VietNam National University
University of Engineering and Technology

Modelling Sequential Logic Circuit and FSM with VHDL

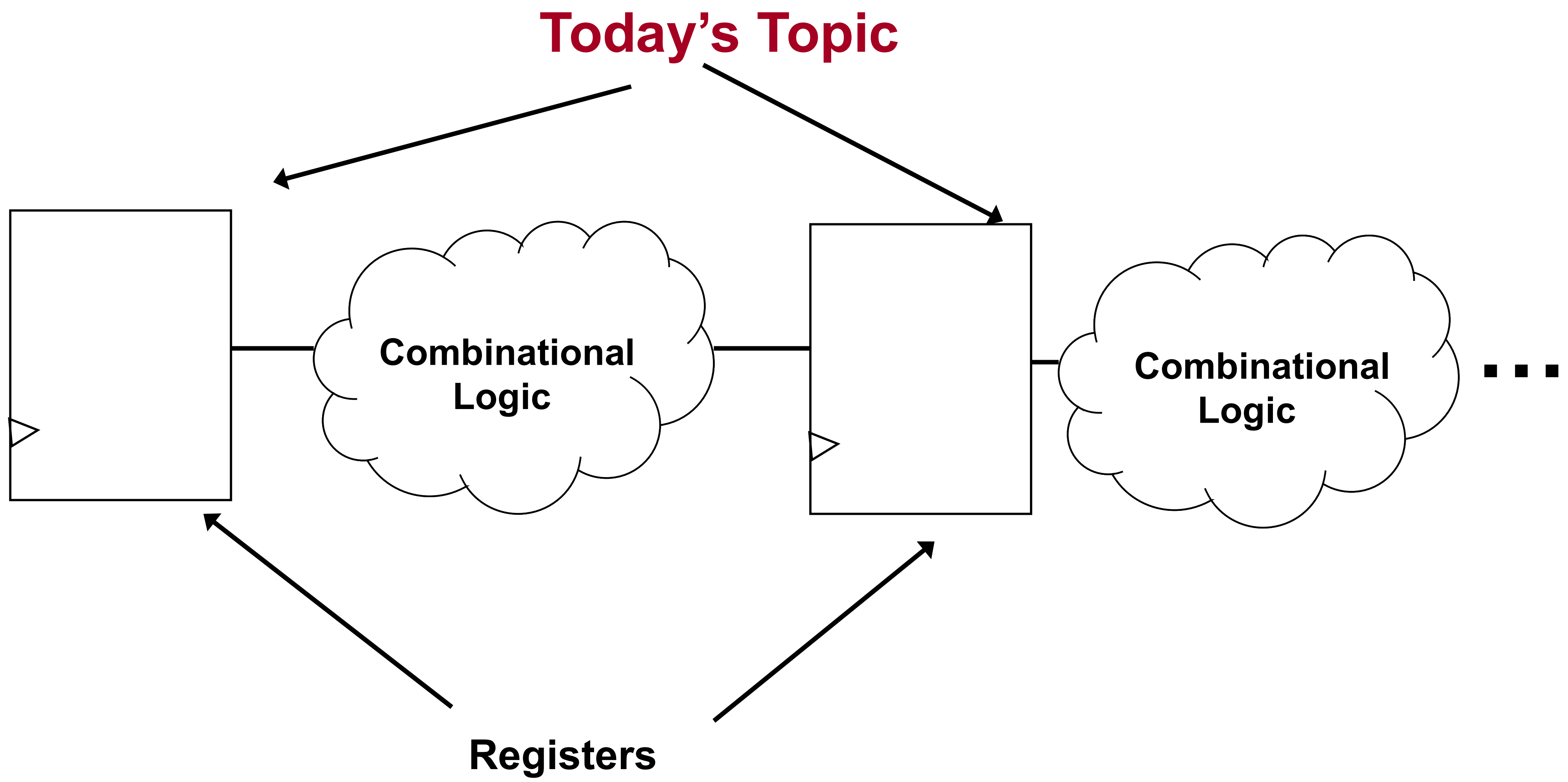
TS. Nguyễn Kiêm Hùng
Email: kiemhung@vnu.edu.vn

Laboratory for Smart Integrated Systems

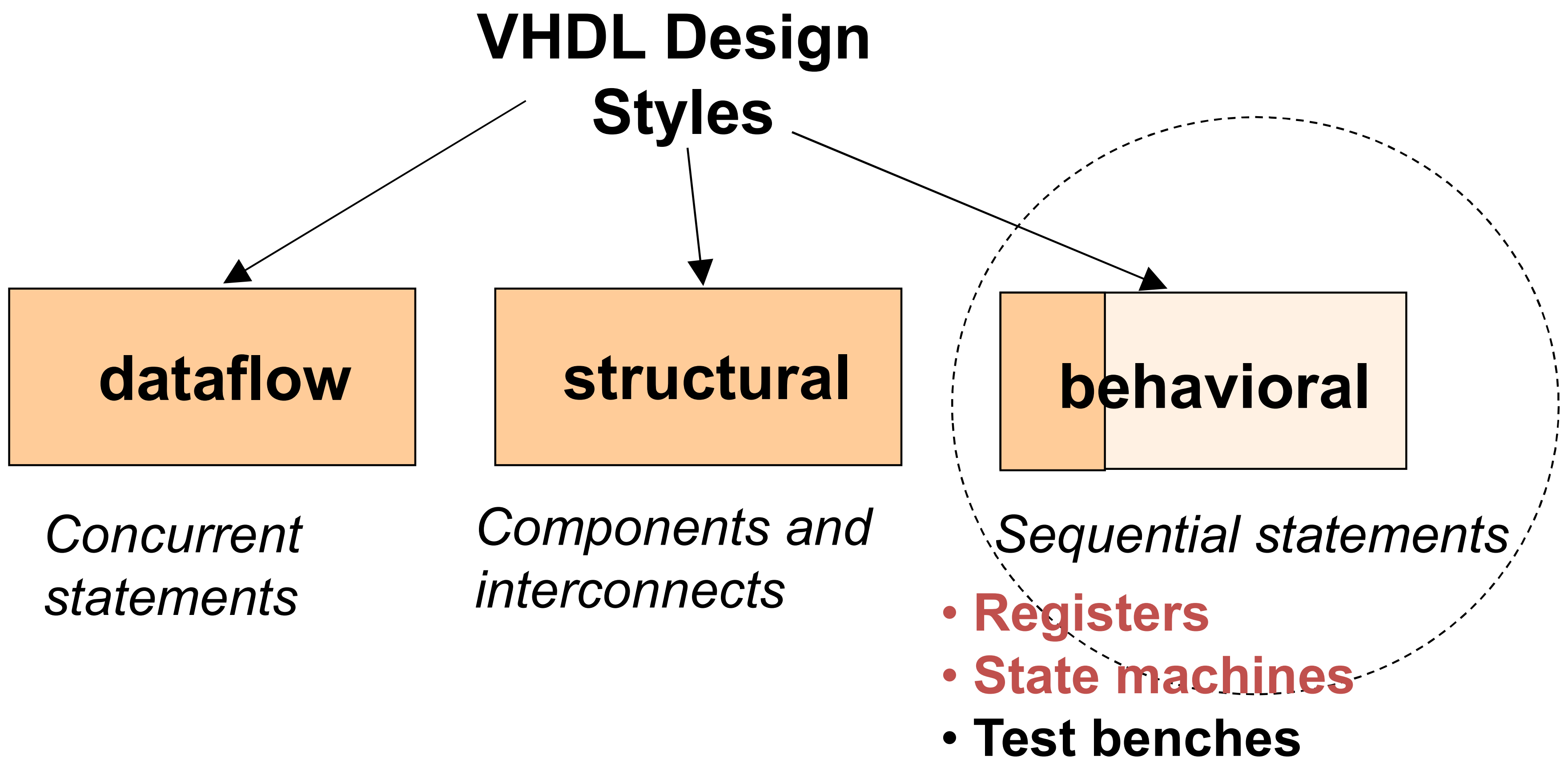
Objectives

- In this lecture you will be introduced to:
  - **VHDL constructs used to implement storage elements**
  - **Design of Flip-flops, Registers, Shift registers, Counters by using VHDL**
  - **Design of sequential circuits, finite state machines**

Register Transfer Level (RTL) Description



VHDL Design Styles



Data Objects

Data Objects

ARCHITECTURE acrhitecture_name **OF** entity_name **IS**
-- Signal and constant declaration is here
→ **SIGNAL** signal_name: data_type [:= initial value];
→ **CONSTANT** constant_name: data_type [:= initial value]; --global
BEGIN
PROCESS ()
-- Variable and Constant declaration is here
CONSTANT constant_name: data_type [:= initial value]; --local
→ **VARIABLE** constant_name: data_type [:= initial value];

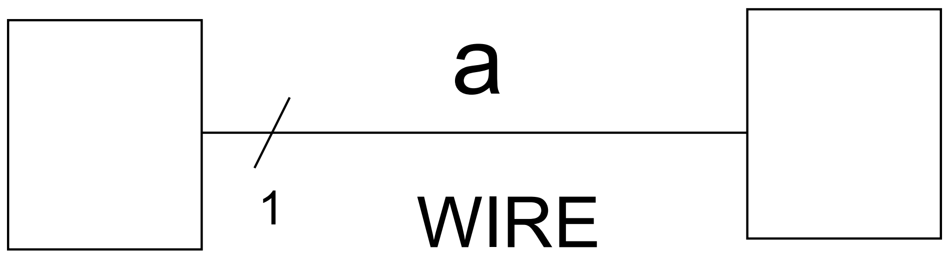
BEGIN
END PROCESS ;
END acrhitecture_name ;

Signals (here)

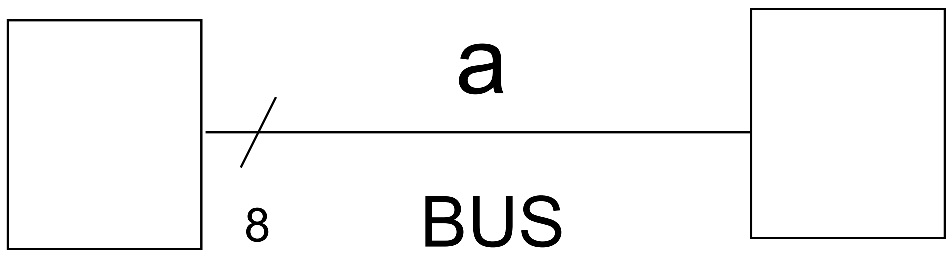
SIGNAL represent the logic signals or wires in a circuit, and are a function of the signal assignment statements (⇐)

```
CONSTANT N: Integer;
SIGNAL  a : STD_LOGIC_VECTOR (N-1 DOWNTO 0);
```

