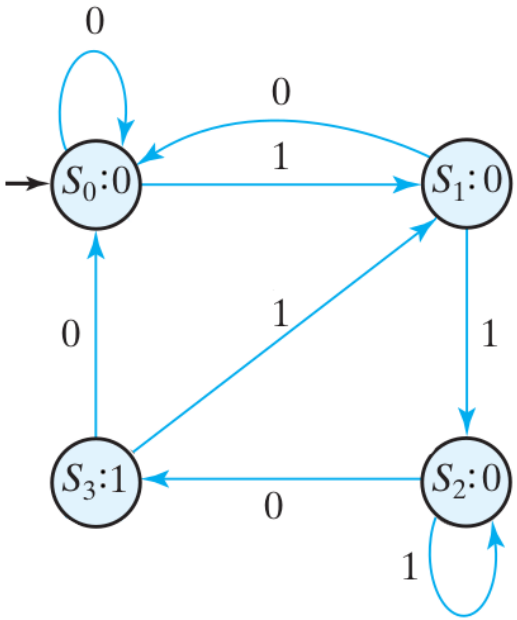


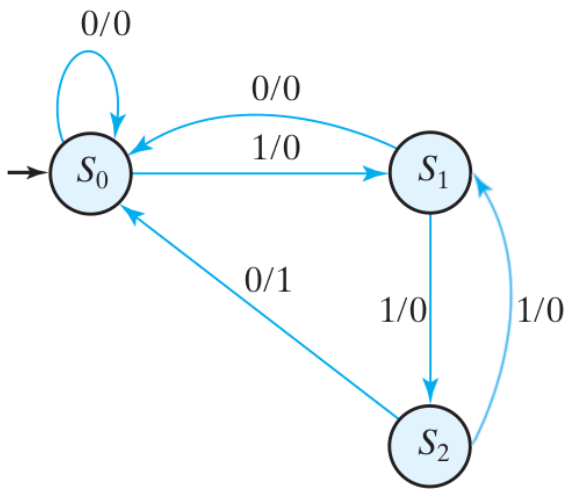
State Diagram

- **State Diagram:**

A visual representation that depicts state transitions and outcomes in response to inputs

- **States:** Different conditions a system can be in
- **Transitions:** Arrows showing movement between states
- **Inputs:** Triggers for state changes
- **Reset state:** Start and end points of the process
- **State Actions:** Actions within states or during transitions

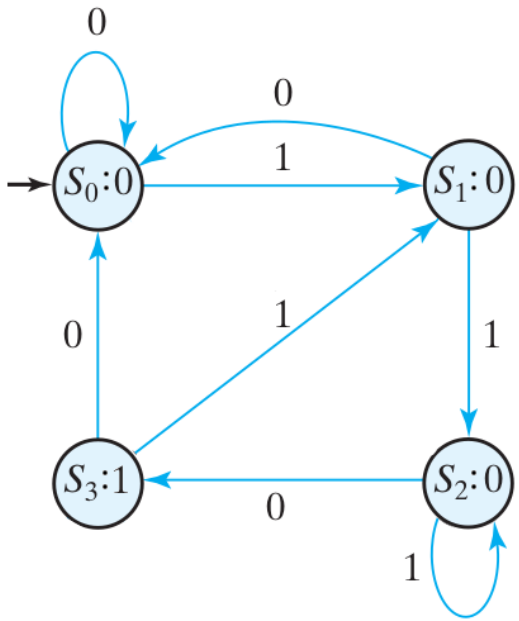




Moore Machine Design

1. Assigning binary codes to sates

$$S_0 \rightarrow 00$$
$$S_1 \rightarrow 01$$
$$S_2 \rightarrow 10$$
$$S_3 \rightarrow 11$$

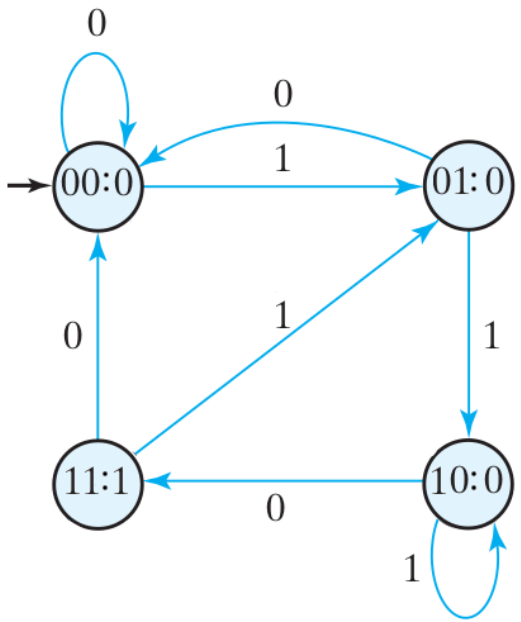


- State coding is not unique and will affect the design

Moore Machine Design

1. Assigning binary codes to sates
2. Obtain the binary-coded state table

Present State		Input	Next State	
S_1	S_0	x	S_1'	S_0'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1



Moore Machine Design

1. Assigning binary codes to sates
2. Obtain the binary-coded state table
3. Obtain the equations for flip flops inputs

Present State		Input	Next State	
S_1	S_0	x	S_1'	S_0'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

$$S_0' = S_1 S_0 x + S_1 \overline{S_0}\,\overline{x} + \overline{S_1}\,\overline{S_0} x$$

