Logic Design
Lecture 7 Sequential Circuits

EE2280, Fall 2017

Jenny Yi-Chun Liu

jennyliu@ee.nthu.edu.tw

Outline

- Digital systems and information
- Boolean algebra
- Verilog introduction
- Combinational logic design
- Combinational building blocks
- Arithmetic functions
- Sequential logic
- Datapath sequential logic
- Memory

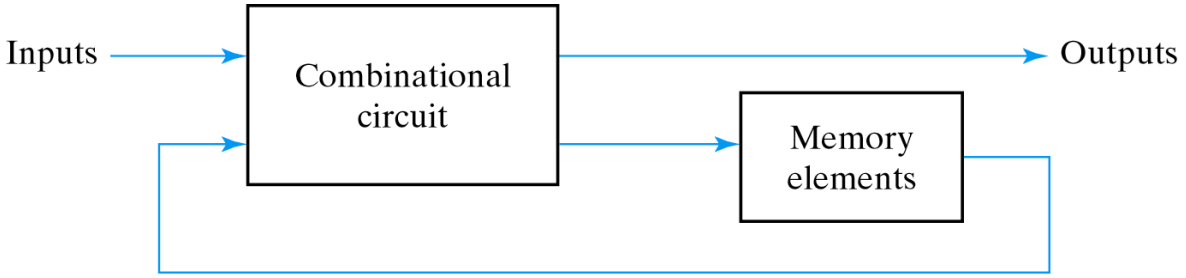
Overview

- Introduction
- Storage Element: Latches
- Storage Element: Flip-Flops
- Analysis of Clocked Sequential Circuits
- Design of Clocked Sequential Circuits
- Verilog Examples

Introduction

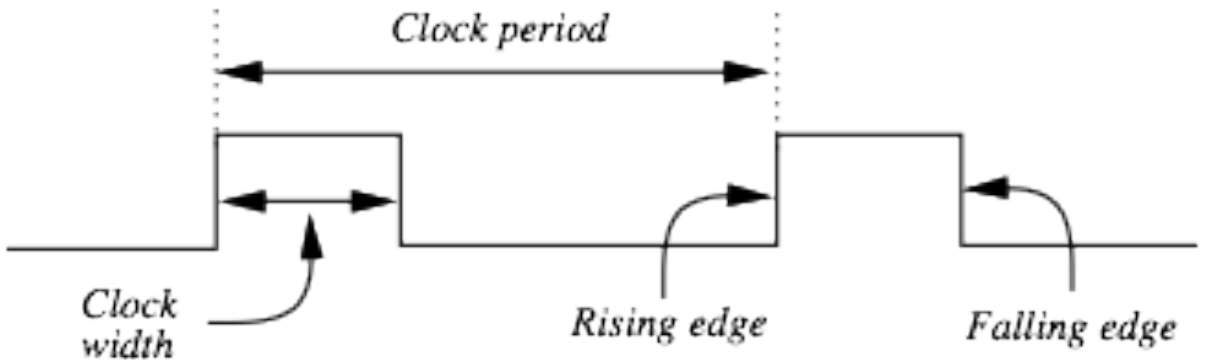
Introduction to Sequential Circuits

- A Sequential circuit contains:
  - Combinational logic
  - Storage element (flip-flops, latches)
- Inputs and present state determine the outputs and next state.
  - Binary information stored in the memory elements defines the state of the sequential circuit.
- Logic diagram of a sequential circuit:



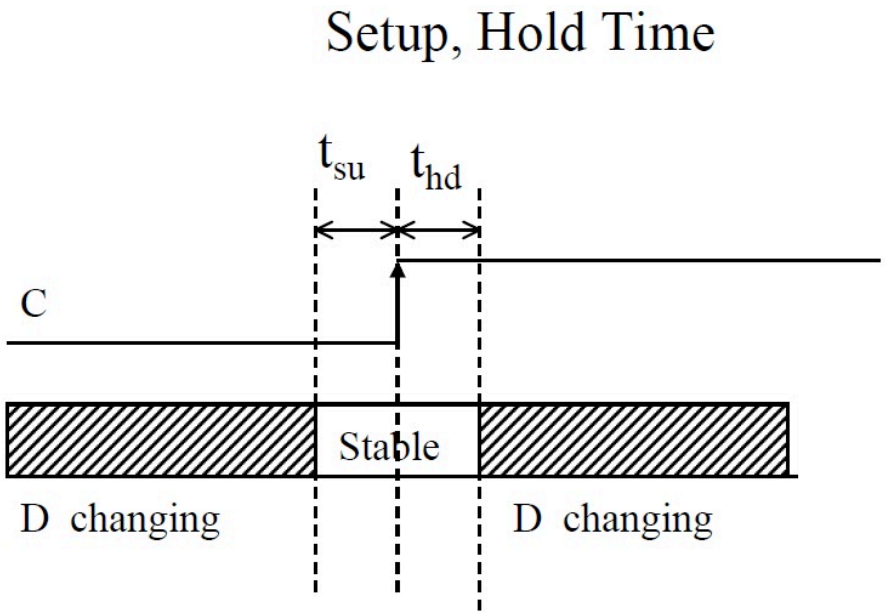
Synchronous vs. Asynchronous Sequential Circuits

- Timing of the respective signals differ.
- Synchronous sequential circuit: inputs and state are only defined at discrete time.
- Asynchronous sequential circuit: inputs and state can change at any time.
- Clock: a periodic train of clock pulses generated by timing device and distributed throughout the system.



Setup Time/Hold Time

- Setup time: Input must be maintained at a minimum amount of time prior to the clock edge.
- Hold time: Input must be maintained at a minimum amount of time after the clock edge.



Storage Elements: Latches

Latches

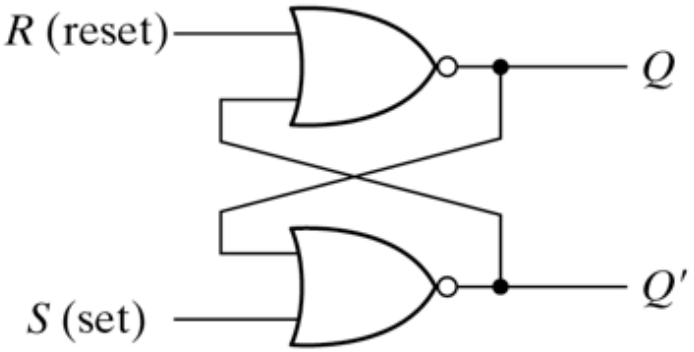
- Storage element: maintain a binary state indefinitely until directed by an input signal to switch state.

- Most basic storage element: lathes
  - Latches are asynchronous sequential circuit. Its state changes whenever inputs change.
  - Latches have
  - Common latches:

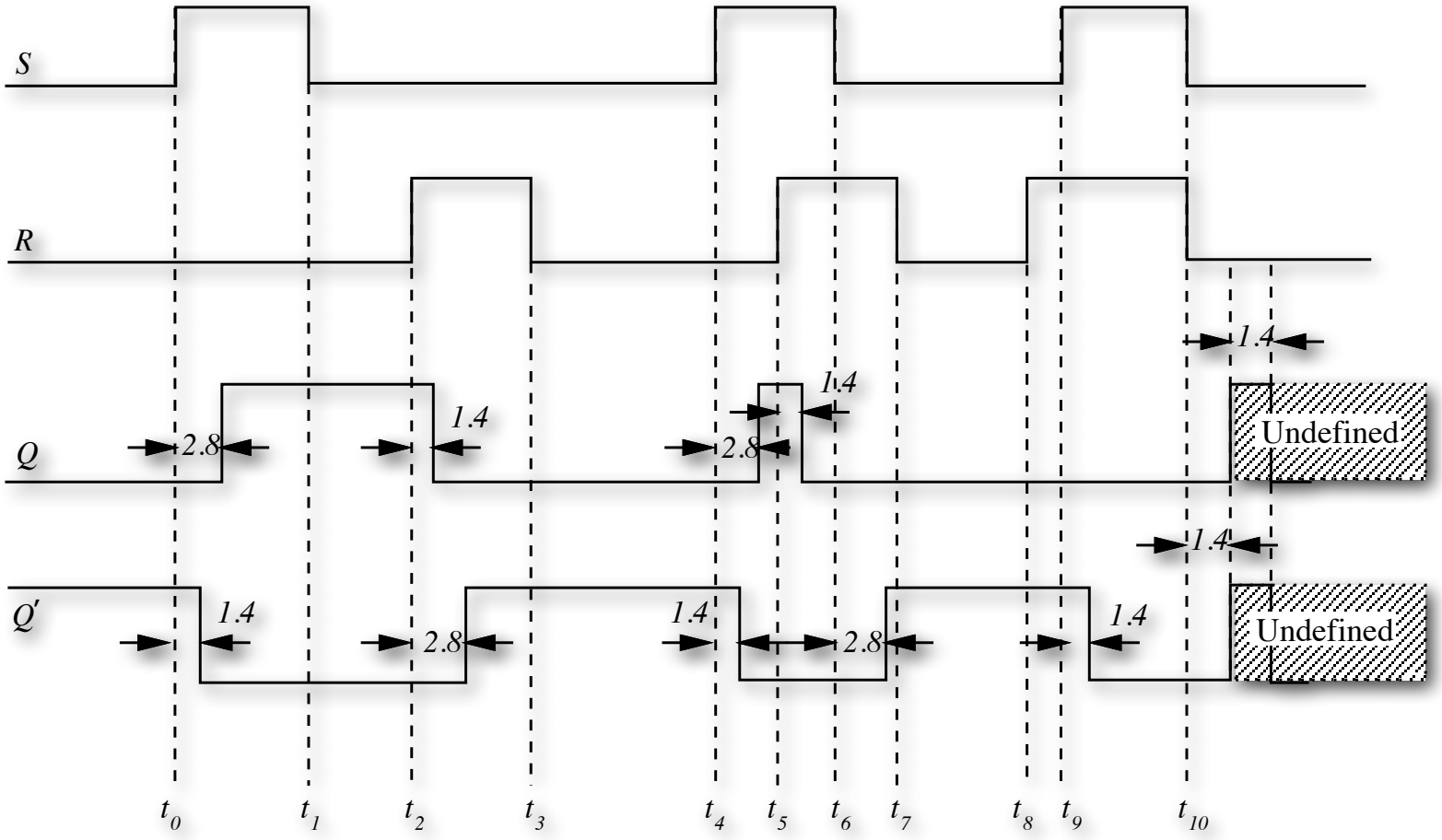
SR Latch

- SR latch can be formed by two cross-coupled NORs or NANDs.
- Two inputs:
- Two states:
- Under normal conditions, both inputs (R, S) = 0 unless the state is to be changed.