N = 1



N = 8



Constants

- ✓ a data object whose value **CANNOT** be changed.
- ✓ **DO NOT** represent a wire in a circuit
- ✓ the purpose is to improve the readability of code

```
constant RISE_FALL_TME: time := 2 ns;
constant DELAY1: time := 4 ns;
constant RISE_TIME, FALL_TIME: time:= 1 ns;
constant DATA_BUS: integer:= 16;
```

Variables

✓ DO NOT necessarily represent a wire in a circuit

✓ used to hold the results of computations and for the index variables in loops

✓A variable can be updated using a variable assignment statement:

Variable_name := expression;

```
variable CNTR_BIT: bit :=0;
variable VAR1: boolean :=FALSE;
variable SUM: integer range 0 to 256 :=16;
variable STS_BIT: bit_vector (7 downto 0);
```

Generate Statements: QUIZ

```
ARCHITECTURE adder_4bit_structure OF adder_4bit IS
   signal c: std_logic_vector (4 downto 0);
   component FULLADDER
           port (x, y, cin:  in std_logic;
                s, cout:  out std_logic);
   end component;
BEGIN
     C(0) <= cin;
     G1: FOR i IN 0 TO 3 GENERATE
       FA: FULLADDER
        port map (x(i), y(i), c(i), sum(i), c(i+1));
     END GENERATE ;

     Cout <= c(4);
      V <= c(3) xor c(4);
END adder_4bit_structure ;
```

Where is the variable I stored?

Signal Vs. Variable

Signal Assignment Statement: Signal_name **<=** expression;	**Variable Assignment Statement:** Variable_name **:=** expression;
A signal changes AFTER A DELAY after the assignment expression is evaluated.	**A variable changes instantaneously when the variable assignment is executed.**

Signal Vs. Variable

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY signal_exp IS
 PORT (TRIGGER : IN STD_LOGIC;
                 Result : OUT integer);
END signal_exp;
ARCHITECTURE bev OF signal_exp IS
  --SIGNAL TRIGGER, RESULT: integer := 0;
  SIGNAL s1: integer :=1;
  SIGNAL s2: integer :=2;
  SIGNAL s3: integer :=3;
BEGIN
  PROCESS
   BEGIN
    WAIT on TRIGGER;
     s1 <= s2;
     s2 <= s1 + s3;
     s3 <= s2;
     RESULT <= s1 + s2 + s3;
     END PROCESS;
END bev;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY variable_exp IS
 PORT (TRIGGER : IN STD_LOGIC;
                 Result : OUT integer);
END variable_exp;
ARCHITECTURE bev OF variable_exp IS
  --SIGNAL TRIGGER, RESULT: integer := 0;
BEGIN
  PROCESS
    VARIABLE v1: integer :=1;
    VARIABLE v2: integer :=2;
    VARIABLE v3: integer :=3;
    BEGIN
     WAIT on Trigger;
     v1 := v2;
     v2 := v1 + v3;
     v3 := v2;
    Result <= v1 + v2 + v3;
   END PROCESS;
END bev;
```

Signal Vs. Variable

```
ARCHITECTURE Behavior OF dec2to4 IS
BEGIN
   PROCESS ( w, En )
   BEGIN
    IF En = '1' THEN
      CASE w IS
            WHEN "00" =>
                        y <= "1000" ;
            WHEN "01" =>
                        y <= "0100" ;
            WHEN "10" =>
                        y <= "0010" ;
            WHEN OTHERS =>
                        y <= "0001" ;
      END CASE ;
    ELSE
            y <="0000" ;
    END IF ;
  END PROCESS ;
END Behavior ;
END bev;
```

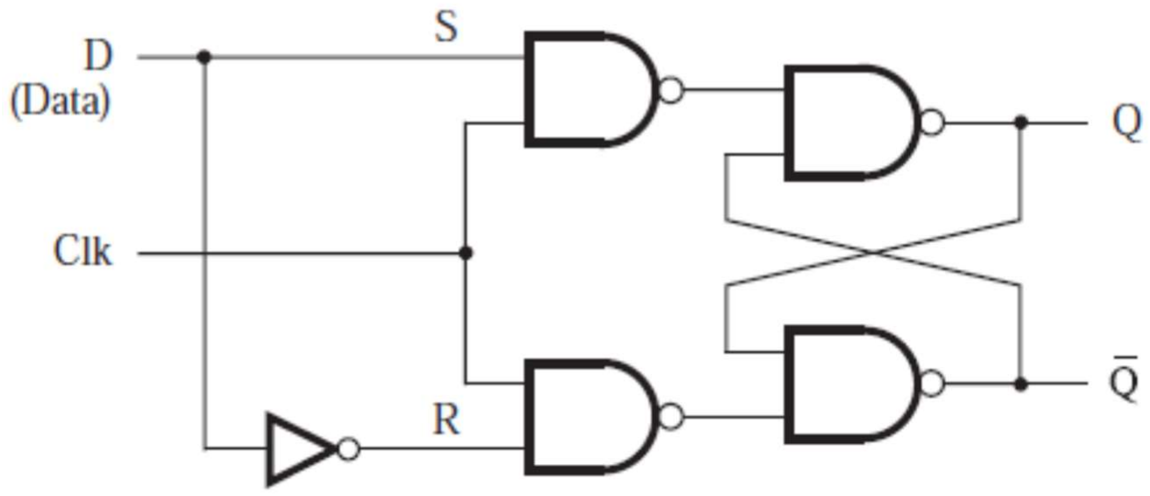
```
ARCHITECTURE Behavior OF dec2to4 IS
 SIGNAL t : STD_LOGIC_VECTOR(2 downto 0);
BEGIN
   PROCESS ( w, En )
   BEGIN
      t <= En&w;
      CASE t IS
            WHEN "100" =>
                        y <= "1000" ;
            WHEN "101" =>
                        y <= "0100" ;
            WHEN "110" =>
                        y <= "0010" ;
            WHEN "111" =>
                        y <= "0001" ;
            WHEN OTHERS =>
                        y <= "0000" ;
      END CASE ;
;
  END PROCESS ;
END Behavior ;
END bev;
```

Describing Sequential Logic

Sequential Assignment Statements

- ✓ The ordering of the *sequential assignment statements* may affect the meaning of the code.
- ✓ The sequential assignment statements be placed inside another type of statement, called a *process* statement
- ✓ VHDL provides two types of sequential assignment statements:
  - ➢ **IF-THEN-ELSE** statement
  - ➢ **CASE** statement
- ✓ and more:
  - ➢ Attribute of data object
  - ➢ WAIT statements
    - **WAIT ON** signal changes
    - **WAIT UNTIL** an expression is true
    - **WAIT FOR** a specific amount of time
  - ➢ Etc.

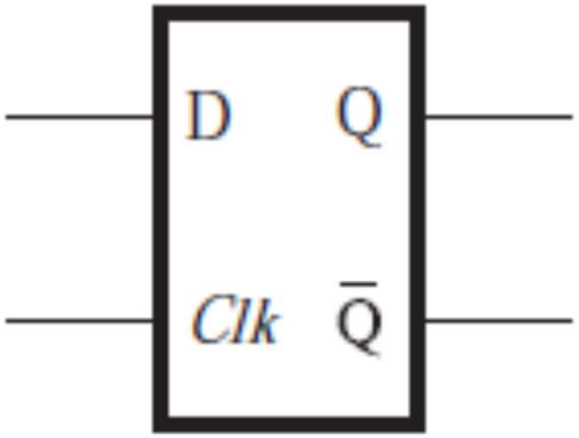
Gated D Latch



(a) Circuit

Clk	D	Q(t+1)
0	-	Q(t)
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol

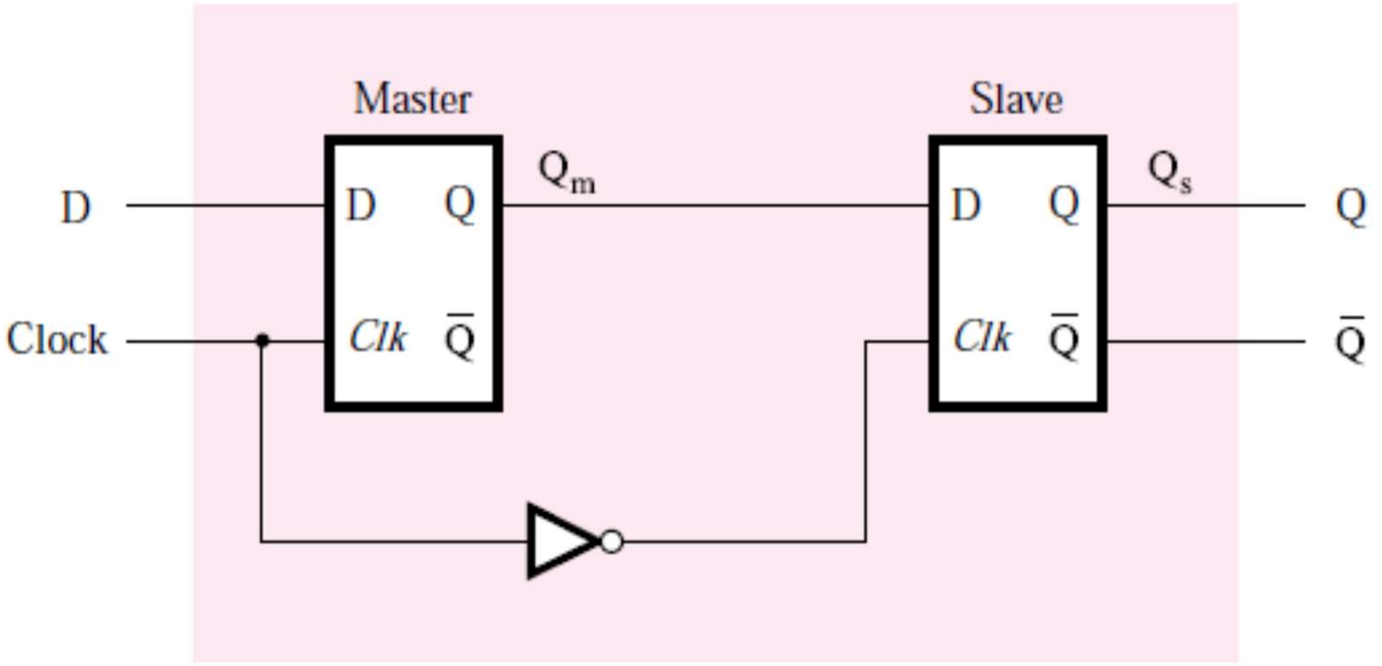
Gated D Latch

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY latch IS
  PORT ( D, Clk : IN STD LOGIC ;
              Q : OUT STD LOGIC) ;
  END latch ;
ARCHITECTURE Behavior OF latch IS
BEGIN
  PROCESS (         )
    BEGIN
      IF          THEN



      END IF ;
   END PROCESS ;
END Behavior ;
```