$$S_1' = S_1 \overline{S_0} + \overline{S_1} S_0 x$$

Moore Machine Design

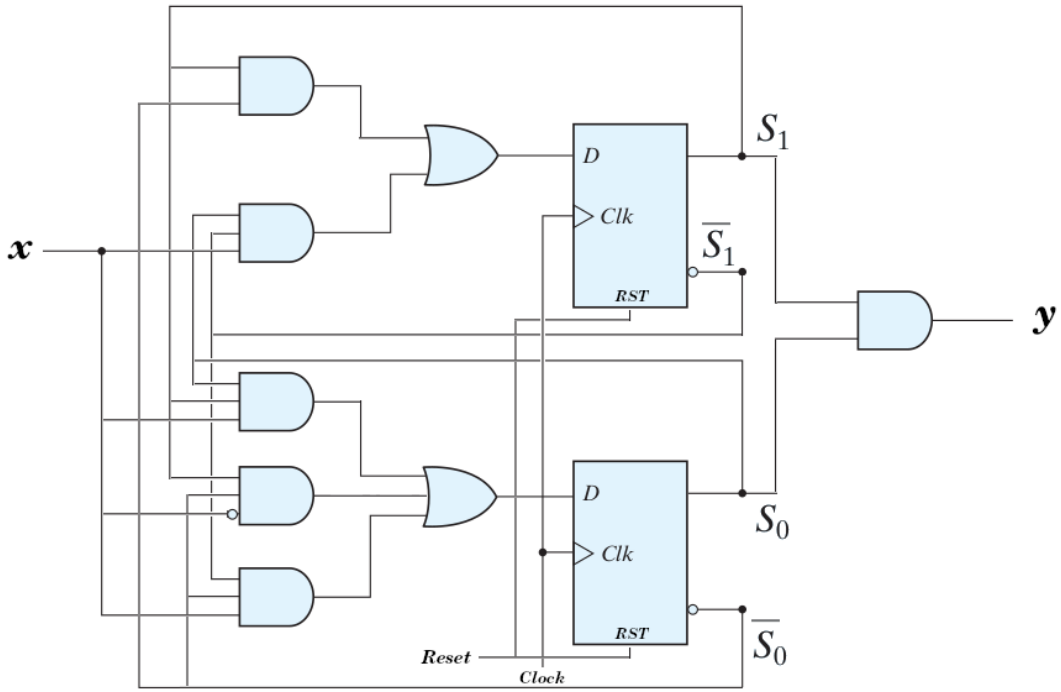
1. Assigning binary codes to sates
2. Obtain the binary-coded state table
3. Obtain the equations for flip flops inputs
4. Obtain the output equations in terms of states

Present State		Output
S_1	S_0	y
0	0	0
0	0	0
0	1	0
0	1	0
1	0	0
1	0	0
1	1	1
1	1	1

$y = S_1 S_0$

Moore Machine Design

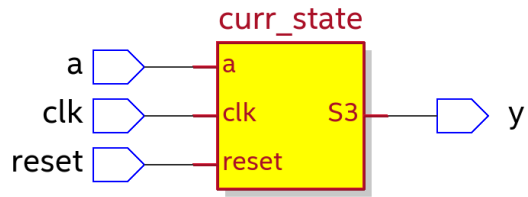
1. Assigning binary codes to sates
2. Obtain the binary-coded state table
3. Obtain the equations for flip flops inputs
4. Obtain the output equations in terms of states
5. Draw the logic circuit diagram

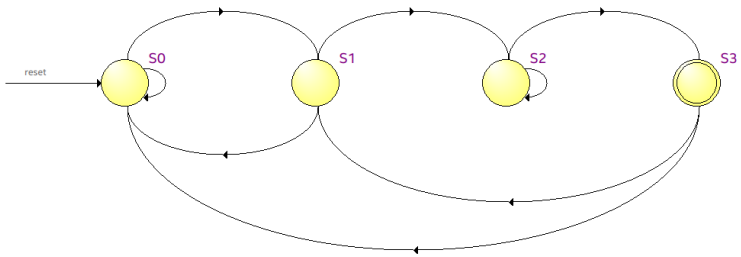


SystemVerilog

- Moore's FSM implementation
- typedef,always_ff, always_comb

```
module moore_FSM   ( input logic clk,
                     input logic reset,
                     input logic a,
                     output logic y    );

    typedef enum logic [1:0] {S0,S1,S2,S3} State;
    State curr_state, next_state;

    always_ff @(posedge clck, posedge reset)
            if (reset) curr_state <=S0
            else curr_state <= next_state;

    always_comb
            case (curr_state)
               S0: if (a) next_state = S1;
                   else   next_state = S0;
               S1: if (a) next_state = S2;
                   else   next_state = S0;
               S2: if (a) next_state = S2;
                   else   next_state = S3;
               S3: if (a) next_state = S1;
                   else   next_state = S0;
            default: next_state = S0;
            endcase

    assign y = (state==S3);
endmodule
```

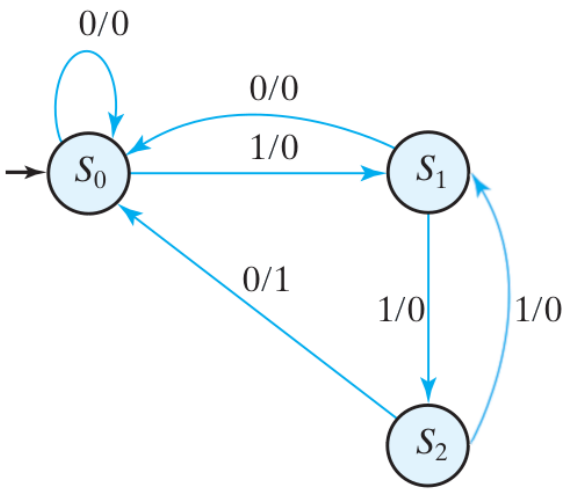




Mealy Machine Design

1. Assigning binary codes to sates

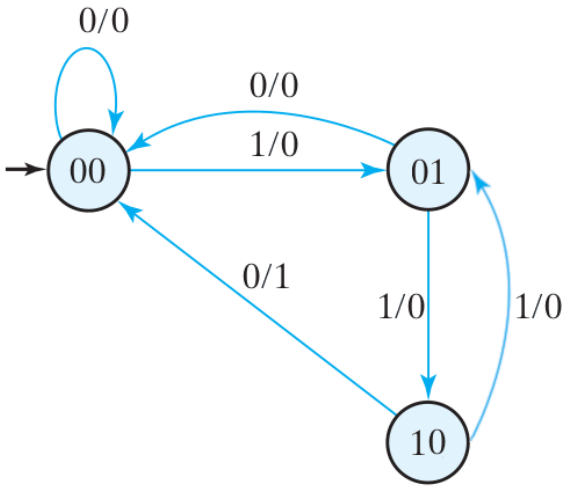
$$S_0 \rightarrow 00$$
$$S_1 \rightarrow 01$$
$$S_2 \rightarrow 10$$