SR Latch with NORs

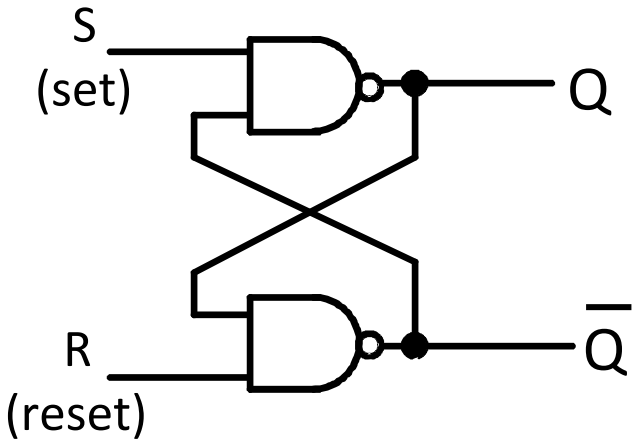


SR Latch with NORs
Timing Diagram

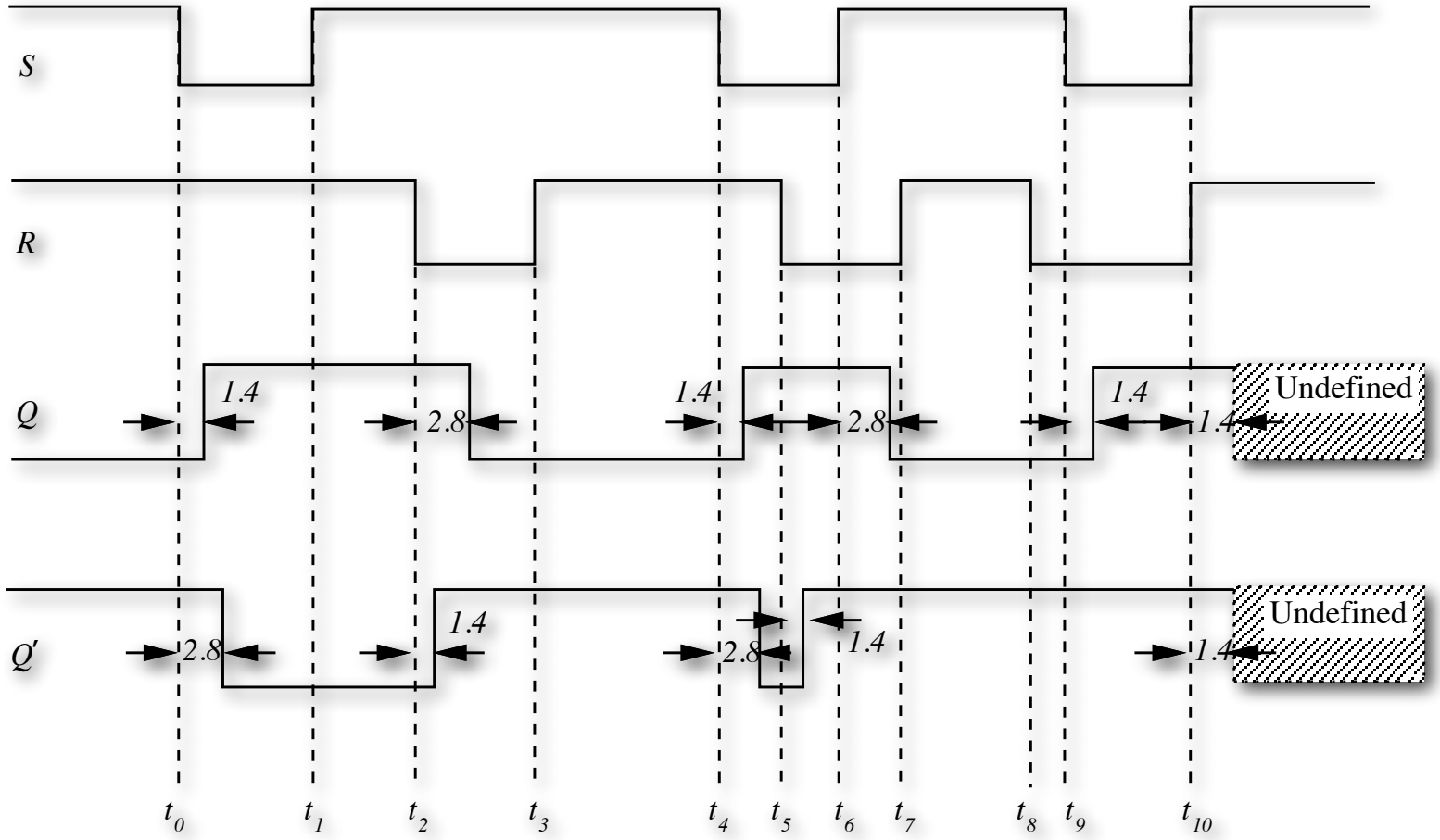


SR Latch with NANDs

- Under normal conditions, both inputs (R, S) = 1 unless the state is to be changed.

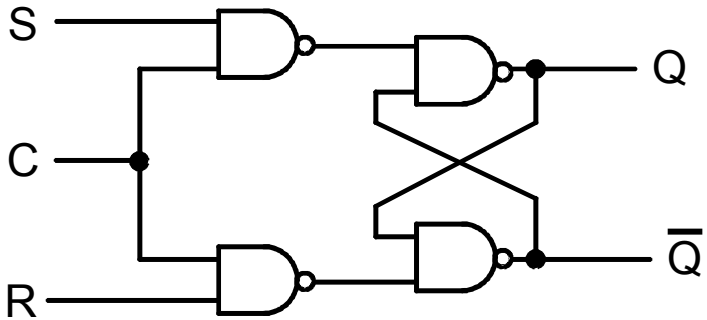


SR Latch with NANDs Timing Diagram



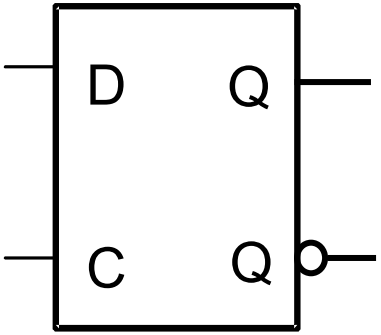
Clocked SR Latch

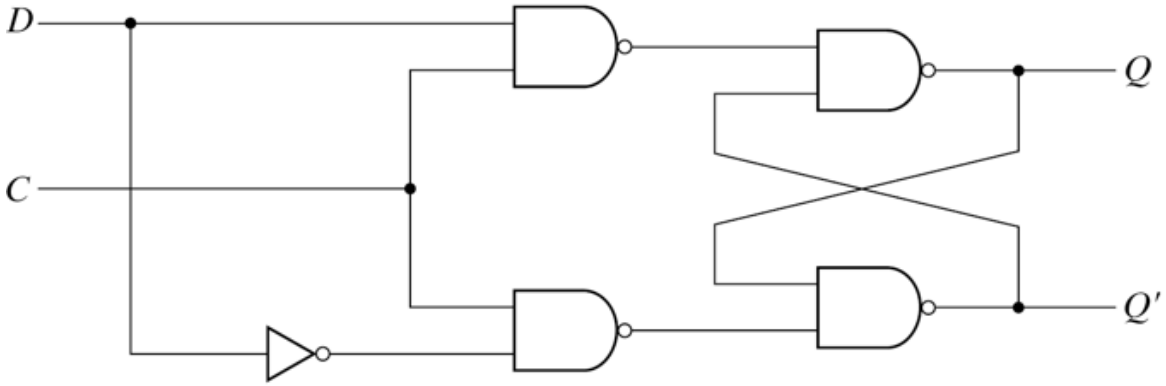
- Control input (C): determines when the state can be changed.



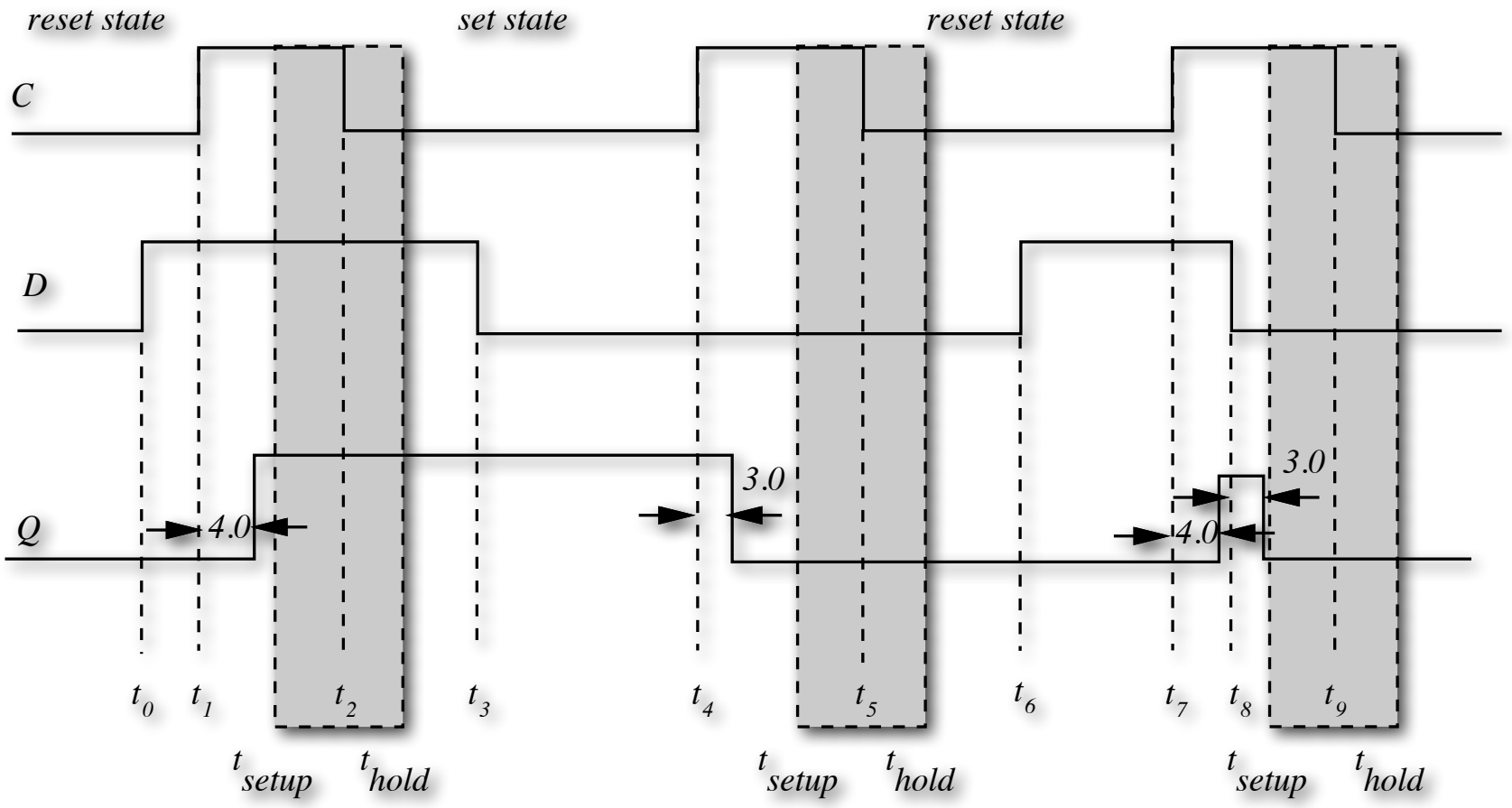
D Latch

- D latch has two inputs:
- No undefined state.





D Latch Timing Diagram



Storage Elements: Flip-Flops

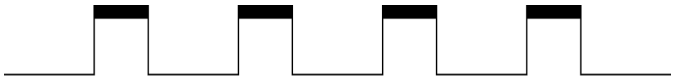
Flip-Flops Outline

- The latch timing problem
- Master-slave flip-flop
- Edge-triggered flip-flop
- Standard symbols for storage elements
- Direct inputs

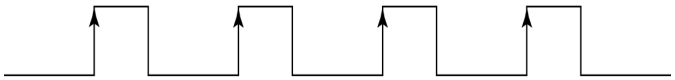
Trigger

- Trigger
  - The state of a latch or flip-flop is switched by a change of the control input.
- Level-triggered
  - The state transition starts as soon as the clock is logic 1 (positive level-sensitive) or logic 0 (negative level-sensitive) level.
- Edge-triggered
  - The state transition starts only at positive (positive edge-triggered) or negative edge (negative edge-triggered) of the clock signal.

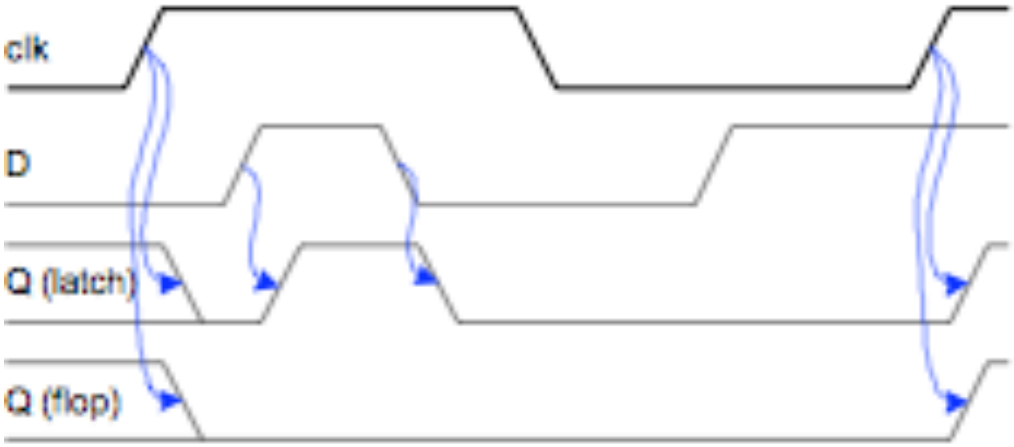
Level-Triggered vs. Edge-Triggered



(a) Response to positive level



(b) Positive-edge response



The Latch Timing Problem (1/3)

- In a sequential circuit, paths may exist through combinational logic:
  - From one storage element to another
  - From a storage element back to the same storage element
- The combinational logic between a latch output and a latch input may be as simple as an interconnect.
- For a clocked D latch, the output Q depends on the input D whenever the clock input C has value 1.