Gated D Latch

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY latch IS
  PORT ( D, Clk : IN STD LOGIC ;
              Q : OUT STD LOGIC) ;
  END latch ;
ARCHITECTURE Behavior OF latch IS
BEGIN
  PROCESS ( D, Clk )
    BEGIN
       IF Clk = ‘1’ THEN
            Q <= D ;
      END IF ;
   END PROCESS ;
END Behavior ;
```

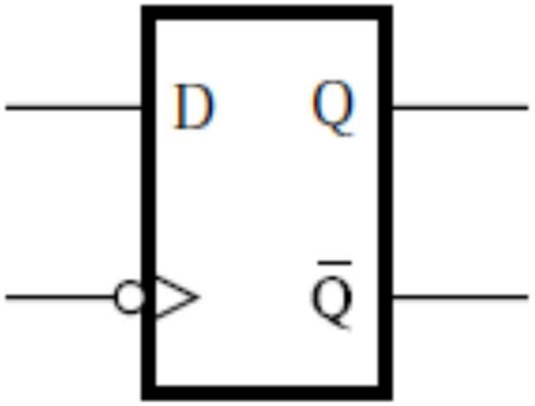
D FLIP-FLOP



(a) Circuit

Clk	D	Q(t+1)
↓	0	0
↓	1	1
Others	-	Q(t)

(b) Characteristic table



(c) Graphical symbol

D FLIP-FLOP

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
    PORT ( D, Clock : IN STD LOGIC ;
                Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
  BEGIN
    PROCESS ( Clock )
      BEGIN
        IF (Clock'EVENT AND Clock =  '0') THEN
                    Q <= D ;
        END IF ;
    END PROCESS ;
 END Behavior ;
```

contains only the clock signal

' EVENT attribute refers to any change in the *Clock* signal

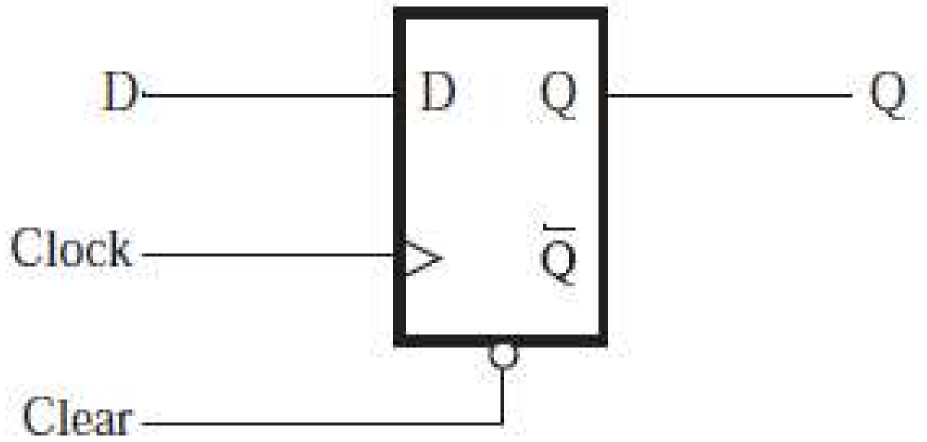
D FLIP-FLOP: Alternative Code

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
    PORT ( D, Clock : IN STD LOGIC ;
                Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
  BEGIN
    PROCESS
      BEGIN
          WAIT UNTIL Clock’EVENT AND Clock =  ’0’;
                  Q <= D ;

    END PROCESS ;
 END Behavior ;
```

D FLIP-FLOP with ASYNCHRONOUS CLEAR

Clear	Clock	D	Q(t+1)
0	-	-	0
1	↑	0	0
1	↑	1	1
1	Others	-	Q(t)

(a) Characteristic table



D flip-flop with asynchronous clear

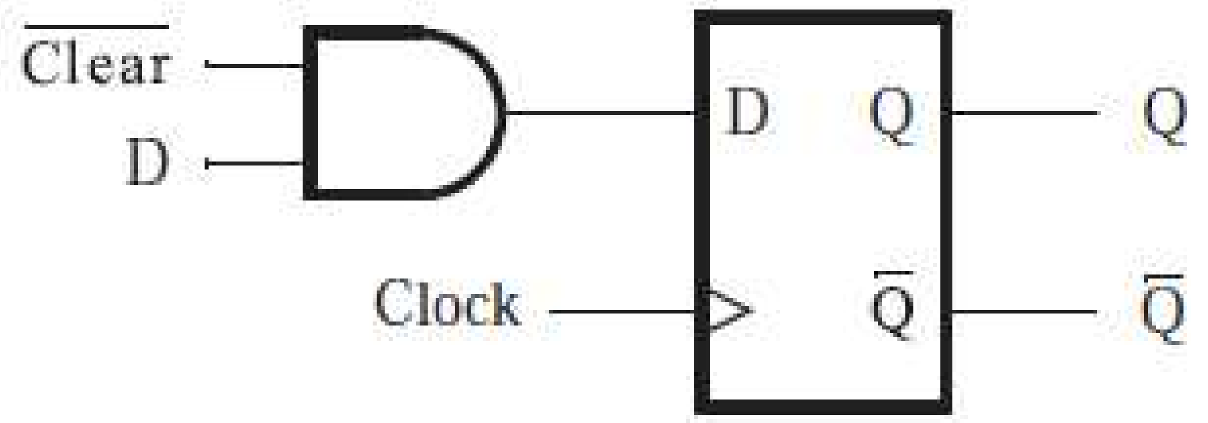
D FLIP-FLOP with ASYNCHRONOUS CLEAR

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
PORT ( D, nClear, Clock : IN STD LOGIC ;
                Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS (          )
    BEGIN






  END PROCESS ;
END Behavior ;
```

D FLIP-FLOP with ASYNCHRONOUS CLEAR

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
PORT ( D, nClear, Clock : IN STD LOGIC ;
                Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS ( nClear, Clock )
    BEGIN
      IF nClear = ’0’ THEN
                Q <=’0’ ;
      ELSIF (Clock’EVENT AND Clock = ’1’) THEN
                Q <= D ;
      END IF ;
  END PROCESS ;
END Behavior ;
```

D FLIP-FLOP with SYNCHRONOUS CLEAR



A synchronous clear

Clear	Clock	D	Q(t+1)
0	↑	-	0
1	↑	0	0
1	↑	1	1
Others	Others	-	Q(t)

(b) Characteristic table

D FLIP-FLOP with SYNCHRONOUS CLEAR

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
PORT ( D, nClear, Clock : IN STD LOGIC ;
                Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
    BEGIN







  END PROCESS ;
END Behavior ;
```

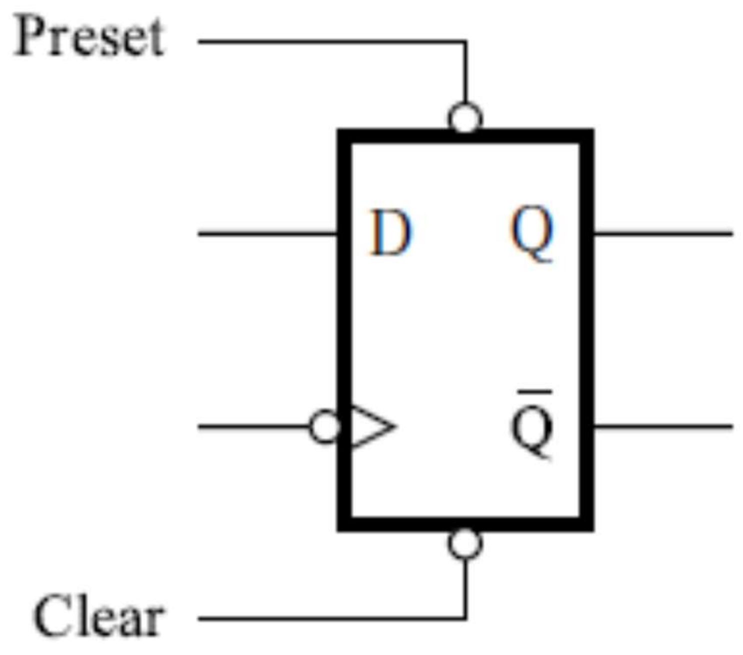
D FLIP-FLOP with SYNCHRONOUS CLEAR

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
PORT ( D, nClear, Clock : IN STD LOGIC ;
                Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
    BEGIN
      WAIT UNTIL Clock'EVENT AND Clock =  '1';
      IF nClear = '0' THEN
                Q <='0' ;
      ELSE
                Q <= D ;
      END IF ;
  END PROCESS ;
END Behavior ;
```

QUIZ

Write VHDL code for a Master-slave D flip-flop with *Clear* and *Preset?*

Clear	Preset	Clock	D	Q(t+1)
0	-	↑	-	0
1	0	↑	-	1
1	1	↑	0	0
1	1	↑	1	1
Others	Others	Others	-	Q(t)



Master-slave D flip-flop with synchronous *Clear* and synchronous *Preset*

D flip-flop with synchronous *Clear* and *Preset*

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
PORT ( D, nClear, nPreset, Clock : IN STD LOGIC ;
         Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
 PROCESS ( )
  BEGIN




END PROCESS ;
END Behavior ;
```