- State coding is not unique and will affect the design

Mealy Machine Design

1. Assigning binary codes to sates
2. Obtain the binary-coded state table



Mealy Machine Design

1. Assigning binary codes to sates
2. Obtain the binary-coded state table
3. Obtain the equations for flip flops inputs and outputs

Present State		Input	Next State		Output
S_1	S_0	x	S_1'	S_0'	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	X	X	X	X

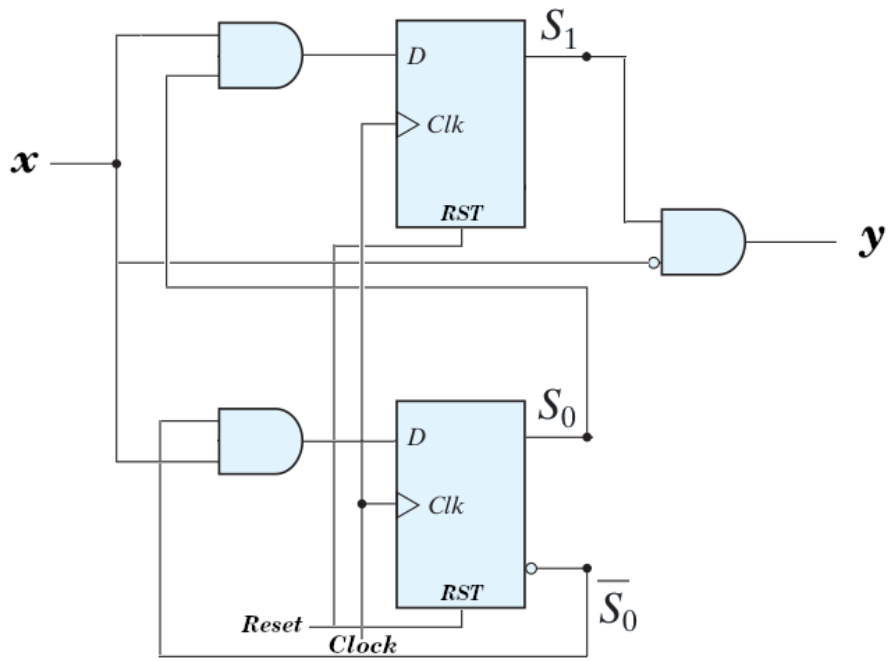
$S_0' = \overline{S_0}x$

$S_1' = S_0x$

$y = S_1\overline{x}$

Mealy Machine Design

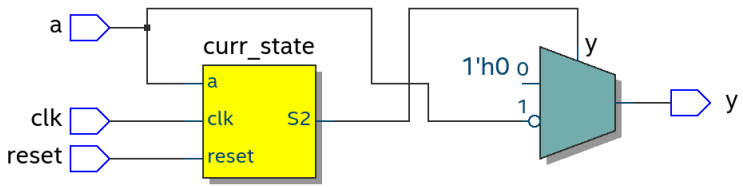
1. Assigning binary codes to sates
2. Obtain the binary-coded state table
3. Obtain the equations for flip flops inputs and outputs
4. Draw the logic circuit diagram