The Latch Timing Problem (2/3)

- Consider the following circuit:

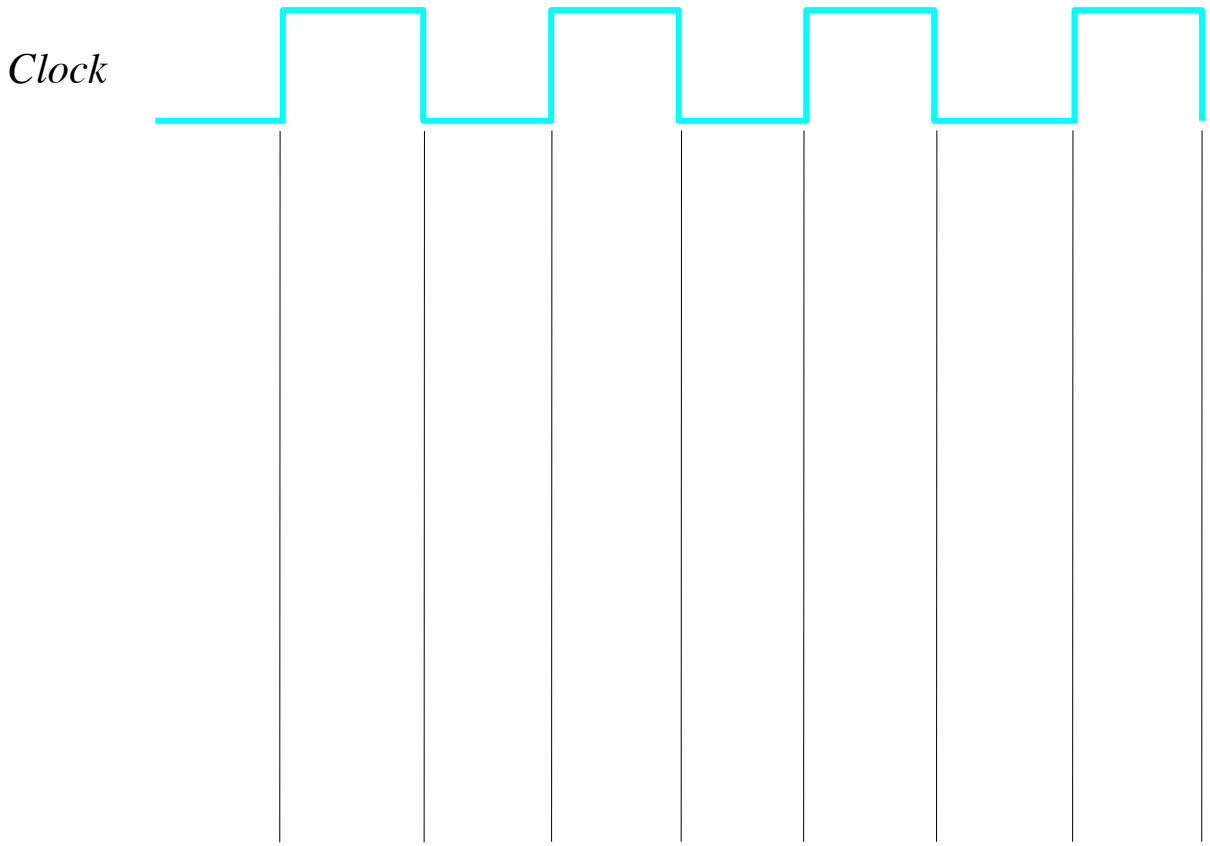
The Latch Timing Problem (3/3)

- Solution: break the closed path within the storage element.
- The commonly used, path-breaking circuits replaced the D-latch.
  1. Master-slave FF: inputs present to FF when clock = 1, state of FF changes when clock=0.
  2. Edge-triggered FF: FF is triggered only when clock transits from 0→1 or 1→0.

SR Master-Slave Flip-Flop

- Two clocked SR latches in series. The clock on the second latch is inverted.
- Input is observed when C = 1.
- Output is changed by the second latch when C = 0.
- The path from input to output is broken by the difference in the clock values of two latches.

SR Master-Slave FF Timing Diagram



Flip-Flop Problem

- S and/or R are permitted to change while C = 1.
  - Suppose Q = 0 and S goes to 1 and then back to 0 with R remaining at 0.
    - The master latch sets to 1.
    - A 1 is transferred to the slave.
  - Suppose Q = 0 and S goes to 1 and back to 0 and R goes to 1 and back to 0.
    - The master latch sets and then resets.
    - A 0 is transferred to the slave.
  - This behavior is called *1s catching.*

Flip-Flop Solution

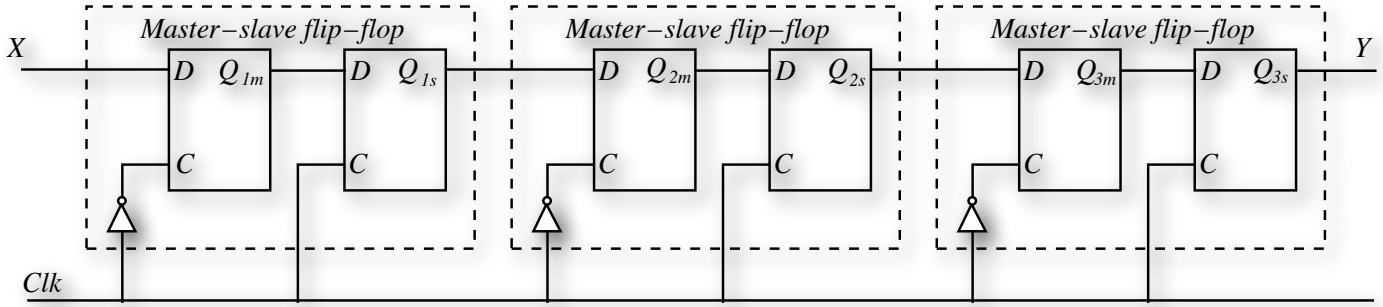
- Use edge-triggering instead of master-slave.
- An edge-triggered flip-flop ignores the pulse while it is at a constant level and triggers only during a transition of the clock signal.
- Edge-triggered flip-flops can be built directly at the electronic circuit level.
- A master-slave D flip-flop which also exhibits edge-triggered behavior can be used.

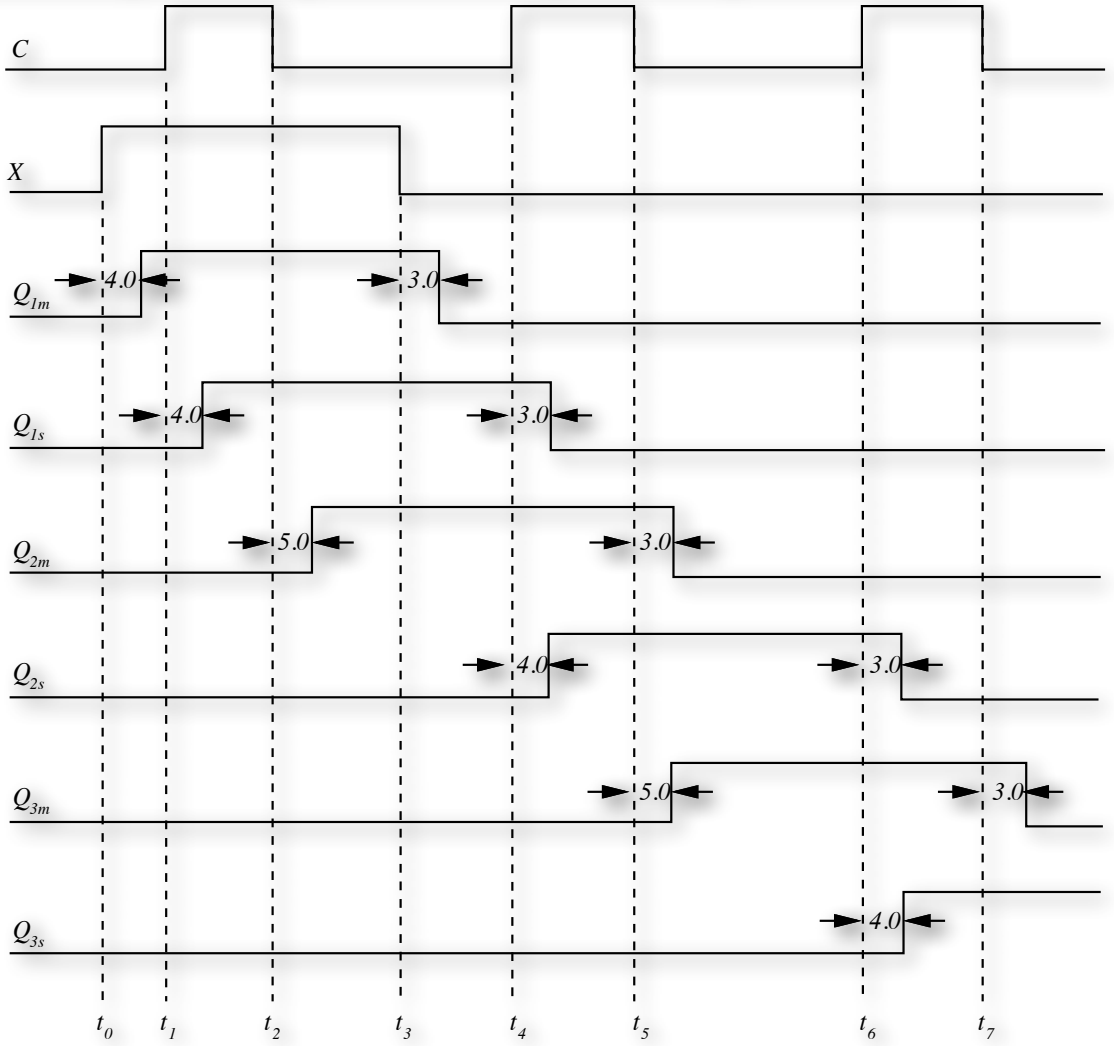
Edge-Triggered D Flip-Flop

- The edge-triggered D flip-flop is the same as the master-slave D flip-flop.
- The change of the D flip-flop output is associated with the negative edge at the end of the pulse.

Edge-Triggered DFF Timing

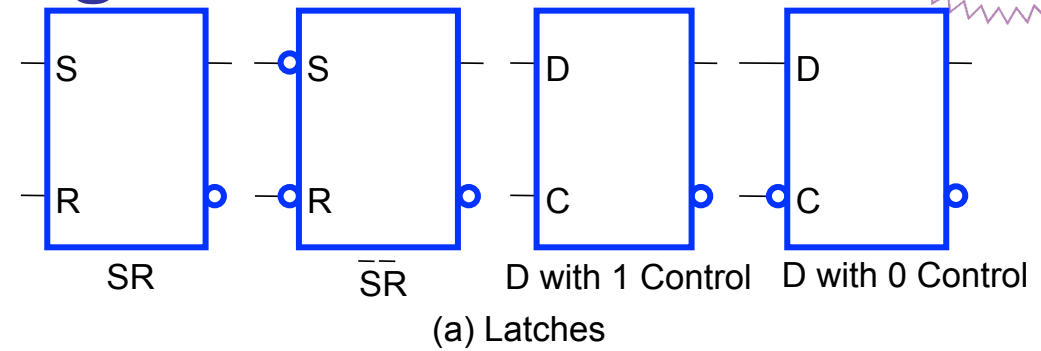
- A master-slave D flip-flop is formed by two separate latches
  - A master D latch (negative level sensitive)
  - A slave D latch (positive level sensitive)

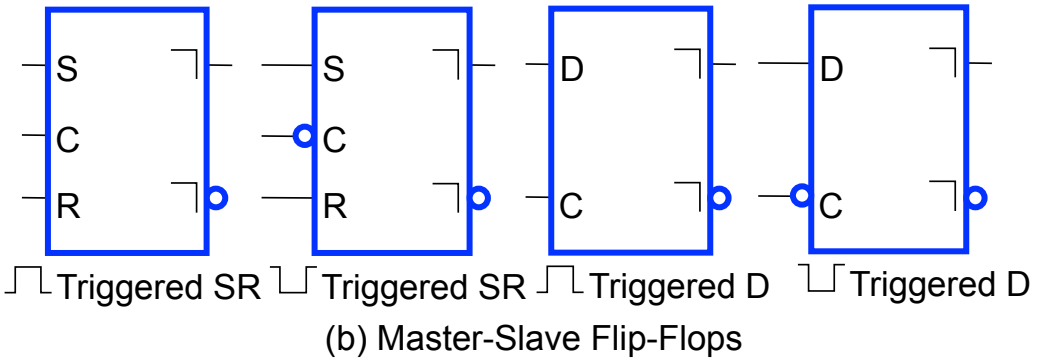

X
Master–slave flip–flop
D
Q 1m
C
D
Q 1s
C
Master–slave flip–flop
D
Q 2m
C
D
Q 2s
C
Master–slave flip–flop
D
Q3m
C
D
Q3s
C
Y
Clk