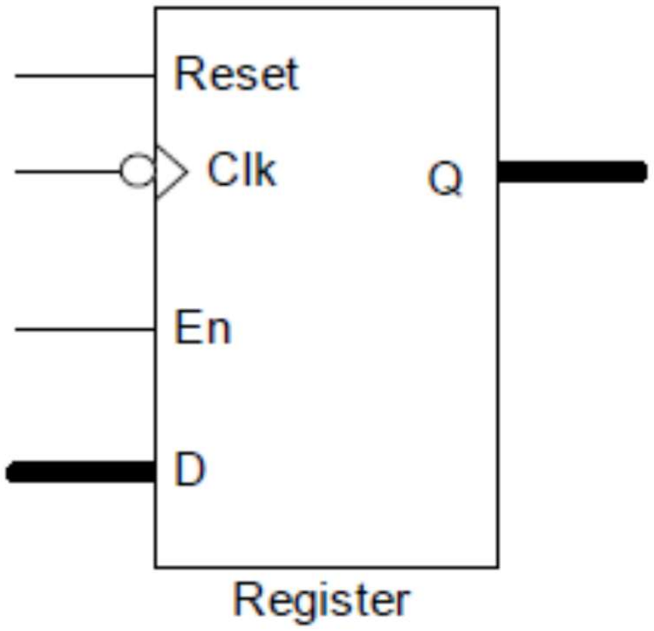
D flip-flop with synchronous *Clear* and *Preset*

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY flipflop IS
PORT ( D, nClear, nPreset, Clock : IN STD LOGIC ;
                 Q : OUT STD LOGIC) ;
END flipflop ;
ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
    BEGIN
      WAIT UNTIL Clock’EVENT AND Clock =  ’1’;
      IF nClear = ’0’ THEN
                 Q <=’0’ ;
      ELSIF nPreset = ‘0’ THEN
                 Q <=’1’ ;
     ELSE
                 Q <= D ;
      END IF ;
  END PROCESS ;
END Behavior ;
```

n-bit Register with Asynchronous Clear

Register **is a set of *n* flip-flops is used to store *n* bits of information, such as an *n*-bit number.**

Reset	**En**	**Clk**	**D**	**Q(t+1)**
1	-	-	-	0
0	1	↓	0	0
0	1	↓	1	1
Others	Others	Others	-	Q(t)



Standard Register Symbol

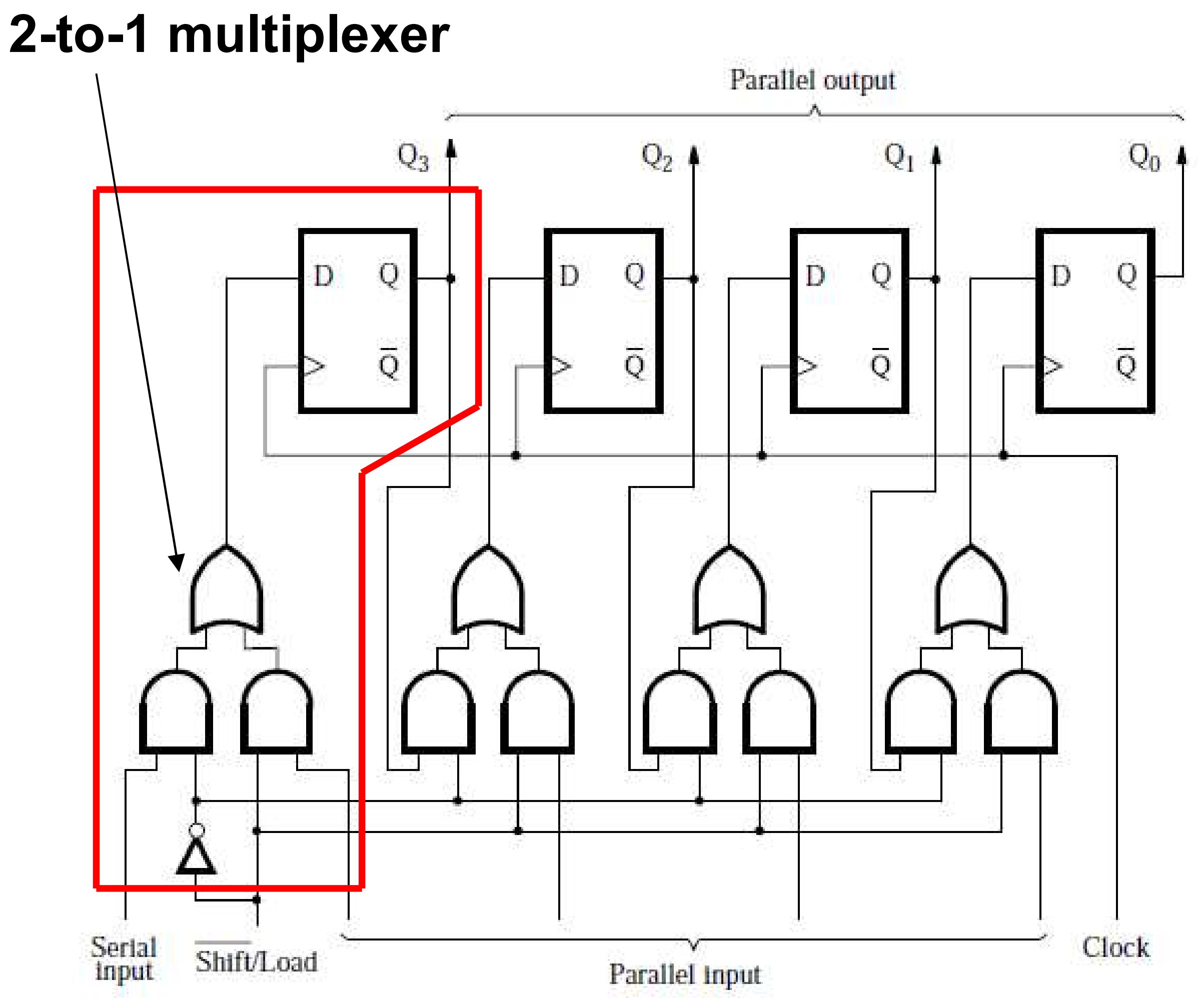
n-bit Register with Asynchronous Clear

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY regn IS
  GENERIC ( N : INTEGER : = 16 ) ;
  PORT ( D : IN STD_LOGIC_VECTOR(N −1 DOWNTO 0) ;
                Reset, Clk, En : IN STD LOGIC ;
                Q : OUT STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
END regn ;
ARCHITECTURE Behavior OF regn IS
  BEGIN
    PROCESS (         )
      BEGIN







   END PROCESS ;
END Behavior ;
```

n-bit Register with Asynchronous Clear

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY regn IS
  GENERIC ( N : INTEGER : = 16 ) ;
  PORT ( D : IN STD_LOGIC_VECTOR(N −1 DOWNTO 0) ;
                Reset, Clock, En : IN STD LOGIC ;
                Q : OUT STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
END regn ;
ARCHITECTURE Behavior OF regn IS
  BEGIN
    PROCESS ( Reset, Clk )
      BEGIN
        IF Reset = ’1’ THEN
                Q <=  (OTHERS => ’0’) ;
       ELSIF Clk’EVENT AND Clk = ’0’ THEN
             IF (En = ‘1’) THEN
                Q <= D ;
            END IF;
      END IF ;
   END PROCESS ;
END Behavior ;
```

FOUR-BIT PARALLEL-ACCESS SHIFT REGISTER



The control signal *Shift*/*Load* is used to select the mode of operation:

- If *Shift*/*Load* = 0, then the circuit operates as a shift register.
- If *Shift*/*Load* = 1, then the parallel input data are loaded into the register.

FOUR-BIT SHIFT REGISTER (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY muxdff IS
 PORT ( D0, D1, Sel, Clock : IN STD_LOGIC ;
              Q : OUT STD_LOGIC ) ;
 END muxdff ;
ARCHITECTURE Behavior OF muxdff IS
 BEGIN
  PROCESS
   BEGIN







  END PROCESS ;
END Behavior ;
```

FOUR-BIT SHIFT REGISTER (1)

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY muxdff IS
  PORT ( D0, D1, Sel, Clock : IN STD LOGIC ;
                Q : OUT STD LOGIC ) ;
  END muxdff ;
ARCHITECTURE Behavior OF muxdff IS
  BEGIN
    PROCESS
     BEGIN
       WAIT UNTIL Clock'EVENT AND Clock = '1' ;
       IF Sel =  '0' THEN
                Q <= D0 ;
       ELSE
                Q <= D1 ;
       END IF ;
  END PROCESS ;
END Behavior ;
```

FOUR-BIT SHIFT REGISTER (2)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS
PORT ( D : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
                L, w, Clock : IN STD LOGIC ;
                Q : BUFFER STD LOGIC VECTOR(3 DOWNTO 0) ) ;
END shift4 ;
ARCHITECTURE Structure OF shift4 IS
  COMPONENT muxdff
     PORT ( D0, D1, Sel, Clock : IN STD LOGIC ;
                  Q : OUT STD LOGIC ) ;
   END COMPONENT ;
   BEGIN





        END Structure ;
```

FOUR-BIT SHIFT REGISTER (2)

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY shift4 IS
PORT ( D : IN STD LOGIC VECTOR(3 DOWNTO 0) ;
                L, w, Clock : IN STD LOGIC ;
                Q : BUFFER STD LOGIC VECTOR(3 DOWNTO 0) ) ;
END shift4 ;
ARCHITECTURE Structure OF shift4 IS
  COMPONENT muxdff
     PORT ( D0, D1, Sel, Clock : IN STD LOGIC ;
                  Q : OUT STD LOGIC ) ;
   END COMPONENT ;
   BEGIN
        Stage3: muxdff PORT MAP ( w, D(3), L, Clock, Q(3) ) ;
        Stage2: muxdff PORT MAP ( Q(3), D(2), L, Clock, Q(2) ) ;
        Stage1: muxdff PORT MAP ( Q(2), D(1), L, Clock, Q(1) ) ;
        Stage0: muxdff PORT MAP ( Q(1), D(0), L, Clock, Q(0) ) ;
  END Structure ;
```

FOUR-BIT SHIFT REGISTER: Alternative code

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY shift4 IS
PORT ( D : IN STD LOGIC VECTOR(3 DOWNTO 0) ;
           Clock : IN STD LOGIC ;
           L, w : IN STD LOGIC ;
           Q : BUFFER STD LOGIC VECTOR(3 DOWNTO 0) ) ;
END shift4 ;
ARCHITECTURE Behavior OF shift4 IS
 BEGIN
  PROCESS
   BEGIN
    WAIT UNTIL Clock'EVENT AND Clock  = '1' ;
    IF L = '1' THEN
           Q <= D ;
    ELSE
           Q(0) <= Q(1) ;
           Q(1) <= Q(2);
           Q(2) <= Q(3) ;
           Q(3) <= w ;
    END IF ;
 END PROCESS ;
END Behavior ;
```