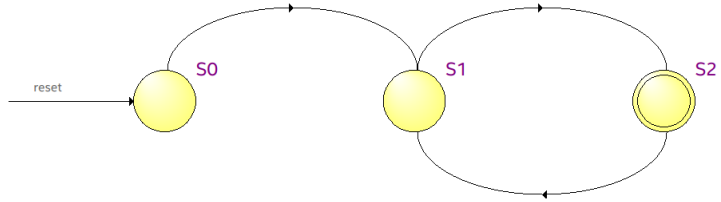


SystemVerilog

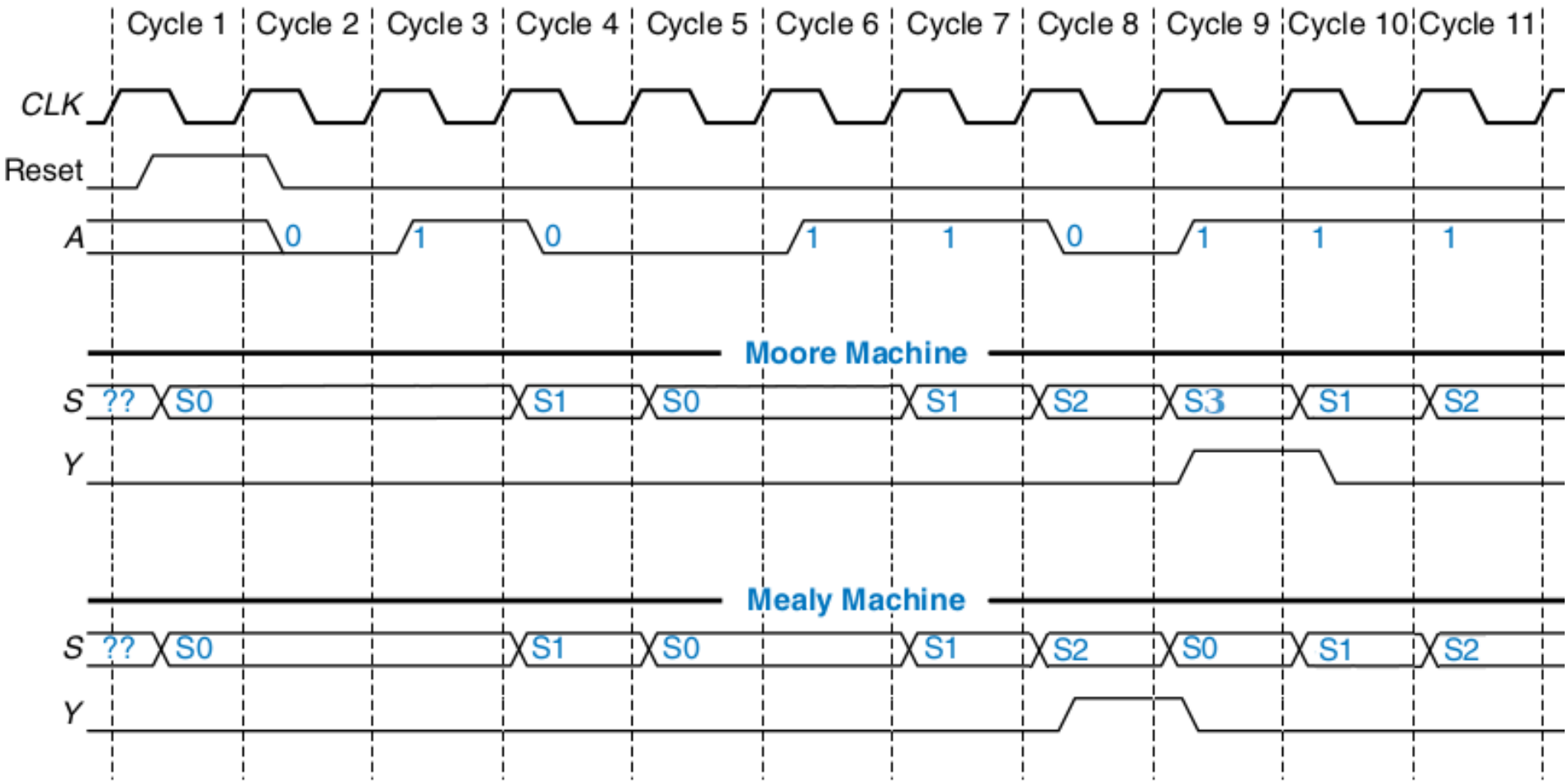
- Mealy's FSM implementation

```
module mealy_FSM   ( input  logic clk,
                     input  logic reset,
                     input  logic a,
                     output logic y    );

typedef enum logic [1:0] {S0,S1,S2} State;
State curr_state, next_state;

always_ff @(posedge clk, posedge reset)
        if (reset) curr_state <= S0;
        else curr_state <= next_state;

always_comb
        case (curr_state)
           S0: if (a) next_state = S1;
               else   next_state = S0;
           S1: if (a) next_state = S2;
               else   next_state = S0;
           S2: if (a) next_state = S1;
               else   next_state = S0;

        default: next_state = S0;
        endcase

always_comb
        case (curr_state)
           S0: y = 1'b0;
           S1: y = 1'b0;
           S2: y = (a) ? 1'b0 : 1'b1;
        default: y = 1'b0;
        endcase

endmodule
```

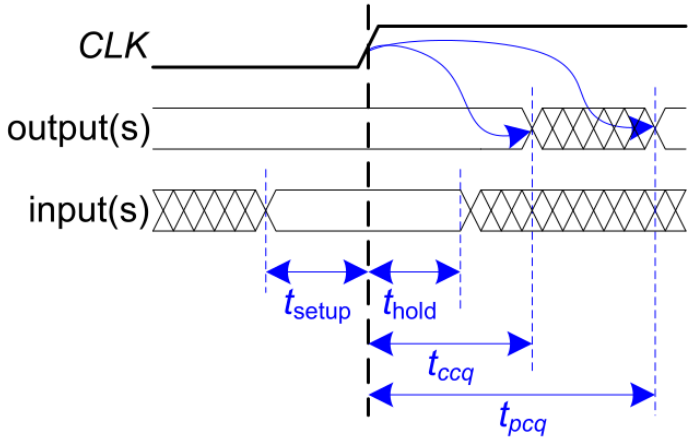




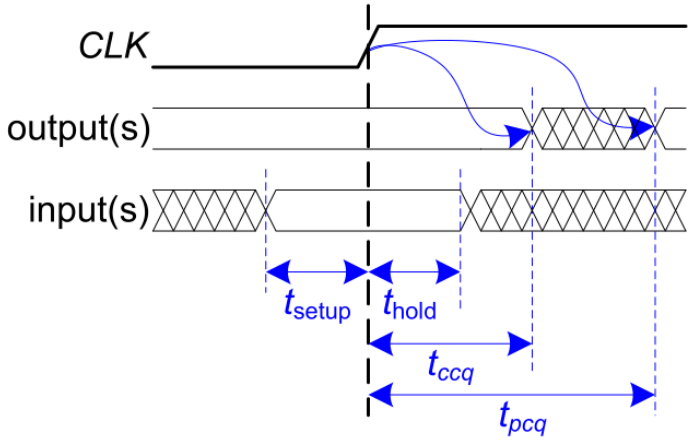
Timing Diagram



Timing Specification

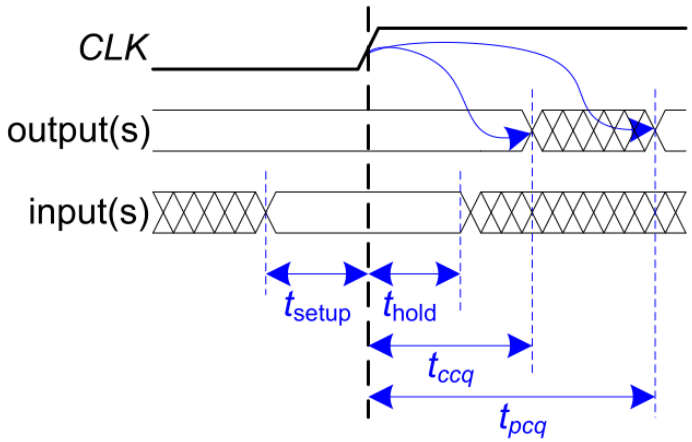


Timing Specification



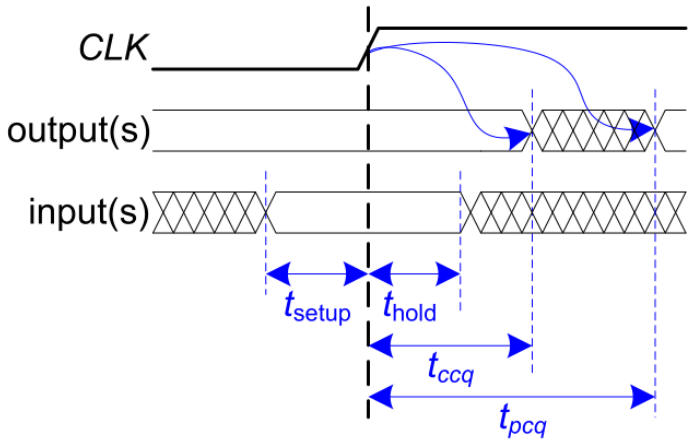
- t_{setup}: It's the time input must have stabilized, before the rising edge