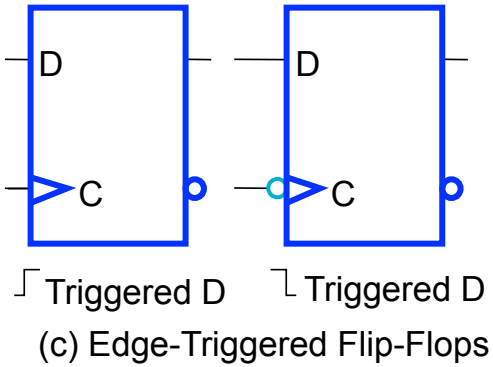

C
X
Q1m
4.0
3.0
Q1s
4.0
3.0
Q2m
5.0
3.0
Q2s
4.0
3.0
Q3m
5.0
3.0
Q3s
4.0
t0
t1
t2
t3
t4
t5
t6
t7

Standard Symbols for Storage Elements

- Inputs: at left
- Outputs: at right
- Positive pulse:
- Positive edge:

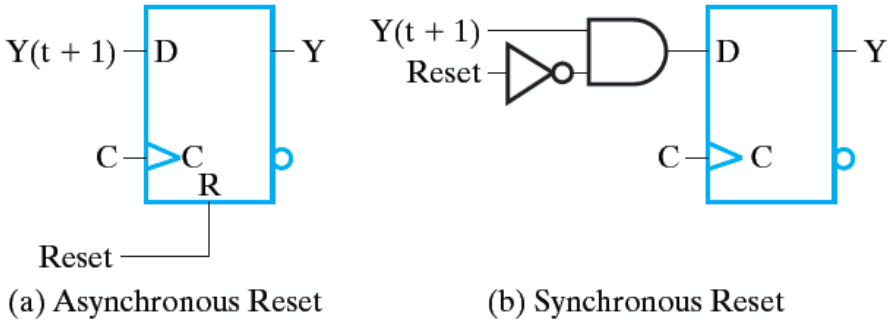






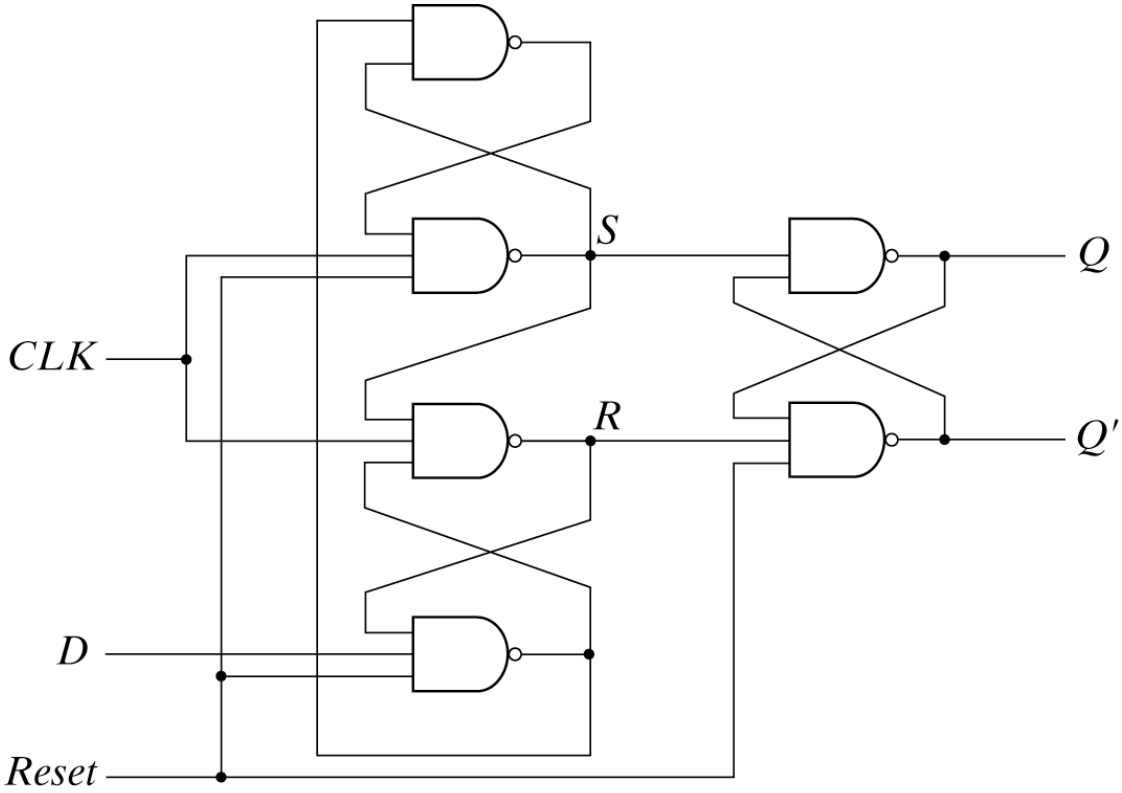
Direct Inputs

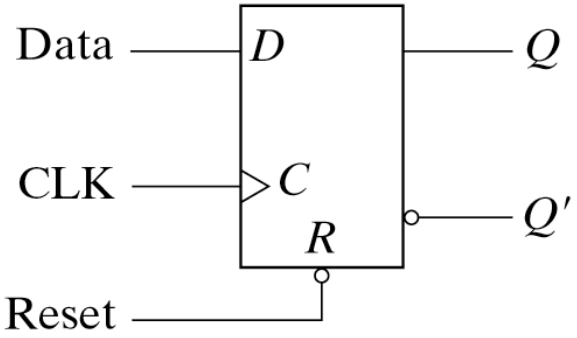
- At power up or at reset, all or part of a sequential circuit usually is initialized to a known state before it begins operation.
- The direct input asynchronously sets the flip-flop.
  - Preset/set: set to 1
  - Reset/clear: set to 0
- The direct input can also be controlled by the clock, called synchronous direct input.



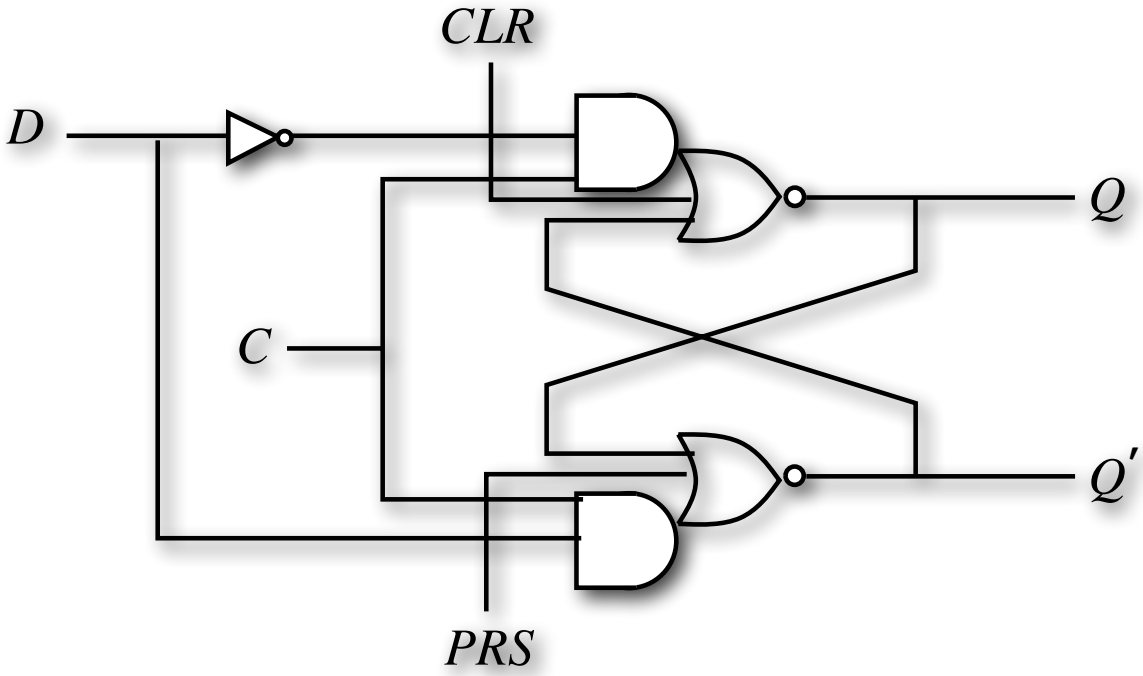
(a) Asynchronous Reset (b) Synchronous Reset

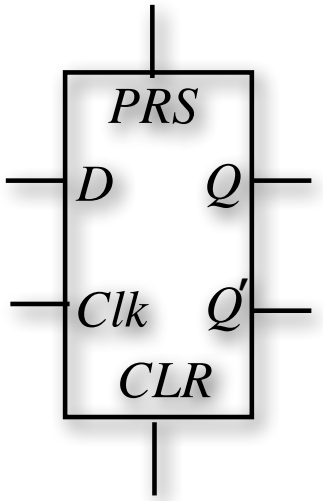
DFF with Direct Input





D Latch with PRS/CLR





Other Flip-Flop Types

- Four major common FFs
  - SR (set-reset)
  - D (data)
  - JK
  - T (toggle)
- Basic descriptors for understanding and using different flip-flop types
  - Characteristic tables
  - Characteristic equations
  - Excitation tables

JK Flip-Flop (1/2)

- JKFF behavior
  - Same as S-R flip-flop with J analogous to S and K analogous to R.
  - Except that J = K = 1 is allowed.
    - For J = K = 1, the flip-flop changes to the opposite state.
  - As a master-slave FF, has same “1s catching” behavior as S-R flip-flop.

JK Flip-Flop (2/2)

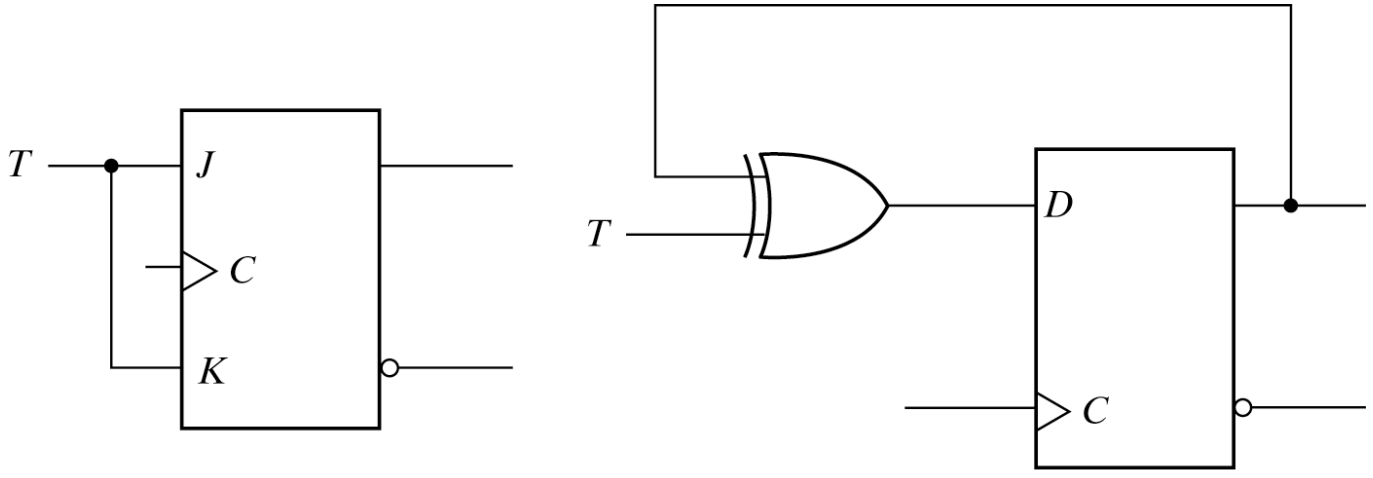
- Implementation
  - To avoid 1s catching behavior, one solution used is to use an edge-triggered D as the core of the flip-flop.

T Flip-Flop (1/2)

- TFF behavior
  - Has a single input T
    - For T = 0, no change to state.
    - For T = 1, changes to opposite state.
- Same as a J-K flip-flop with J = K = T.
- As a master-slave, has same “1s catching” behavior as J-K flip-flop.

T Flip-Flop (2/2)

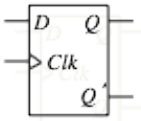
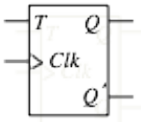
- Implementation
  - To avoid 1s catching behavior, one solution used is to use an edge-triggered D as the core of the flip-flop.