FOUR-BIT SHIFT REGISTER: QUIZ

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY shift4 IS
PORT ( D : IN STD LOGIC VECTOR(3 DOWNTO 0) ;
          Clock : IN STD LOGIC ;
          L, w : IN STD LOGIC ;
          Q : BUFFER STD LOGIC VECTOR(3 DOWNTO 0) ) ;
END shift4 ;
ARCHITECTURE Behavior OF shift4 IS
 BEGIN
  PROCESS
   BEGIN
    WAIT UNTIL Clock’EVENT AND Clock = ’1’ ;
    IF L = ’1’ THEN
          Q <= D ;
    ELSE
          Q(3) <= w ;
          Q(2) <= Q(3) ;
          Q(1) <= Q(2);
          Q(0) <= Q(1) ;
     END IF ;
 END PROCESS ;
END Behavior ;
```

QUIZ

Write VHDL code for a n-bit shift register by using **GENERIC** declaration to set the number of bits of the register?

N-BIT SHIFT REGISTER

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY shiftn IS
   GENERIC ( N : INTEGER := 8 ) ;
   PORT (D: IN STD LOGIC VECTOR(N-1 DOWNTO 0) ;
                  Clock : IN STD LOGIC ;
                  L, w : IN STD LOGIC ;
                  Q : BUFFER STD LOGIC VECTOR(N-1 DOWNTO 0) ) ;
END shiftn ;
ARCHITECTURE Behavior OF shiftn IS
  BEGIN
   PROCESS
     BEGIN
       WAIT UNTIL Clock’EVENT AND Clock  ’1’ ;
       IF L = ’1’ THEN
                  Q <= D ;
       ELSE
                  Genbits: FOR i IN 0 TO N-2 LOOP
                                    Q(i) <= Q(i + 1) ;
                                    END LOOP ;
                  Q(N-1) <= w ;
      END IF ;
  END PROCESS ;
END Behavior ;
```

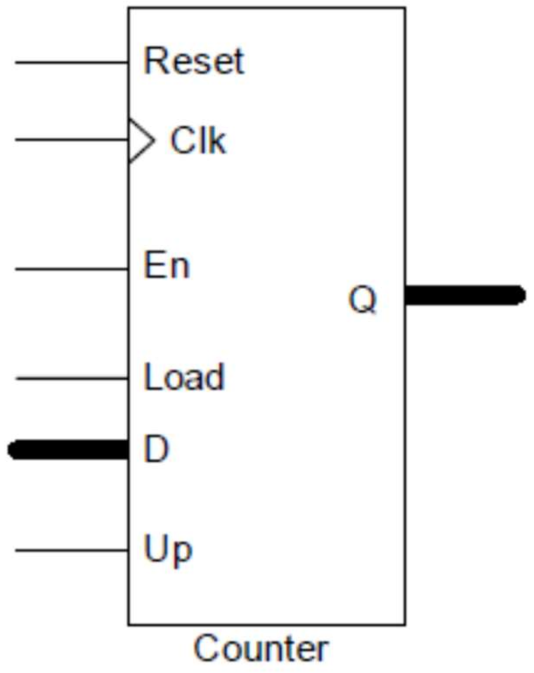
COUNTER

✓ a synchronous register (CLK tick) that increments or decrements its output (Q) at an edge of the clock signal

✓ If the enable signal (En) is low the output remains always unchanged.

✓If the load signal (Load) is high the output is changed to the input (D) value on the clock edge.

✓the up signal (Up) is high the output is increased at the next clock edge



Symbol

FOUR-BIT UP COUNTER

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
USE ieee.std logic unsigned.all ;
ENTITY upcount IS
   PORT ( Clock, nReset, E : IN STD LOGIC ; --E: enable
                  Q : OUT STD LOGIC VECTOR (3 DOWNTO 0)) ;
END upcount ;
ARCHITECTURE Behavior OF upcount IS

BEGIN
  PROCESS (         )
    BEGIN











 END PROCESS ;

END Behavior ;
```

FOUR-BIT UP COUNTER

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
USE ieee.std logic_unsigned.all ;
ENTITY upcount IS
   PORT ( Clock, nReset, E : IN STD LOGIC ;
                 Q : OUT STD LOGIC VECTOR (3 DOWNTO 0)) ;
END upcount ;
ARCHITECTURE Behavior OF upcount IS
   SIGNAL Count : STD LOGIC VECTOR (3 DOWNTO 0) ;
BEGIN
  PROCESS ( Clock, nReset )
    BEGIN
      IF nReset = ‘0’ THEN
                 Count <= “0000”;
       ELSIF (Clock’EVENT AND Clock  = ‘1’) THEN
          IF E = ‘1’ THEN
                 Count <= Count + 1 ;
          ELSE
                 Count <= Count ;
          END IF ;
       END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```

QUIZ

Write VHDL code for a n-bit up counter by using **GENERIC** declaration to set the number of bits of the counter?

N-BIT DOWN COUNTER

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
USE ieee.std logic_unsigned.all ;
ENTITY downcnt IS
  GENERIC ( modulus : INTEGER := 8 ) ;
   PORT ( Clock, L, E : IN STD LOGIC ;
                Q : OUT INTEGER RANGE 0 TO (modulus−1 ) ;
    END downcnt ;
ARCHITECTURE Behavior OF downcnt IS
  SIGNAL Count : INTEGER RANGE 0 TO modulus−1 ;
BEGIN
   PROCESS
     BEGIN



END PROCESS;

END Behavior ;
```

Requirement:

- On the positive clock edge, if *L* = 1, the counter is loaded with the value *modulus*−1, and if *L* = 0, the count is decremented.
- The counter also includes an enable input, *E*. Setting *E* = 1 allows the count to be decremented when an active clock edge occurs