Timing Specification



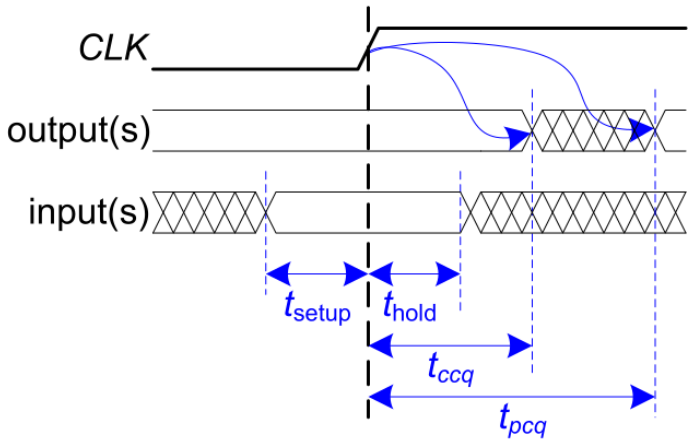
- t_{setup}: It's the time input must have stabilized, before the rising edge
- t_{hold} : It's the time input must have stabilized, after the rising edge

Timing Specification



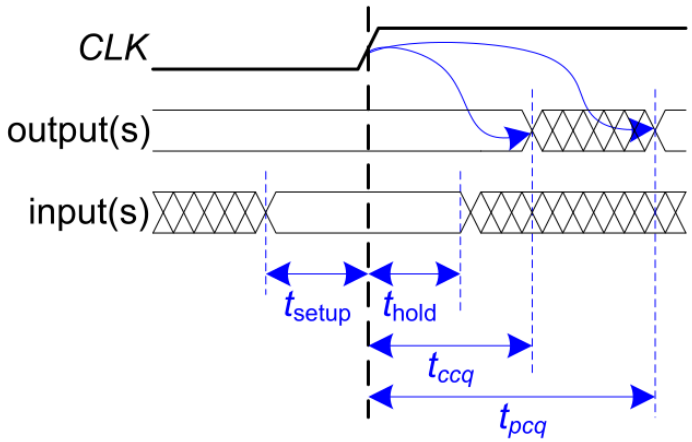
- t_{setup}: It's the time input must have stabilized, before the rising edge
- t_{hold} : It's the time input must have stabilized, after the rising edge
- t_{apr} : It's the sum $t_{setup} + t_{hold}$

Timing Specification



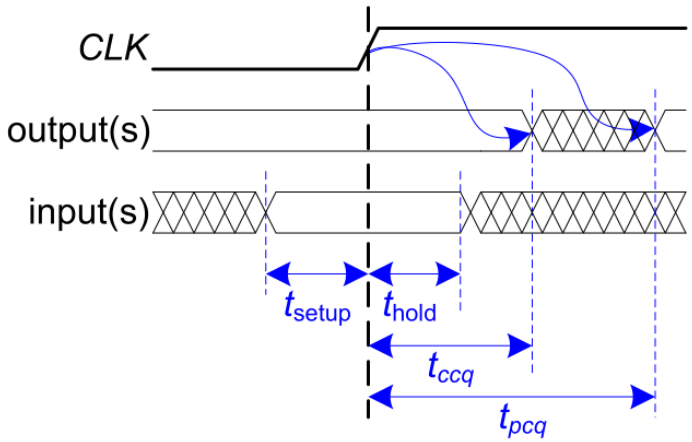
- t_{setup}: It's the time input must have stabilized, before the rising edge
- t_{hold} : It's the time input must have stabilized, after the rising edge
- t_{apr} : It's the sum $t_{setup} + t_{hold}$
- t_{ccq}: It's the contamination delay from the clock to output of the flip flops

Timing Specification



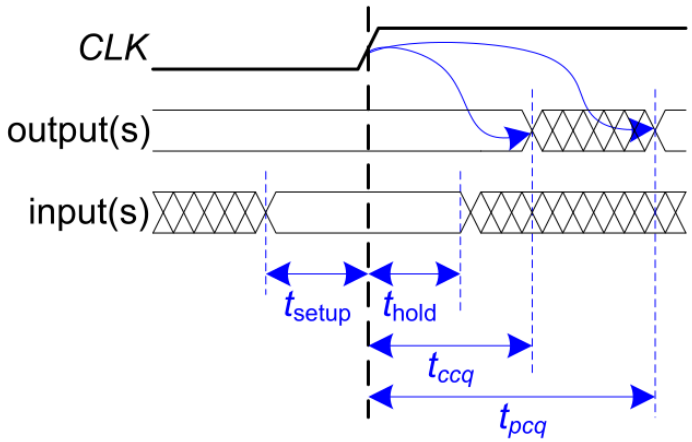
- t_{setup}: It's the time input must have stabilized, before the rising edge
- t_{hold} : It's the time input must have stabilized, after the rising edge
- t_{apr} : It's the sum $t_{setup} + t_{hold}$
- t_{ccq}: It's the contamination delay from the clock to output of the flip flops
- t_{pcq}: It's the propagation delay from the clock to output of the flip flops