(a) From *JK* flip-flop (b) From *D* flip-flop

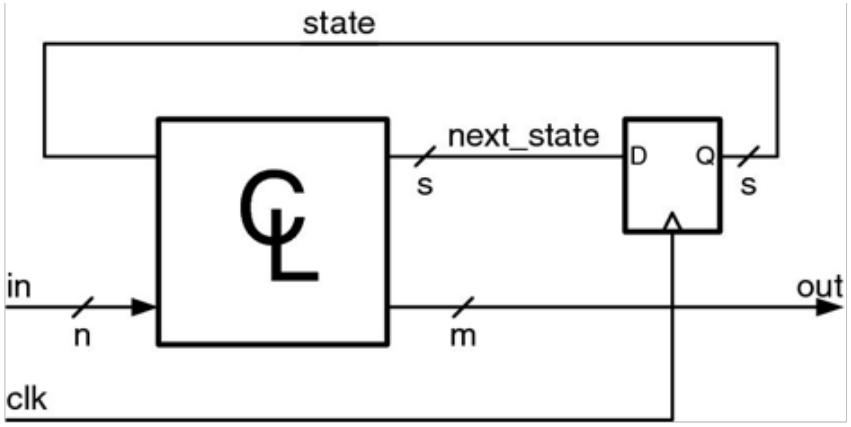
Summary of Flip-Flops

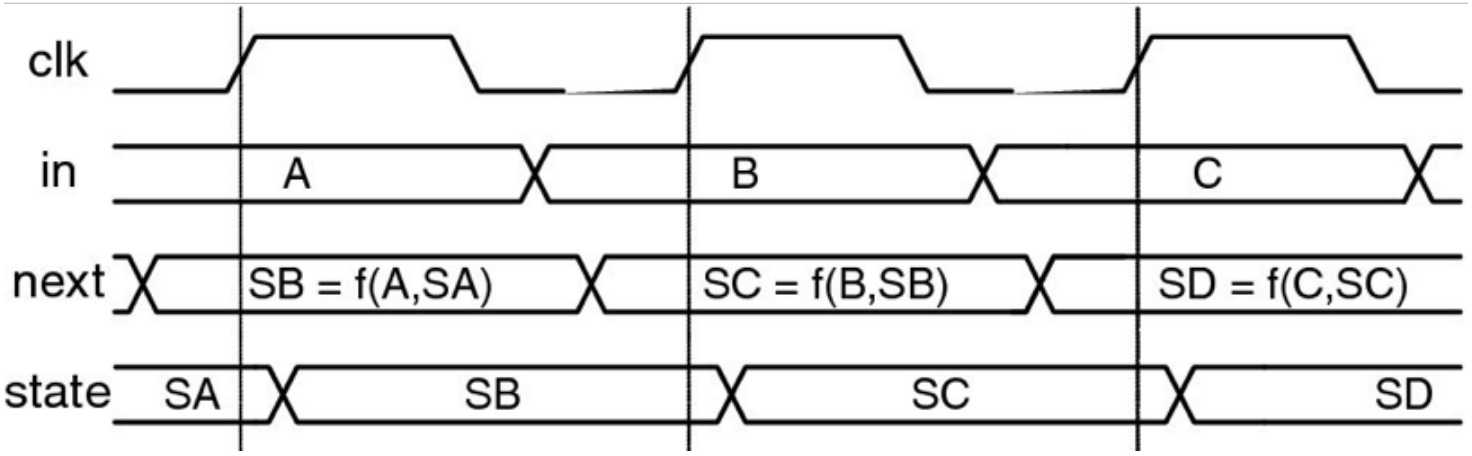
- Characteristic table: define next state in terms of inputs and current states.
- Characteristic equation: define next state as a Boolean function in terms of inputs and current state.
- Excitation table: define input variables as a function of current state and next state.

Flip-flop name	Flip-flop symbol	Characteristic table	Characteristic equation	Excitation table
SR		S R | Q(next) 0 0 | Q 0 1 | 0 1 0 | 1 1 1 | NA	$Q(next) = S + R'Q$ $SR = 0$	Q Q(next) | S R 0 0 | 0 X 0 1 | 1 0 1 0 | 0 1 1 1 | X 0
JK		J K | Q(next) 0 0 | Q 0 1 | 0 1 0 | 1 1 1 | Q'	$Q(next) = JQ' + K'Q$	Q Q(next) | J K 0 0 | 0 X 0 1 | 1 X 1 0 | X 1 1 1 | X 0
D		D | Q(next) 0 | 0 1 | 1	$Q(next) = D$	Q Q(next) | D 0 0 | 0 0 1 | 1 1 0 | 0 1 1 | 1
T		T | Q(next) 0 | Q 1 | Q'	$Q(next) = TQ' + T'Q$	Q Q(next) | T 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Analysis of Sequential Circuits

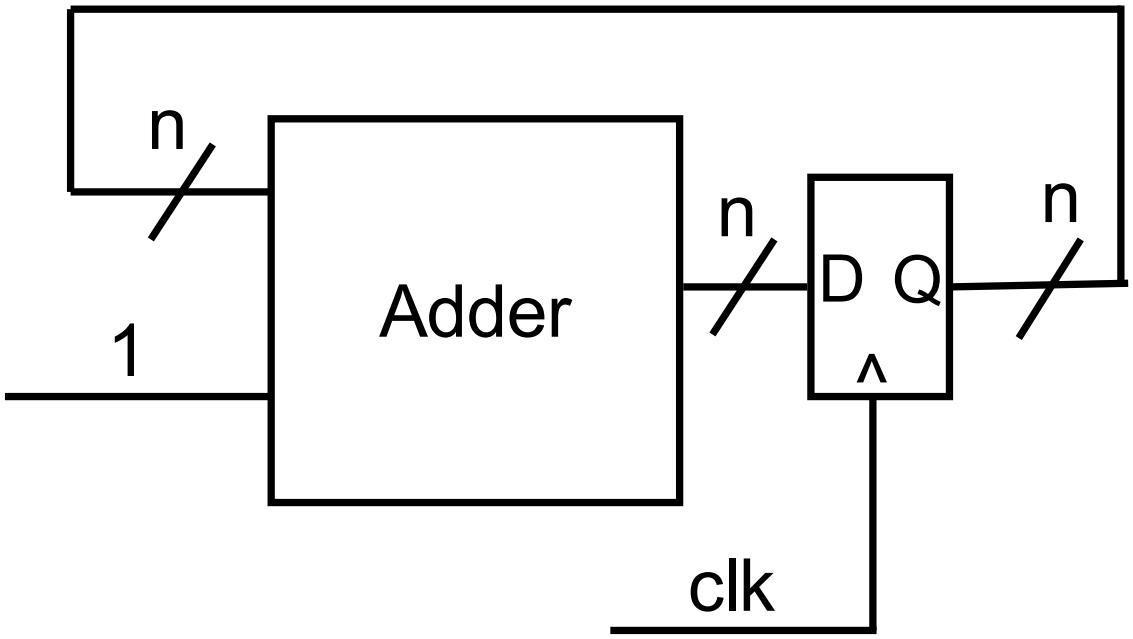
Synchronous Sequential Circuits





Example: Incrementer

- Increment a count on every clock tick.



Sequential Circuit Analysis

- Obtain a table or diagram for the sequence of inputs, outputs, and internal states
- General model
  - Current State at time (t) is stored in an array of flip-flops
  - Next State at time (t+1) is a Boolean function of State and inputs
  - Outputs at time (t) are a Boolean function of state (t) and (sometimes) inputs (t)
- Analysis procedure
  - Derive excitation (input) equation
  - Derive next-state and output equations
  - Generate next-state and output tables
  - Generate state diagram
  - Develop timing diagram
  - Simulate logic circuit

State Table Characteristics

- State table – a multiple variable table with the sections:
  - Present State
  - Input
  - Next-state
  - Output
- From the viewpoint of a truth table:
  - The inputs are Input, Present State
  - The outputs are Output, Next State

State Diagram

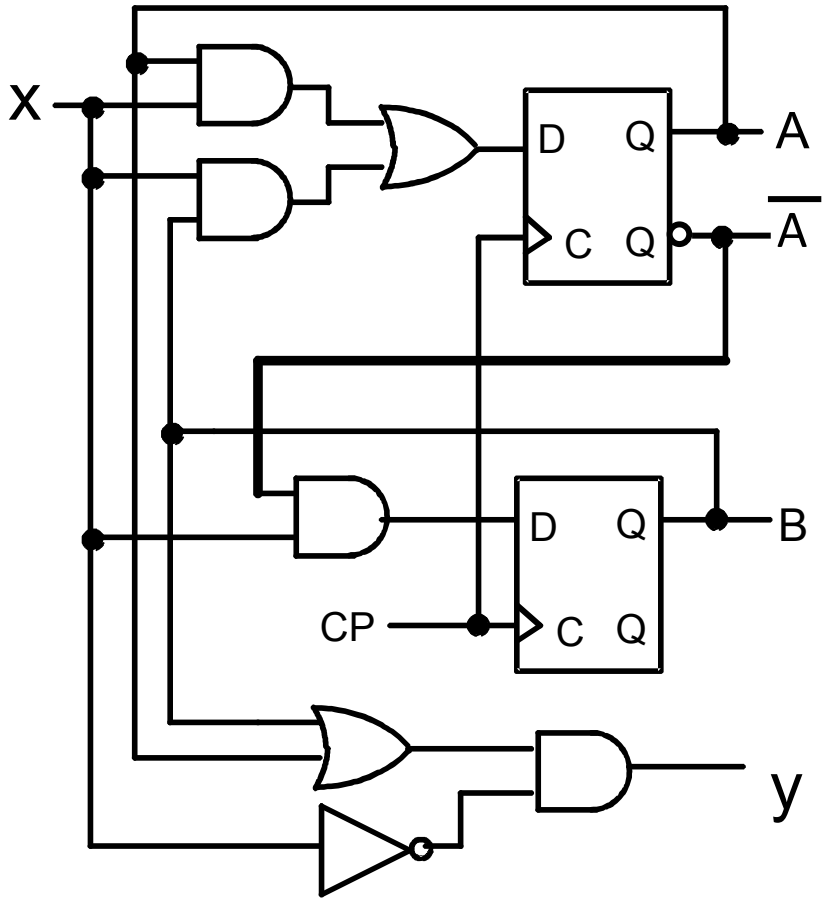
- The sequential circuit can be represented in graphical form as a state diagram with the following components:
  - Circle: with the name of the state in it.
  - Directed line: from the present state to the next state.

1. Outputs depend on present state and inputs

1. Outputs depend only on present state

Sequential Circuit Example I (1/2)

- Input: x(t)
- Output: y(t)
- States: A(t), B(t)
- First, find the inputs of the FFs



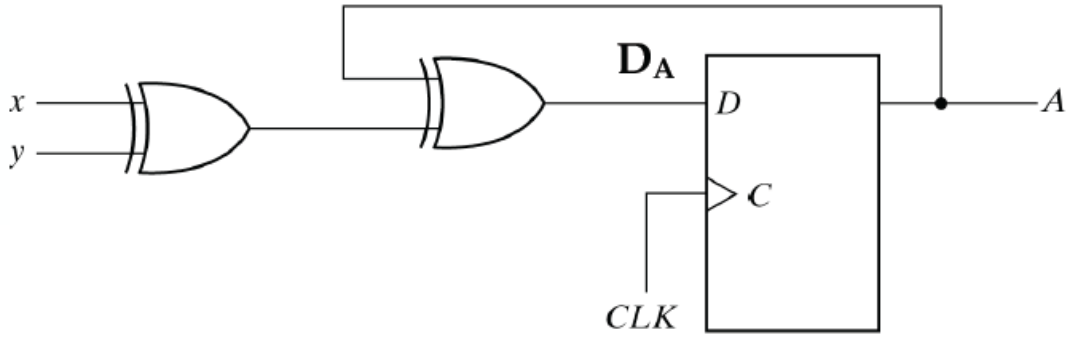
- Output equation:

- Next state equation:

Sequential Circuit Example I (2/2)

Present State		Input	Next State		Output
A(t)	B(t)	x(t)	A(t+1)	B(t+1)	y(t)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Sequential Circuit Example II (1/2)