N-BIT DOWN COUNTER

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY downcnt IS
  GENERIC ( modulus : INTEGER := 8 ) ;
   PORT ( Clock, L, E : IN STD_LOGIC ;
                    Q : OUT INTEGER RANGE 0 TO (modulus - 1));
END downcnt;
ARCHITECTURE Behavior OF downcnt IS
  SIGNAL Count : INTEGER RANGE 0 TO modulus - 1 ;
BEGIN
   PROCESS
     BEGIN
        WAIT UNTIL (Clock'EVENT AND Clock = '1') ;
        IF L = '1' THEN
                      Count <= modulus - 1 ;
       ELSE
          IF E = '1' THEN
                      IF Count = 0 then
                        Count <= modulus - 1;
                      ELSE
                        Count <= Count - 1 ;
                      END IF;
          END IF ;
       END IF ;
   END PROCESS;
   Q <= Count ;
END Behavior ;
```

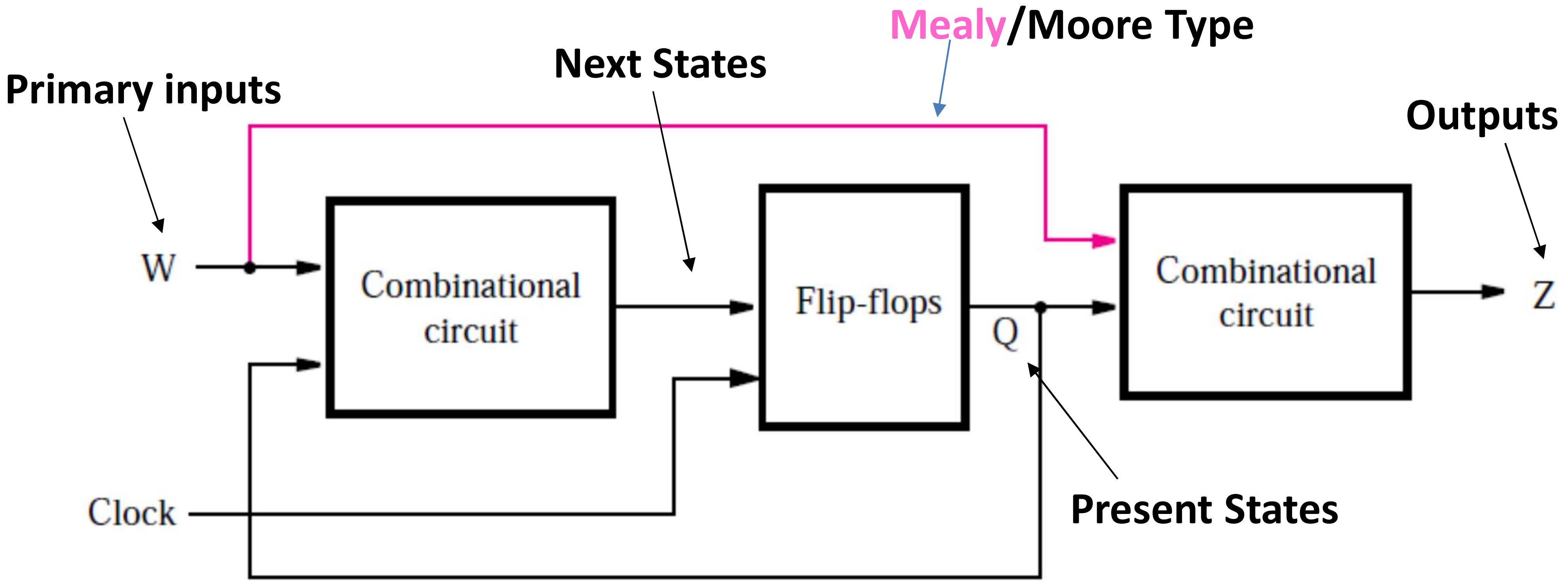
QUIZ

Write VHDL code for a n-bit up/down counter by using **GENERIC** declaration to set the number of bits of the counter. The counter has a port, called *dir*, that indicates the counting direction of the counter: if dir = 1 the counter functions as a up counter, and if dir = 0 the counter functions as a down counter

Synchronous Sequential Circuits