Timing Specification



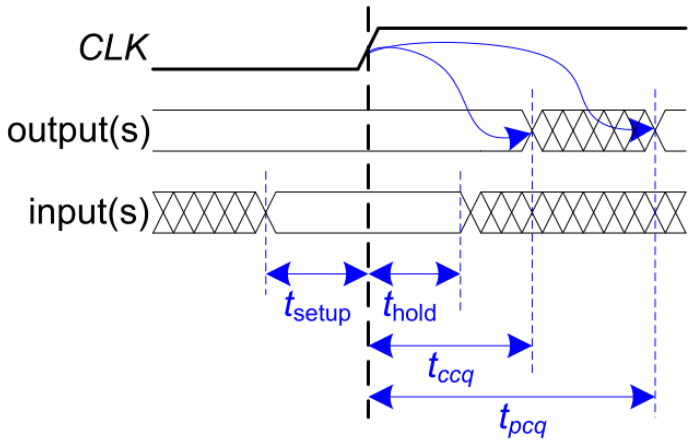
- t_{setup}: It's the time input must have stabilized, before the rising edge
- t_{hold} : It's the time input must have stabilized, after the rising edge
- t_{apr} : It's the sum $t_{setup} + t_{hold}$
- t_{ccq}: It's the contamination delay from the clock to output of the flip flops
- t_{pcq}: It's the propagation delay from the clock to output of the flip flops
- The clock cycle T_c must be long enough for all the signals to be settled

Timing Specification



- t_{setup}: It's the time input must have stabilized, before the rising edge
- t_{hold} : It's the time input must have stabilized, after the rising edge
- t_{apr} : It's the sum $t_{setup} + t_{hold}$
- t_{ccq}: It's the contamination delay from the clock to output of the flip flops
- t_{pcq}: It's the propagation delay from the clock to output of the flip flops
- The clock cycle T_c must be long enough for all the signals to be settled
- The clock frequency is given by $F_c = \frac{1}{T_c}$

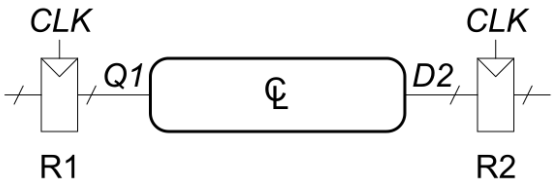
Timing Specification



- t_{setup}: It's the time input must have stabilized, before the rising edge
- t_{hold} : It's the time input must have stabilized, after the rising edge
- t_{apr} : It's the sum $t_{setup} + t_{hold}$
- t_{ccq}: It's the contamination delay from the clock to output of the flip flops
- t_{pcq}: It's the propagation delay from the clock to output of the flip flops
- The clock cycle T_c must be long enough for all the signals to be settled
- The clock frequency is given by $F_c = \frac{1}{T_c}$

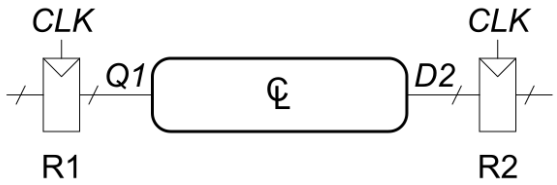
Timing Specification

- Determine the minimum clock cycle in the following circuit

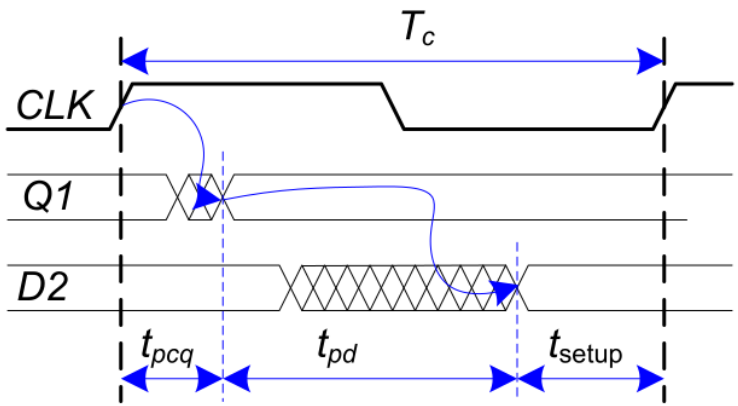


Timing Specification

- Determine the minimum clock cycle in the following circuit

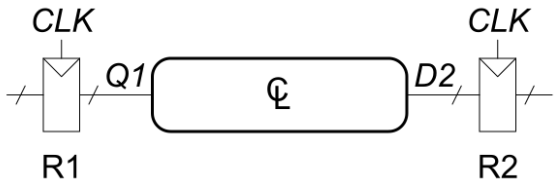


- Based on the following timing constraints,

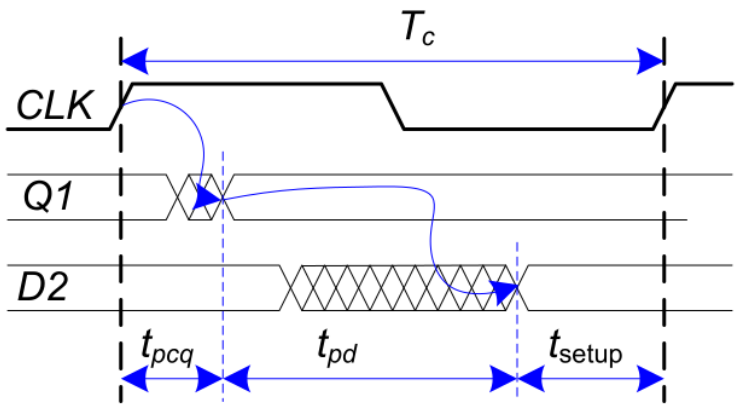


Timing Specification

- Determine the minimum clock cycle in the following circuit



- Based on the following timing constraints,

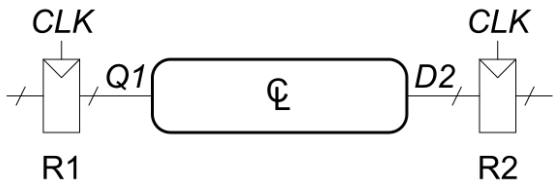


We must have:

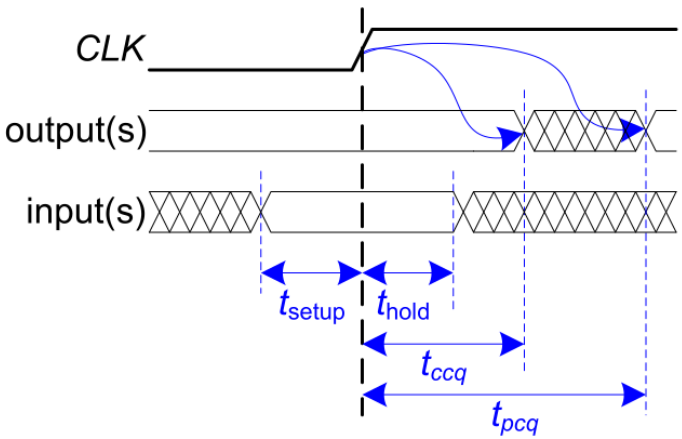
$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

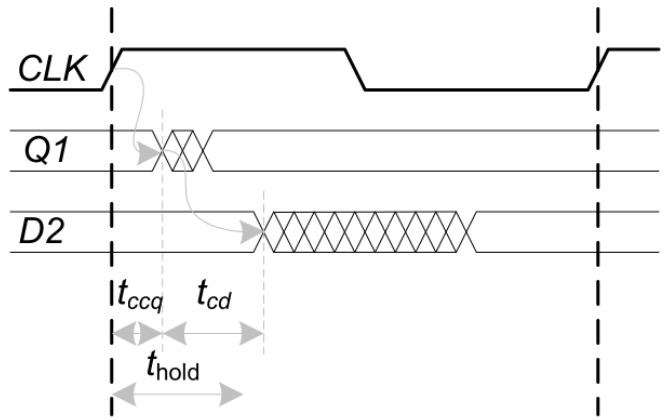
Timing Specification

- Determine the minimum clock cycle in the following circuit



- Based on the following timing constraints,



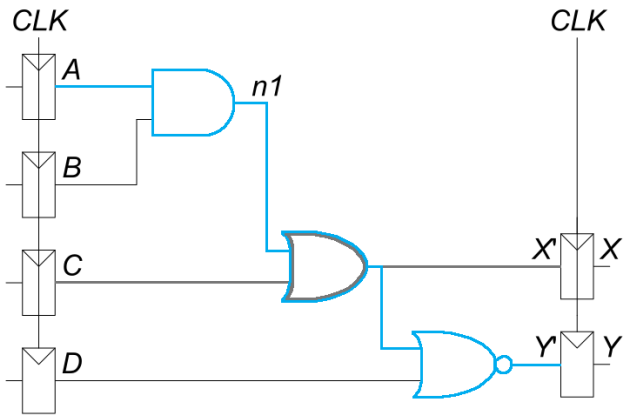


We must have:

$$T_c \geq t_{pcq} + t_{pd} + t_{setup} \qquad t_{hold} \leq t_{ccq} + t_{cd}$$

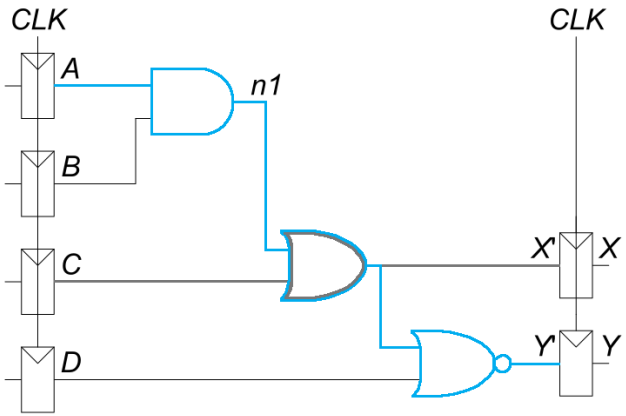
Timing Specification (Example)

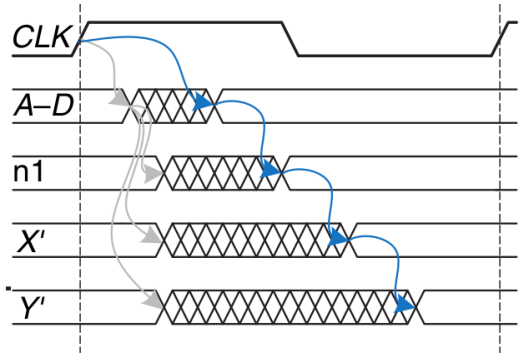
- Determine the max and min delay for the given circuit



Timing Specification (Example)

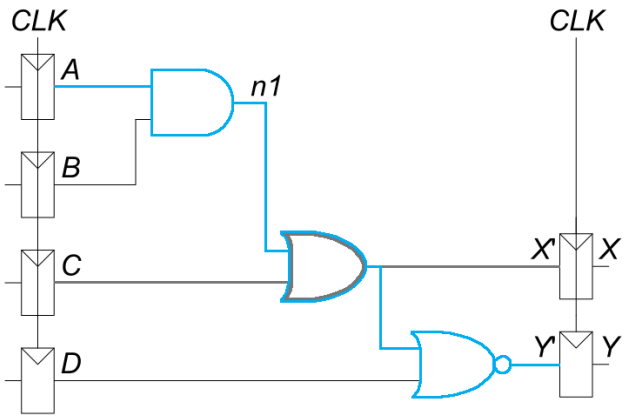
- Determine the max and min delay equation for the given circuit

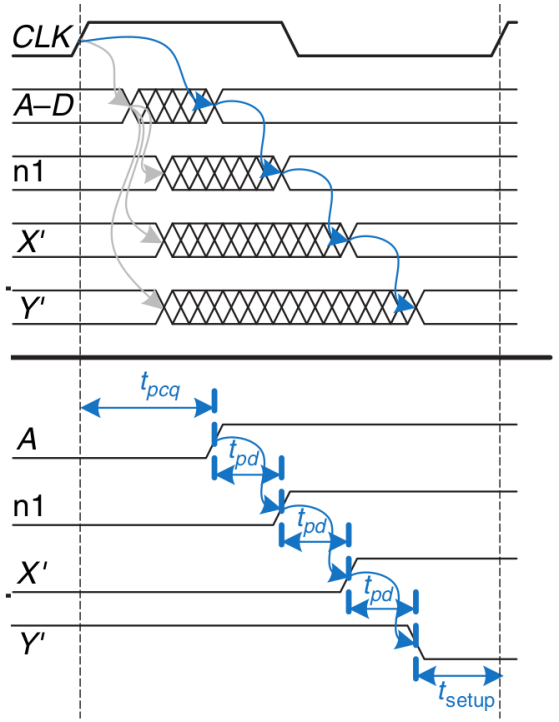




Timing Specification (Example)