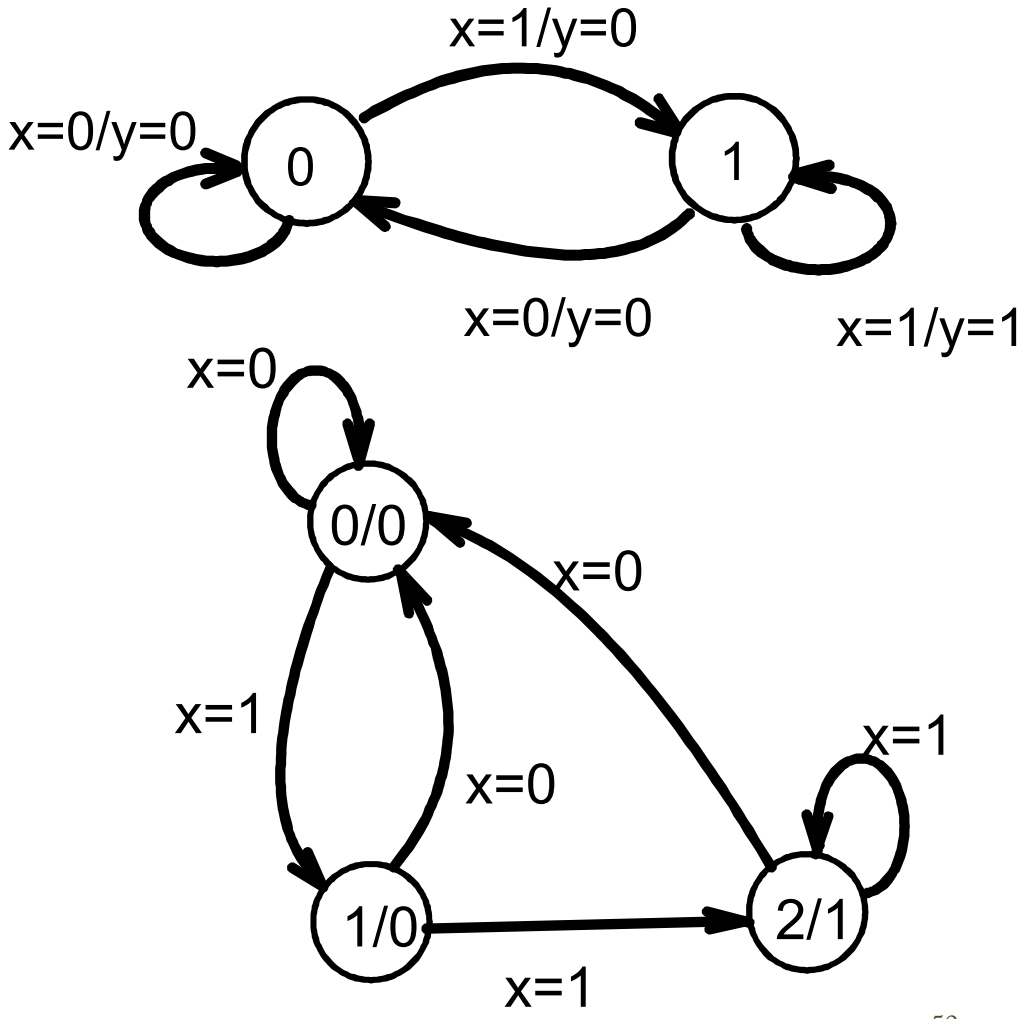
Sequential Circuit Example II (2/2)

A	x	y	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Finite State Machine (FSM)

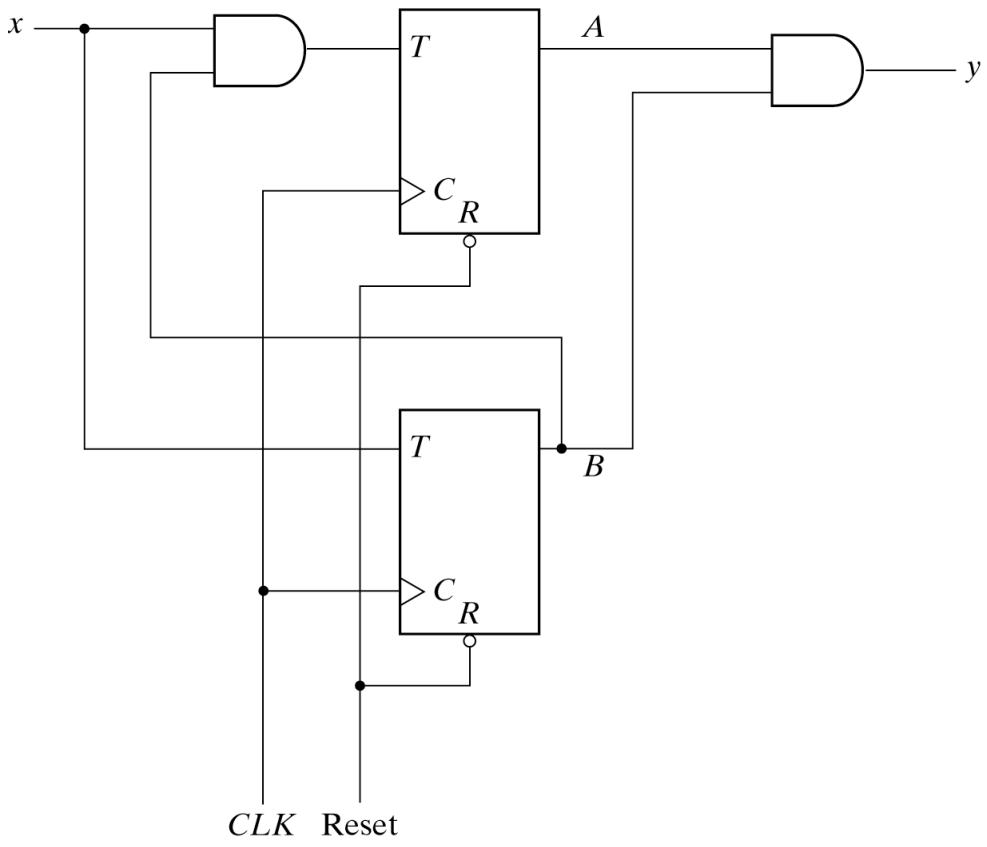
- A synchronous sequential circuit can be modeled by a finite state machine (FSM).
- Two models for FSM:
  - Moore: outputs are function of only the present state
  - Mealy: outputs are function of both the present state and inputs

Moore and Mealy Example Diagrams

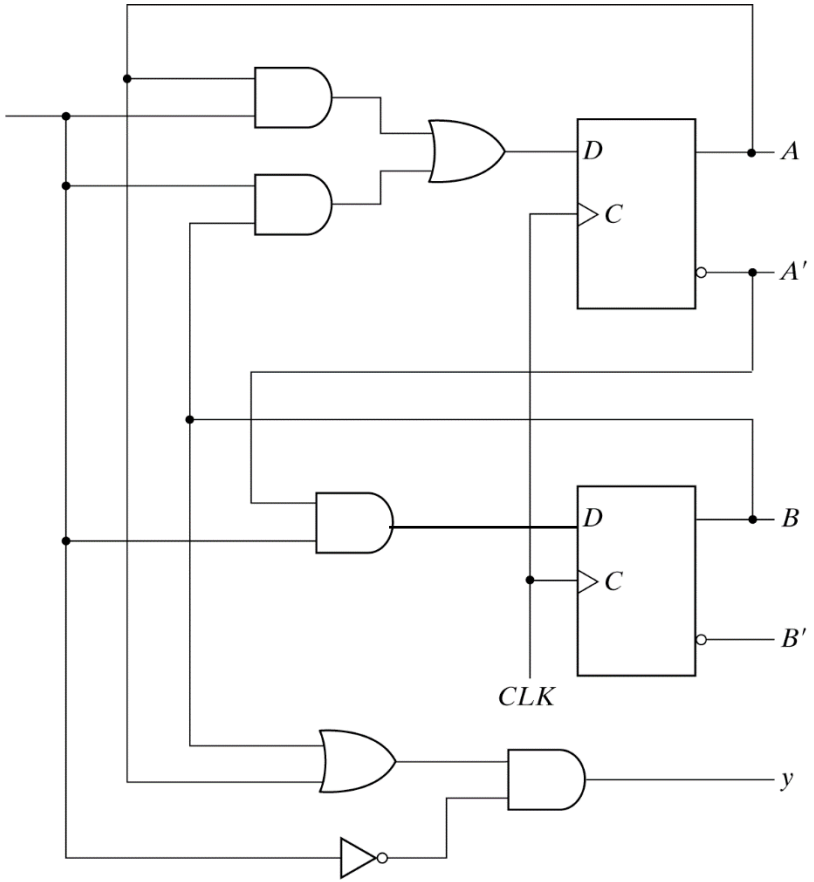


Moore and Mealy Examples

Moore



Mealy



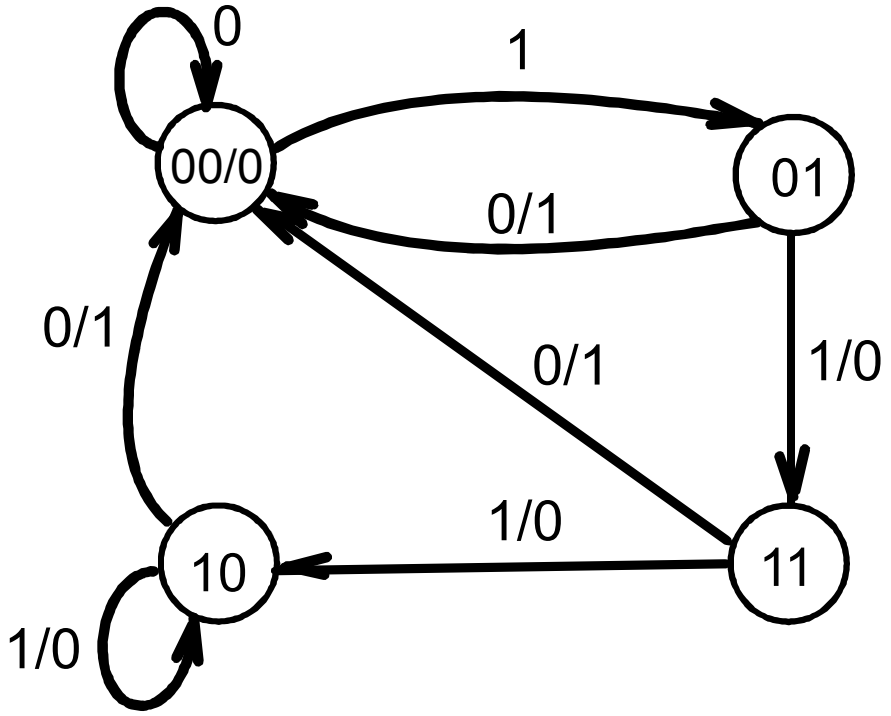
Moore and Mealy Example Tables

Present State	Next State x=0	Next State x=1	Output
0	0	1	0
1	0	2	0
2	0	2	1

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
0	0	1	0	0
1	0	1	0	1

Mixed Moore and Mealy Outputs

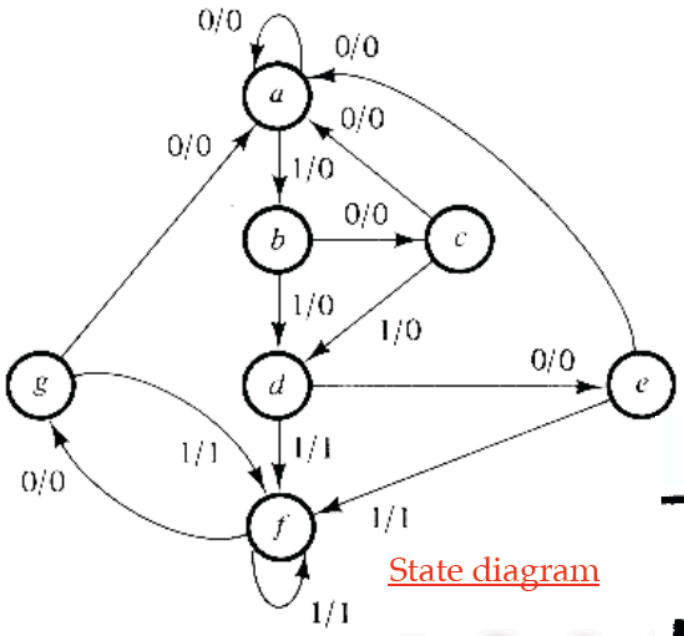
- In real designs, some outputs may be Moore type and other outputs may be Mealy type.



State Minimization

- State reduction
  - Reductions on the number of flip-flops (states) and the number of gates.
  - For an FSM with m states, we need $\lceil \log_2 m \rceil$ FFs.
- Steps
  - Find rows in the state table that have identical next state and output entries. They are equivalent state. One of them can be removed.
  - Update the state table reflecting the change. Continue until there is no equivalent state.

State Minimization Example (1/2)



State diagram

State table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

State Minimization Example (2/2)

State Assignment

- Each of the *m* states must be assigned a unique binary code.
- Minimum number of bits required is *n* such that

- There are useful state assignments that use more than the minimum number of bits.
- Different state assignments result in different circuits for the intended FSM.
- There is no easy state-assignment procedure that guarantees a minimal-cost or minimum-delay combinational circuits.
  - Exploration of all possibilities are impossible
    - Minimum-bit change
    - Prioritized adjacency
    - One-hot encoding

State Assignment Example (1/2)

- Counting Order Assignment: A = 0 0, B = 0 1, C = 1 0, D = 1 1
- The resulting coded state table:

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
0 0	0 0	0 1	0	0
0 1	0 0	1 0	0	0
1 0	1 1	1 0	0	0
1 1	0 0	0 1	0	1

- Gray Code Assignment: A = 0 0, B = 0 1, C = 1 1, D = 1 0

State Assignment Example (2/2)

- One-hot assignment: for m states, use m bits to form the codes that contain only one “1” in each code.