Synchronous Sequential Circuits

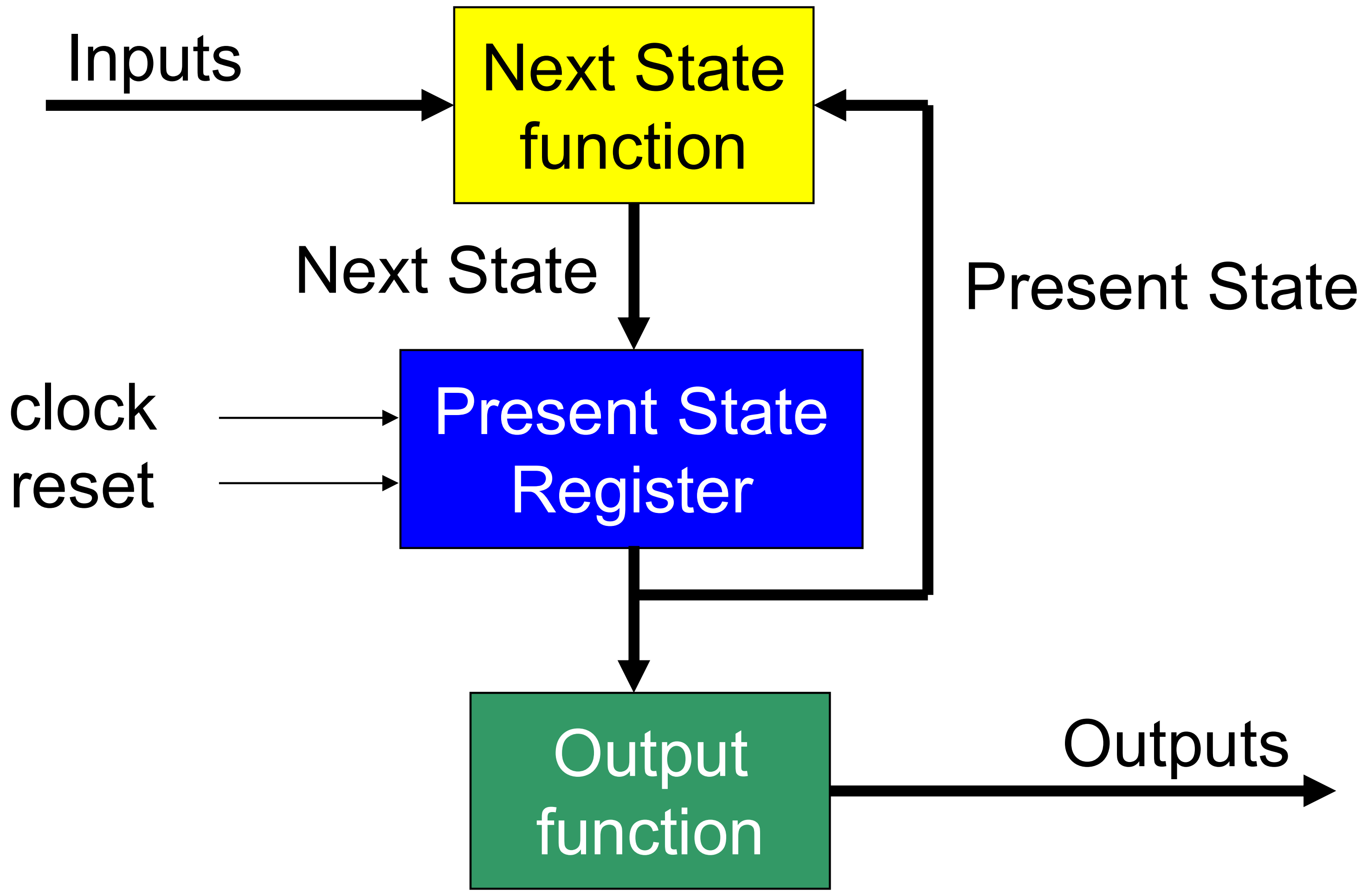
Sequential Circuits: the outputs depend on the past behavior of the circuit, as well as on the present values of inputs.
Synchronous: a clock signal is used to control the operation of a sequential circuit



Finite State Machines (FSMs)

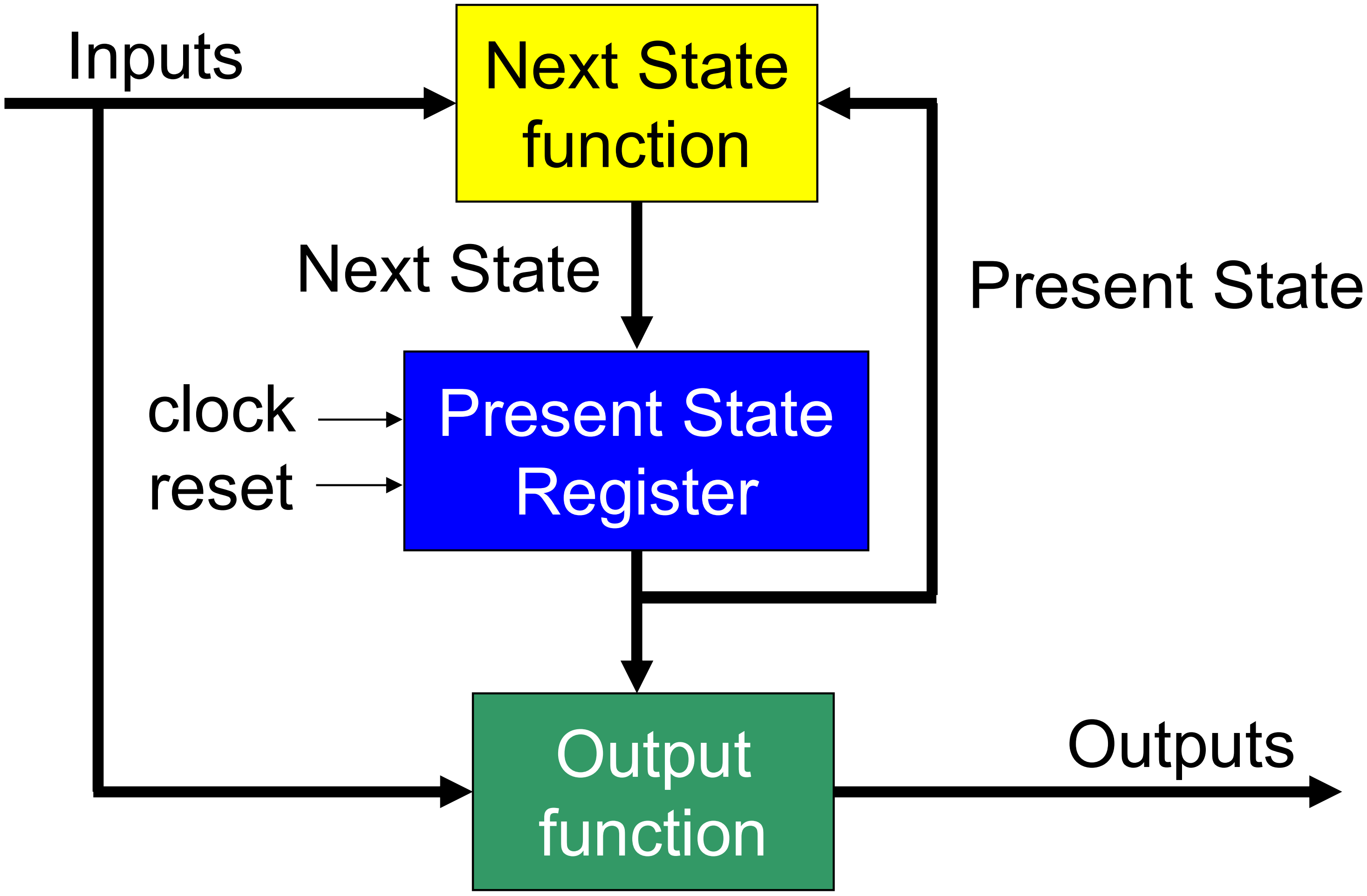
Synchronous Sequential Circuits

- Moore FSM: Output Is a Function of a Present State Only



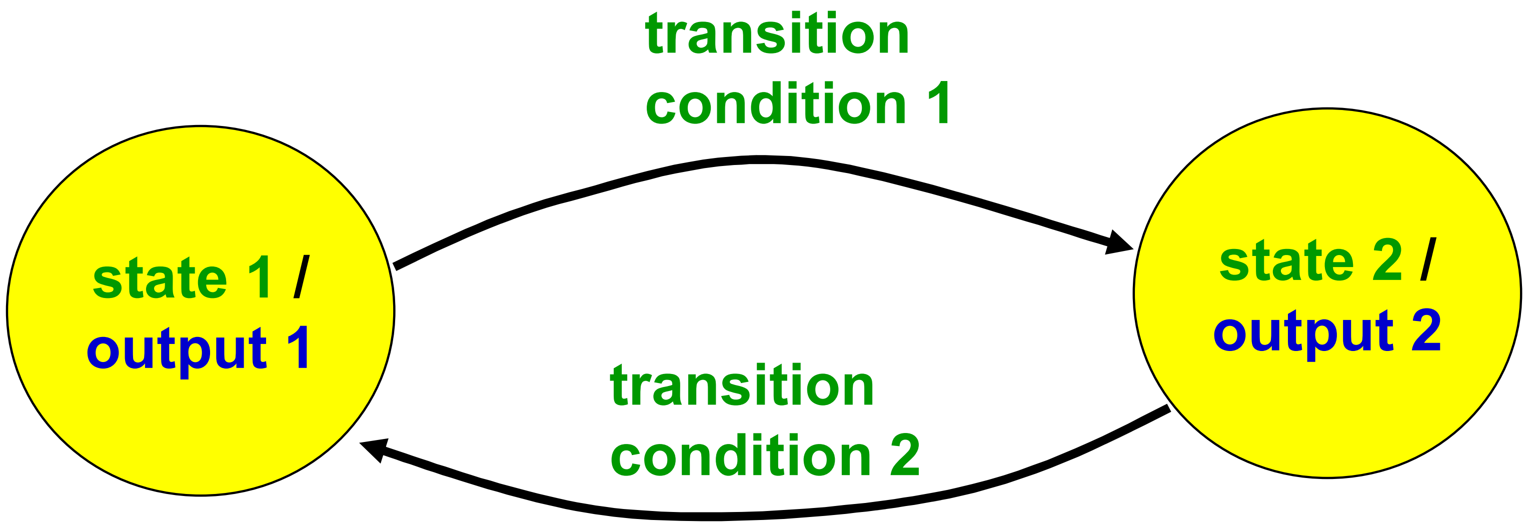
Synchronous Sequential Circuits

- Mealy FSM: Output Is a Function of a Present State and Inputs



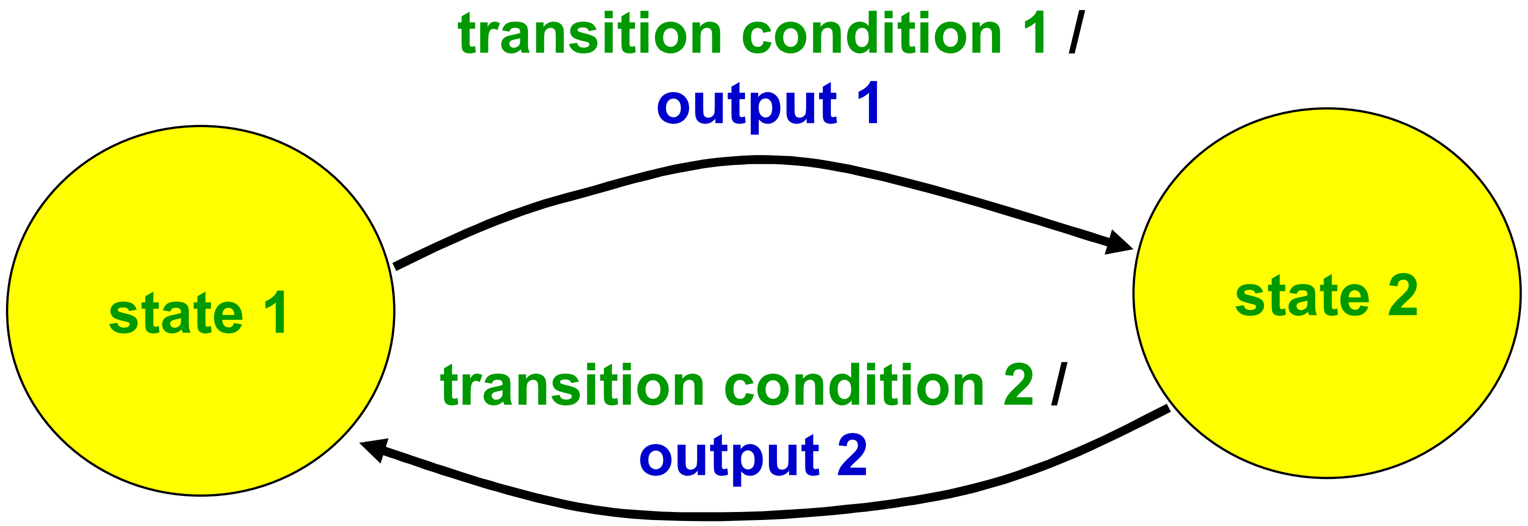
Synchronous Sequential Circuits

State diagram of Moore machine



Synchronous Sequential Circuits

State diagram of Mealy machine



Moore vs. Mealy FSM (1)

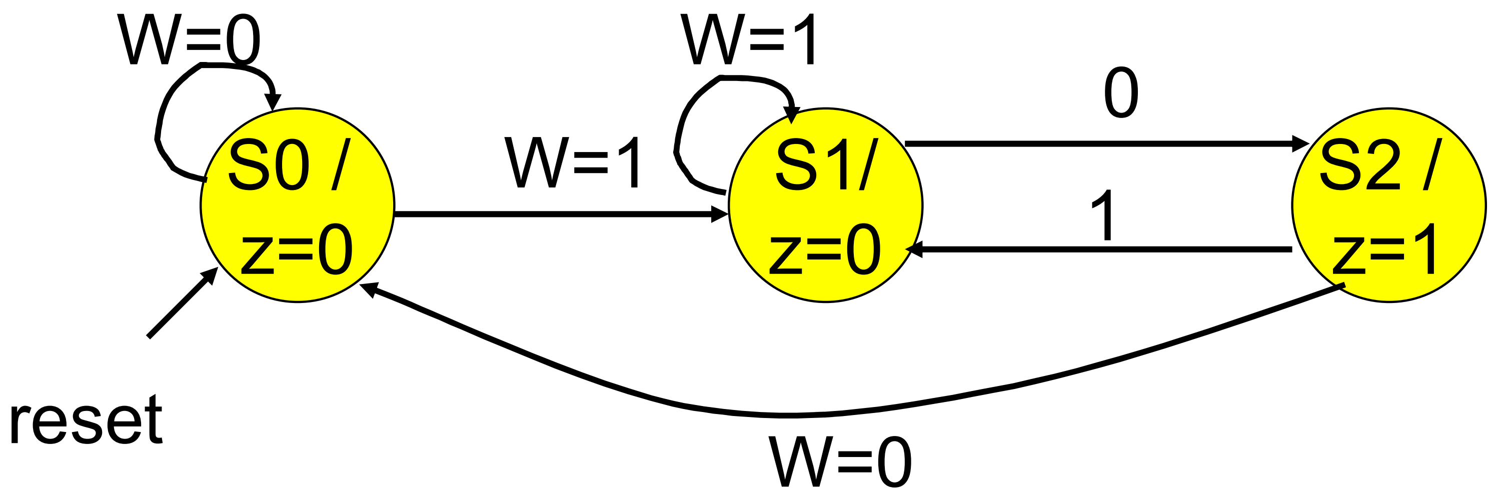
- Moore and Mealy FSMs Can Be Functionally Equivalent
  - Equivalent Mealy FSM can be derived from Moore FSM and vice versa
- Mealy FSM Has Richer Description and Usually Requires Smaller Number of States
  - Smaller circuit area

Moore vs. Mealy FSM (2)

- Mealy FSM Computes Outputs as soon as Inputs Change
  - Mealy FSM responds one clock cycle sooner than equivalent Moore FSM