- Determine the max and min delay equation for the given circuit

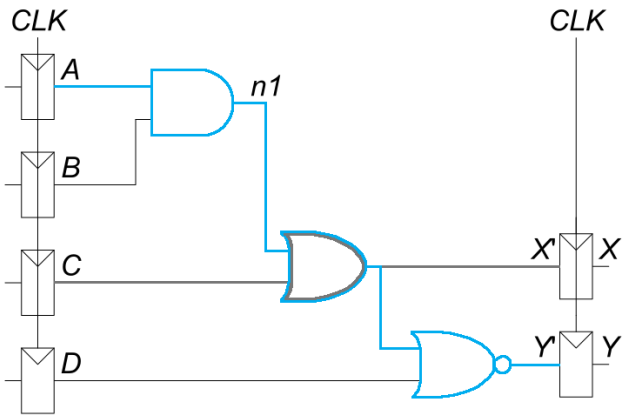


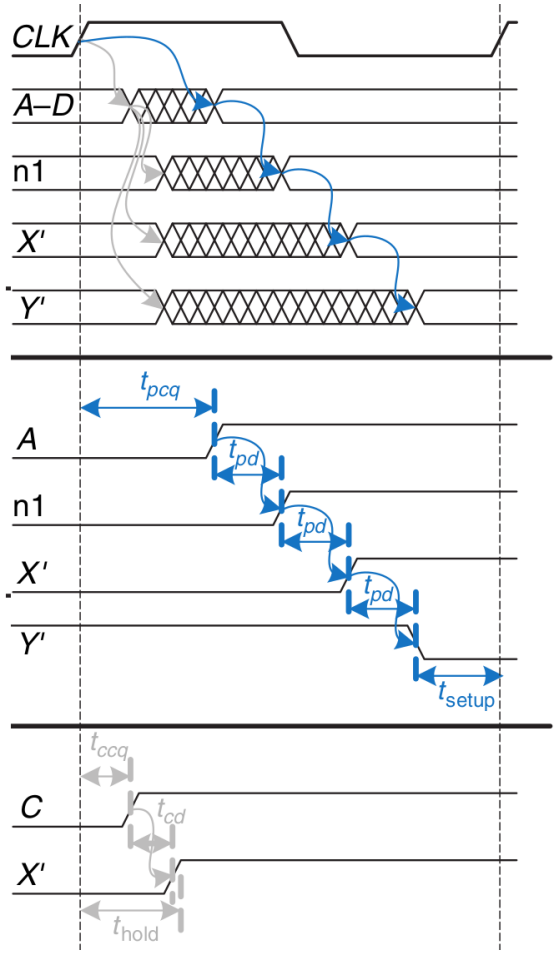


$$T_c \geq t_{pcq} + 3\,t_{pd} + t_{setup}$$

Timing Specification (Example)

- Determine the max and min delay equation for the given circuit



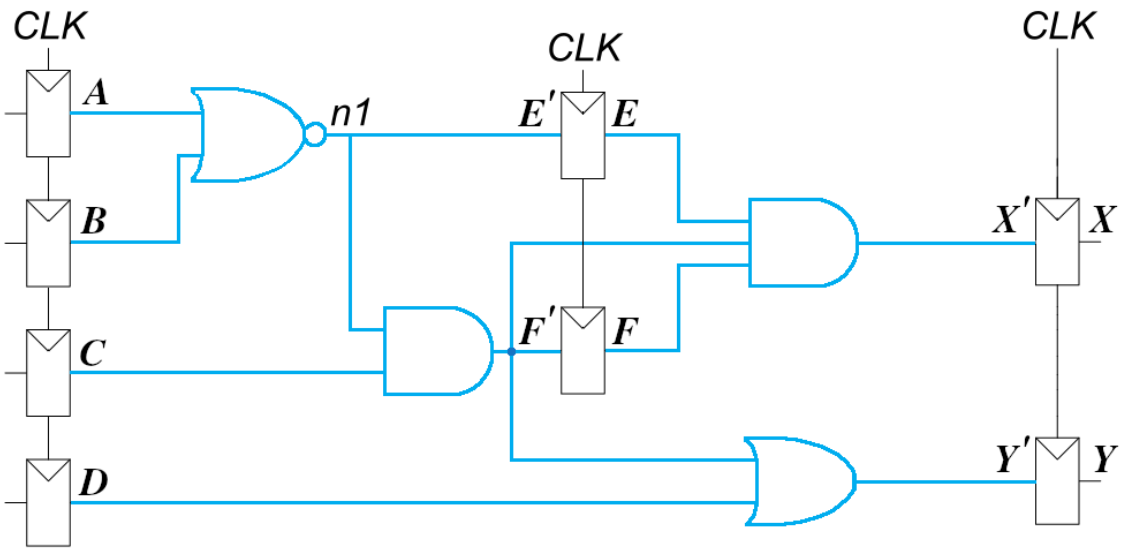


$$T_c \geq t_{pcq} + 3\,t_{pd} + t_{setup}$$

$$t_{hold} \leq t_{ccq} + t_{cd}$$

Question

- Determine the maximum F_c and maximum tolerable t_{hold} for the following circuit



Flip Flop	t_{ccq} (ps)	t_{pcq} (ps)	t_{setup} (ps)	t_{hold} (ps)
	30	80	50	?

Gate	t_{pd} (ps)	t_{cd} (ps)
2-input NAND	20	15
3-input NAND	30	25
2-input NOR	30	25
2-input OR	40	30

$$F_{max} = ?$$

$$t_{hold} = ?$$