Present State	Next State x = 0	Next State x = 1	Output x = 0	Output x = 1
0 0	0 0	0 1	0	0
0 1	0 0	1 0	0	0
1 0	1 1	1 0	0	0
1 1	0 0	0 1	0	1

Choice of Memory Elements

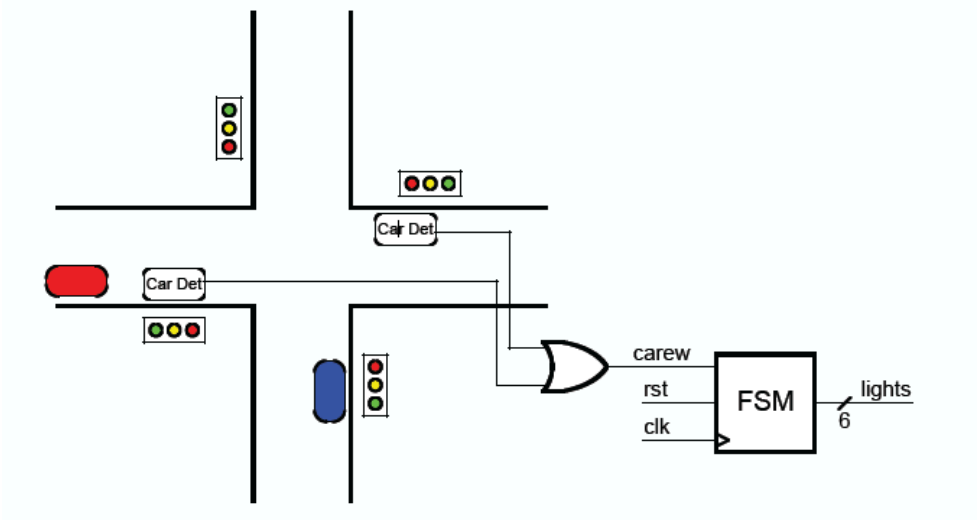
- Given the state table, we need to find the FF input conditions that cause the required transition.
  - Excitation table can be used.
  - SRFFs are used when different signals set/reset FFs.
  - DFFs are good for applications requiring data transfer.
  - TFFs are good for applications involving complementation.
  - Many digital circuits are constructed entirely with JKFFs because of their versatility.

The Design Procedure of Clocked Sequential Circuits

1. - describe the design.
2. - obtain a state diagram and state table.
3. State minimization.
4. State assignment.
5. Input and output equations derivation.
6. Choose memory elements.
7. Map the circuit to the memory and gates (logic diagram).
8. Simulation
9. Verification

Design Example: Traffic Light Counter (1/8)

- Spec
  - Reset to green in north south direction.
  - If light is green or yellow in one direction, it must be red in the other side.
  - A light must be yellow between changing from green to red.
  - If there is a car waiting at east west (carew=1), make the light green in east west and return to green in north south.



Traffic Light Counter FSM (2/8)

- Four states:
  - gns: green north south (red east west)
  - yns: yellow north south (red east west)
  - gew: green east west (red north south)
  - yew: yellow east west (red north south)
- Input
  - Carew: indicate if there is a car waiting at east west
  - Reset: return to initial state (need not go through yellow)
- Output
  - 100 001: NS green, EW red; 001 010 NS red, EW yellow

Traffic Light Counter FSM (3/8)

Traffic Light Counter FSM (4/8)

Traffic Light Counter FSM (5/8)

- State assignment

5

Binary Code

State	Encoding
GNS	00
YNS	01
GEW	10
YEW	11

Gray Code

State	Encoding
GNS	00
YNS	01
GEW	11
YEW	10

State Assignment with Gray Code

current state		carew	next state		next state		output					
$s_1(t)$	$s_0(t)$	c	$s_1(t+1)$	$s_0(t+1)$	$ns_1(t)$	$ns_0(t)$	lgns	lyns	lrns	lgew	lyew	lrew
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	1	0	0	0	0	1
0	1	0	1	1	1	1	0	1	0	0	0	1
0	1	1	1	1	1	1	0	1	0	0	0	1
1	1	0	1	0	1	0	0	0	1	1	0	0
1	1	1	1	0	1	0	0	0	1	1	0	0
1	0	0	0	0	0	0	0	0	1	0	1	0
1	0	1	0	0	0	0	0	0	1	0	1	0

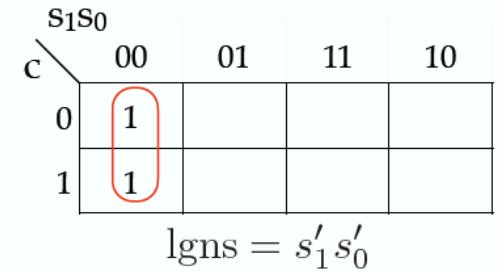
Traffic Light Counter FSM (6/8)

- Next state equations (use DFFs)

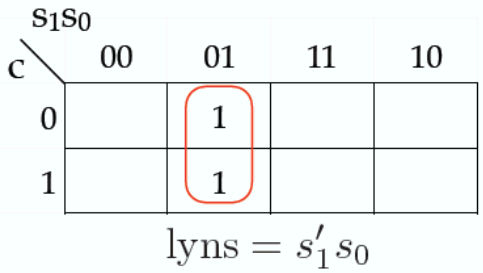
current state		carew	next state		excitation		output					
$s_1(t)$	$s_0(t)$	c	$s_1(t+1)$	$s_0(t+1)$	$ns_1(t)$	$ns_0(t)$	lgns	lyns	lrns	lgew	lyew	lrew
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	1	0	0	0	0	1
0	1	0	1	1	1	1	0	1	0	0	0	1
0	1	1	1	1	1	1	0	1	0	0	0	1
1	1	0	1	0	1	0	0	0	1	1	0	0
1	1	1	1	0	1	0	0	0	1	1	0	0
1	0	0	0	0	0	0	0	0	1	0	1	0
1	0	1	0	0	0	0	0	0	1	0	1	0

Traffic Light Counter FSM (7/8)

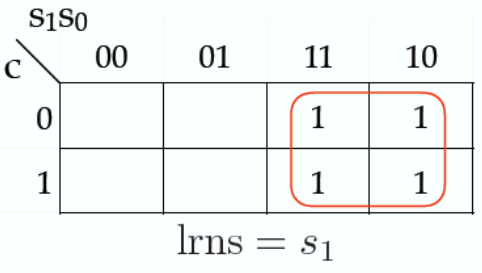
current state		carew	next state		excitation		output					
$s_1(t)$	$s_0(t)$	c	$s_1(t+1)$	$s_0(t+1)$	$ns_1(t)$	$ns_0(t)$	lgns	lyns	lrns	lgew	lyew	lrew
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	1	0	0	0	0	1
0	1	0	1	1	1	1	0	1	0	0	0	1
0	1	1	1	1	1	1	0	1	0	0	0	1
1	1	0	1	0	1	0	0	0	1	1	0	0
1	1	1	1	0	1	0	0	0	1	1	0	0
1	0	0	0	0	0	0	0	0	1	0	1	0
1	0	1	0	0	0	0	0	0	1	0	1	0



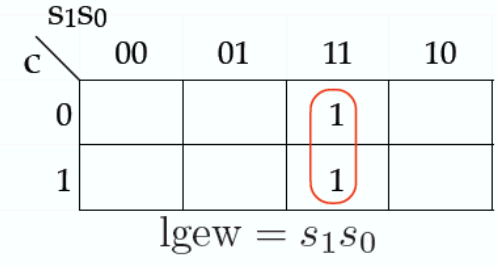
$\text{lgns} = s_1' s_0'$



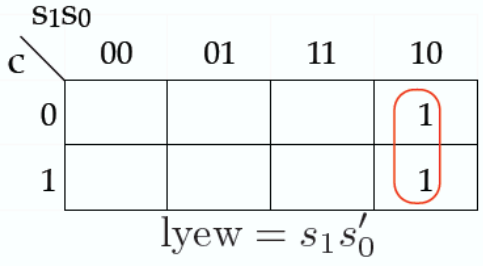
$\text{lyns} = s_1' s_0$



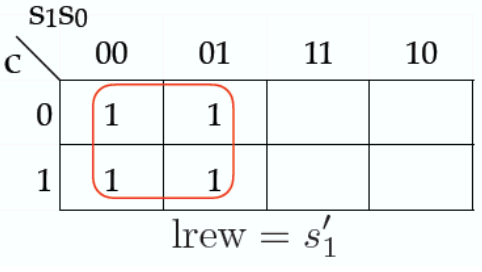
$\text{lrns} = s_1$



$\text{lgew} = s_1 s_0$



$\text{lyew} = s_1 s_0'$



$\text{lrew} = s_1'$

Traffic Light Counter FSM (8/8)

Excitation equation (Input equation)

$$\mathrm{ns}_1 = s_0$$

$$\mathrm{ns}_0 = cs_1' + s_1's_0 = (c + s_0)s_1'$$

Output equation

$$\mathrm{lgns} = s_1's_0'$$

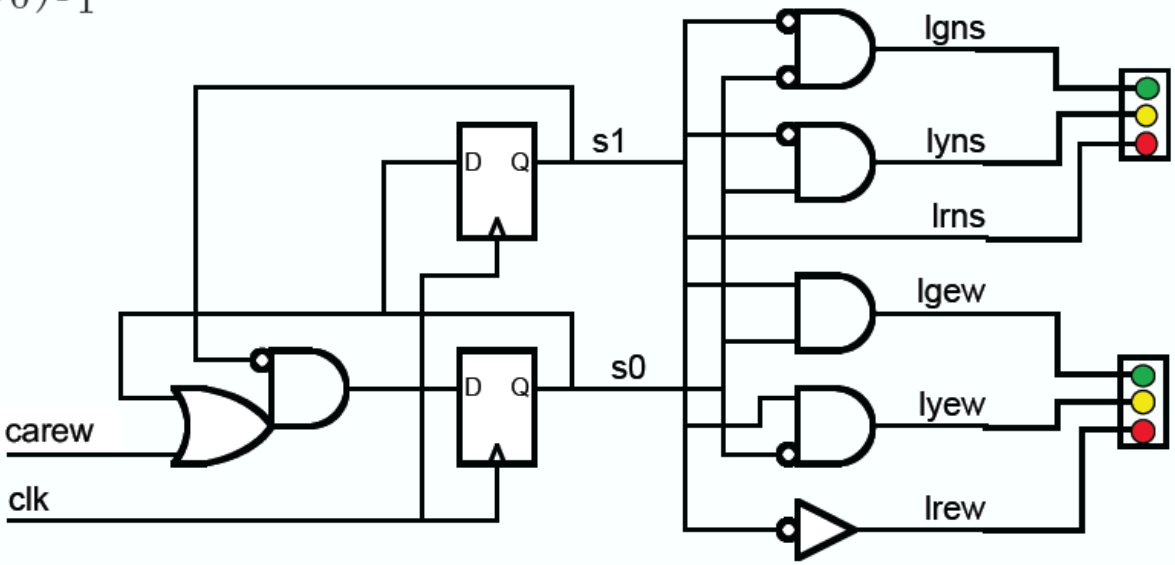
$$\mathrm{lyns} = s_1's_0$$

$$\mathrm{lrns} = s_1$$

$$\mathrm{lgew} = s_1s_0$$

$$\mathrm{lyew} = s_1s_0'$$

$$\mathrm{lrew} = s_1'$$



Sequence Recognizer Example (1/3)

- Example: A circuit that recognizes the sequence 1101

Sequence Recognizer Example (2/3)

- Starting in the initial state (arbitrarily named "A"):
  - Add a state that recognizes the first ”1”.
  - State "A" is the initial state, and state "B" is the state which represents the fact that the "first" one in the input subsequence has occurred. The output symbol "0" means that the full recognized sequence has not yet occurred.
  - Output 1 on the arc from D means the sequence has been recognized.

Sequence Recognizer Example (3/3)

- The state have the following abstract meanings:
  - A: no proper sub-sequence of the sequence has occurred
  - B: the sub-sequence 1 has occurred
  - C: the sub-sequence 11 has occurred
  - D: the sub-sequence 110 has occurred
  - the 1/1 on the arc from D to B means that the last 1 has occurred and thus, the sequence is recognized

Formulation: Find State Table

Present State	Next State x=0 x=1	Output x=0 x=1
A		
B		
C		
D		

Example: Moore Model for Sequence 1101

- For the Moore Model, outputs are associated with states.
- We need to add a state "E" with output value 1 for the final 1 in the recognized input sequence.
- The Moore model for a sequence recognizer usually has *more states* than the Mealy model.

Example: Moore Model (2/3)

- We mark outputs on states for Moore model.
- Arcs now show only state transitions.
- The new state, E produces the same behavior in the future as state B, but it gives a different output at the present time. Thus these states do represent a *different abstraction* of the input history.

Example: Moore Model (3/3)

Present State	Next State x=0	Next State x=1	Output y
A			
B			
C			
D			
E			

Sequence Recognition Design Example (1/2)

- Use Mealy model
- Use counting order state assignment

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
00	00	01	0	0
01	00	10	0	0
10	11	10	0	0
11	00	01	0	1

Sequence Recognition Design Example (2/2)

Verilog Examples

Always@

- always@: describe events that happen under certain conditions. It is always followed by a set of parentheses, begin, some code, and end.

```
always @(sensitivity list)  begin
   ….
end
```

- always@(posedge clock): always at the positive edge of the clock. Used in sequential circuits.
- always@*: elements change its value as soon as one or more of its input change.
  - * sets the sensitivity list to any values that can have an impact on a value.


```
always @(A or B)  begin
        C = A & B;
end
```

```
always @*  begin
        C = A & B;
end
```

DFFs

```
module DFF(clk, in, out) ;

  parameter n = 1;  // width
  input clk ;
  input [n-1:0] in ;
  output [n-1:0] out ;
  reg [n-1:0] out ;

  always @(posedge clk)
    out = in ;

endmodule
```

Blocking and Nonblocking

- Blocking: evaluation and assignment are immediate

  x = a | b; //evaluate a | b and assign it to x

  y = a & b; //evaluate a & b and assign it to y

  – Recommend to use in always block for combinational circuits

- Nonblocking: evaluate first. Assignments are deferred until all evaluations finish.

  x <= a | b; //evaluate a | b

  y <= a & b; //evaluate a & b

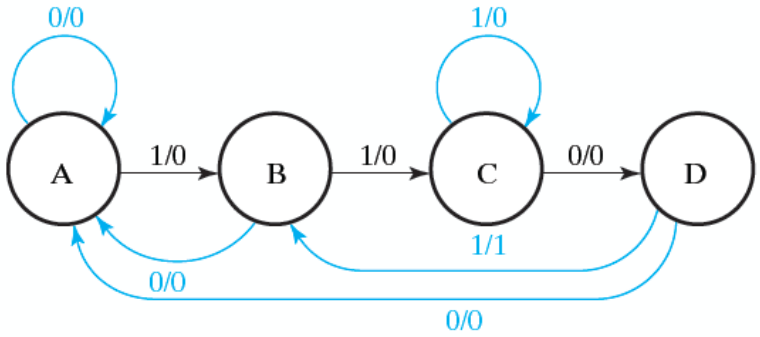
  //assign x and y with their new values

  – Recommend to use in always block for sequential circuits

Style Guide: FSMs in Verilog

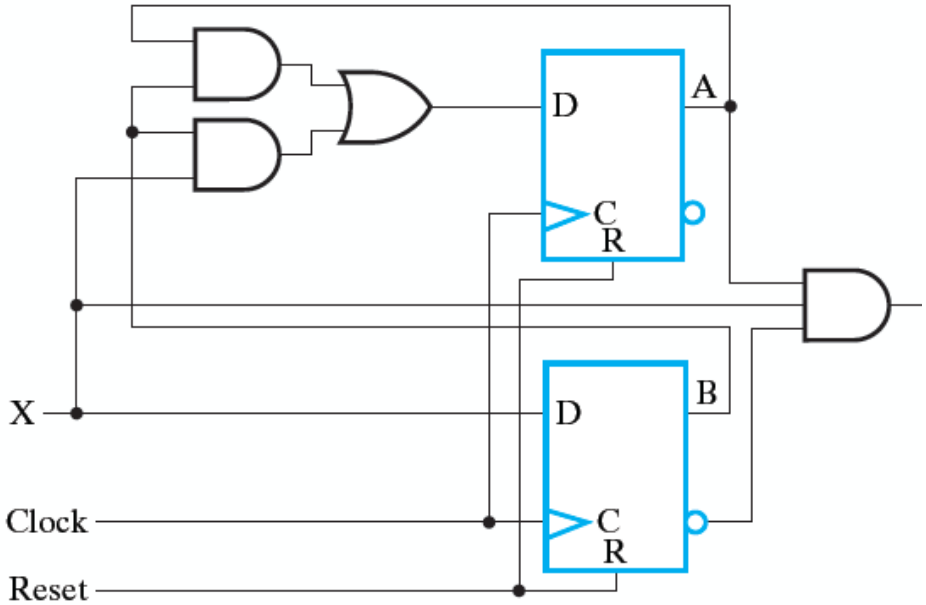
- Feedback: through explicitly instantiate DFF module.
- Next state and output are combinational.
  - Case or assign.
  - No inferred latches.
- Make sure you can reset FSM.
- Use defines for state encoding (and width).

Verilog: 1101 Sequence Recognizer

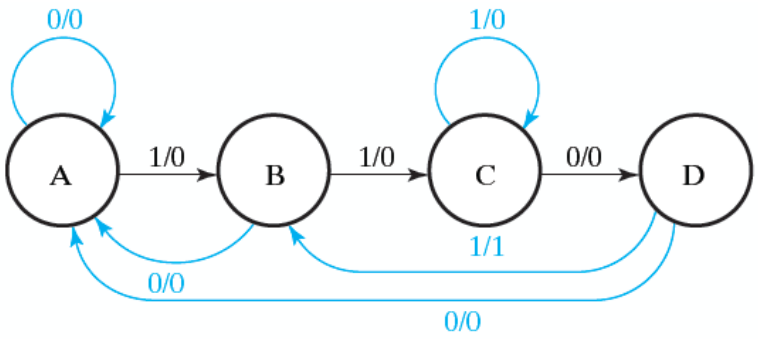


Present State	Next State		Output Z	
	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

a(t+1) = ab + bx
b(t+1) = x
y = ab'x



Present state		Input	Next state		Output
a	b	x	a	b	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	0



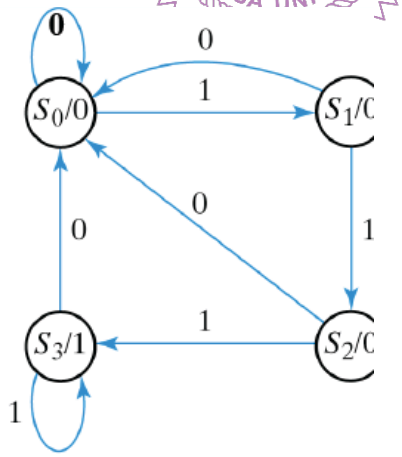

```
//state assignment
`define SWIDTH 2
`define A 2'b00
`define B 2'b01
`define C 2'b11
`define D 2'b10

//define output codes
`define DET 1'b1
`define N_DET 1'b0
```

```
module seq(clk, rst_n, in, out)

input clk;
input rst_n;
input in;
output reg out;
reg [`SWIDTH-1:0] state;
reg [`SWIDTH-1:0] next_state;

always@(posedge clk or negedge rst_n)
 if (~rst_n)
   state <= `A;
 else
   state <= next_state;

always@* begin
 case (state)
    `A: {next_state, out}={(in? `B:`A), (in? `N_DET:`N_DET)};
    `B: {next_state, out}={(in? `C:`A), (in? `N_DET:`N_DET)};
    `C: {next_state, out}={(in? `C:`D), (in? `N_DET:`N_DET)};
    `D: {next_state, out}={(in? `B:`A), (in? `DET:`N_DET)};
    default: {next_state, out}={`SWIDTH+1{1'bx}};
 endcase
end
endmodule
```

Verilog: FSM Example

```
// Gray code state assignment
`define SWIDTH 2
`define S0 2'b00
`define S1 2'b01
`define S2 2'b11
`define S3 2'b10

// define output codes
`define DETECTED 1'b1
`define NOT_DETECTED 1'b0

module Ex2(clk, rst, in, out);
input clk;
input rst;
input in;
output reg out;

reg [`SWIDTH-1:0] state;
reg [`SWIDTH-1:0] next_state;
```

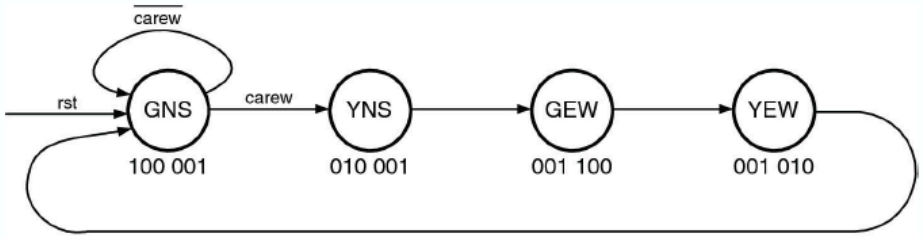
```
// low active asynchronous reset
always @(posedge clk or negedge rst_n)
 if (~rst_n)
   state <= `S0;
 else
   state <= next_state;

// state and output eqs - combinational logic
always @*
 case (state)
   `S0:  {next_state,out} = {(in?`S1:`S0), `NOT_DETECTED};
   `S1:  {next_state,out} = {(in?`S2:`S0), `NOT_DETECTED};
   `S2:  {next_state,out} = {(in?`S3:`S0), `NOT_DETECTED};
   `S3:  {next_state,out} = {(in?`S3:`S0), `DETECTED};
   default: {next_state,out} = {`SWIDTH+1{1'bx}};
 endcase

endmodule
```



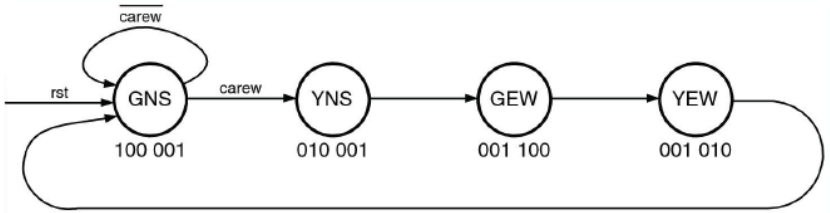
Courtesy of Prof. HP Ma

Verilog: Traffic Light Example

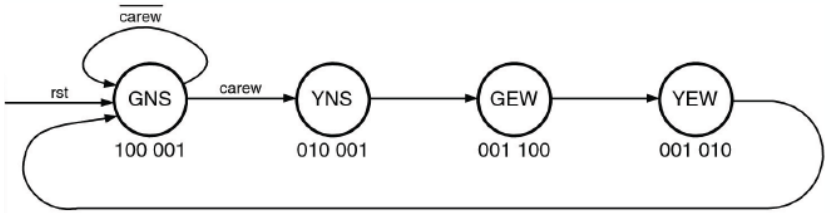
- Use define statement to define constant.
- Define cannot be changed during simulation.




```
// Gray code state assignment
`define SWIDTH 2
`define GNS 2'b00
`define YNS 2'b01
`define GEW 2'b11
`define YEW 2'b10

// define output codes
`define LGNS 6'b100001
`define LYNS 6'b010001
`define LGEW 6'b001100
`define LYEW 6'b001010
```




```
module Traffic_Light(clk, rst, carew, lights) ;
  input clk ;
  input rst ;
  input carew ;
  output reg [5:0] lights ;
  wire [`SWIDTH-1:0] state, next ;
  reg [`SWIDTH-1:0] next1 ;

  DFF #(`SWIDTH) state_reg(clk, next, state) ;

  // next state and output equations - this is combinational logic
  always @* begin
    case(state)
      `GNS: {next1, lights} = {(carew ? `YNS : `GNS), `GNSL} ;
      `YNS: {next1, lights} = {`GEW, `YNSL} ;
      `GEW: {next1, lights} = {`YEW, `GEWL} ;
      `YEW: {next1, lights} = {`GNS, `YEWL} ;
    endcase
   end

  // add reset
  assign next = rst ? `GNS : next1 ;

endmodule
```




```
module Traffic_Light(clk, rst, carew, lights) ;
  input clk ;
  input rst ;
  input carew ;
  output reg [5:0] lights ;
  reg [`SWIDTH-1:0] state ;
  reg [`SWIDTH-1:0] next1 ;

  always@(posedge clk or posedge rst)
   if(rst)
    state <= `GNS;
   else
    state <= state1;

  // next state and output equations - this is combinational logic
  always @* begin
    case(state)
      `GNS: {next1, lights} = {(carew ? `YNS : `GNS), `GNSL} ;
      `YNS: {next1, lights} = {`GEW, `YNSL} ;
      `GEW: {next1, lights} = {`YEW, `GEWL} ;
      `YEW: {next1, lights} = {`GNS, `YEWL} ;
    endcase
   end
endmodule
```