- Moore FSM Has No Combinational Path Between Inputs and Outputs
  - Moore FSM is more likely to have a shorter critical path

Moore FSM - Example 1

- Moore FSM that Recognizes Sequence "10"



Meaning of states:

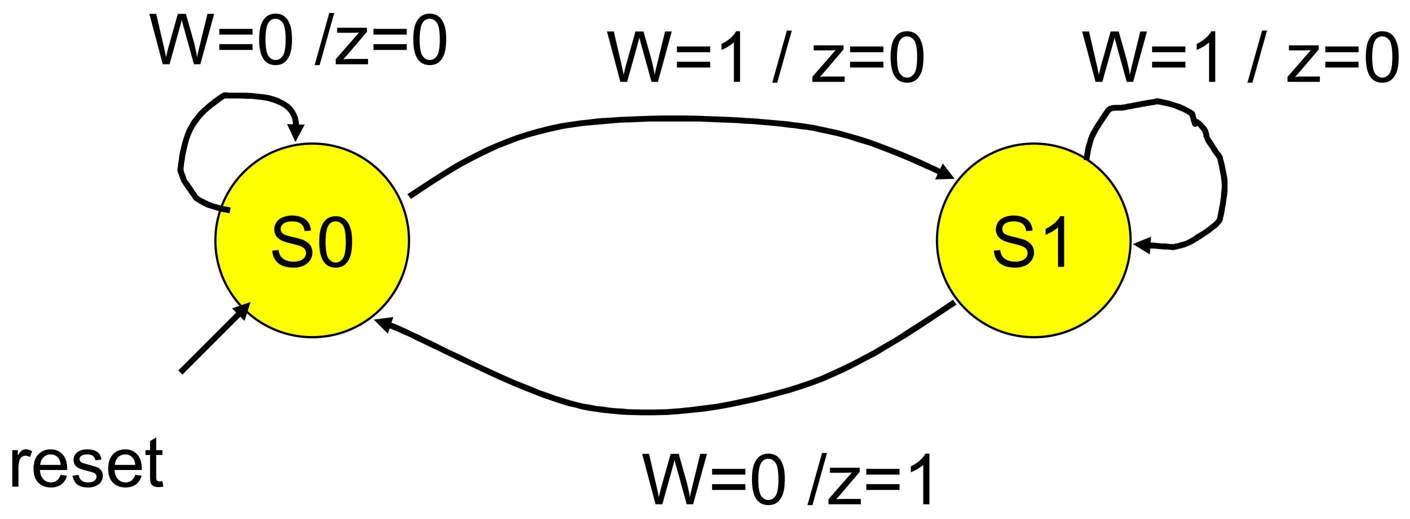
S0: No elements of the Sequence "10" observed

S1: "1" observed

S2: "10" observed

Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence “10”

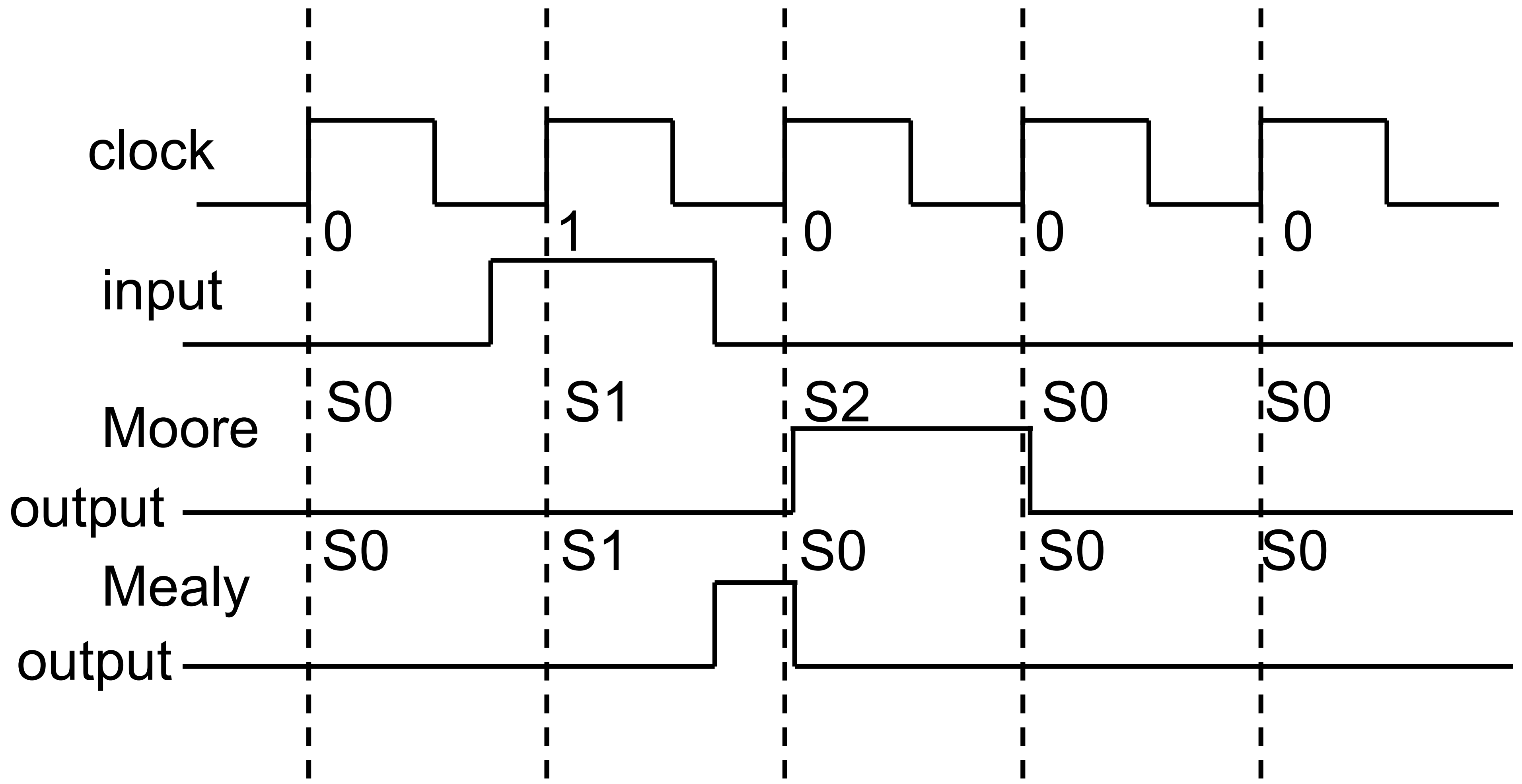


Meaning of states:

S0: No elements of the sequence observed

S1: “1” observed

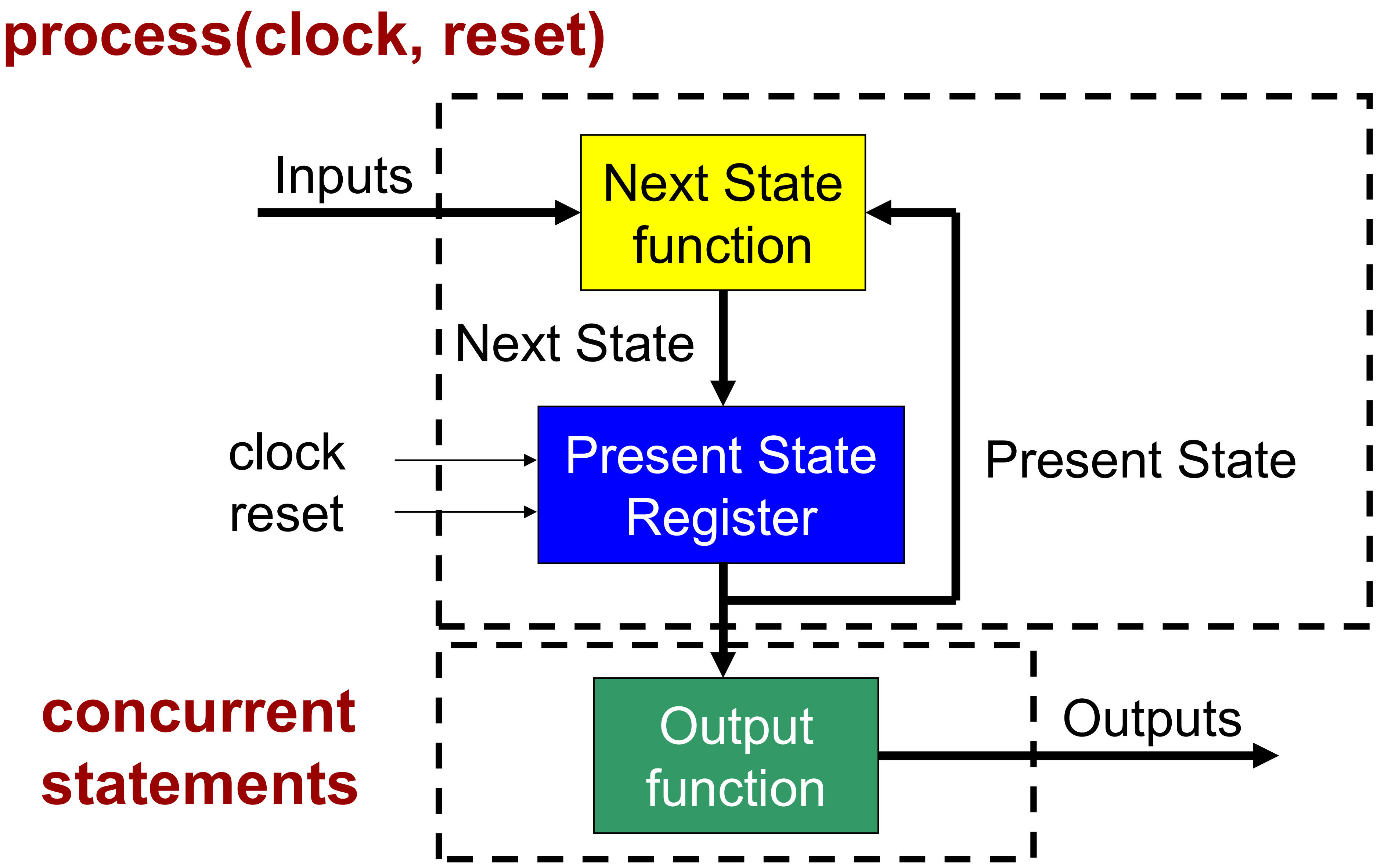
Moore & Mealy FSMs – Example 1



FSMs in VHDL

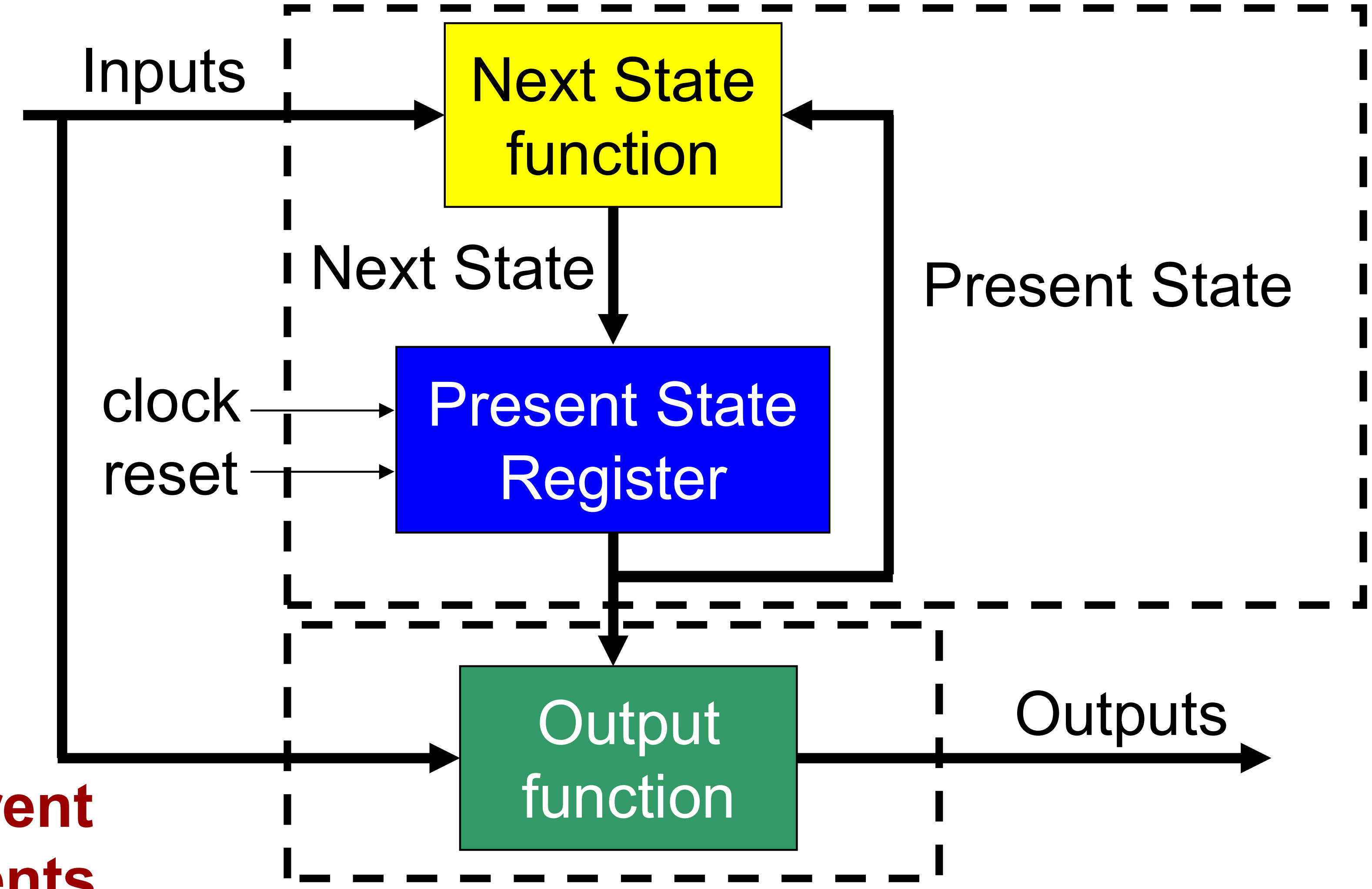
- Draw the state diagram using a graphical tool
- **Write VHDL code that represents the state diagram**
  - ➢ Finite State Machines Can Be Easily Described With Processes
  - ➢ Synthesis Tools Understand FSM Description If Certain Rules Are Followed
    - ✓ State transitions should be described in a process sensitive to *clock* and *asynchronous reset* signals only
    - ✓ Outputs described as concurrent statements outside the process

Moore FSM in VHDL



Mealy FSM in VHDL

process(clock, reset)

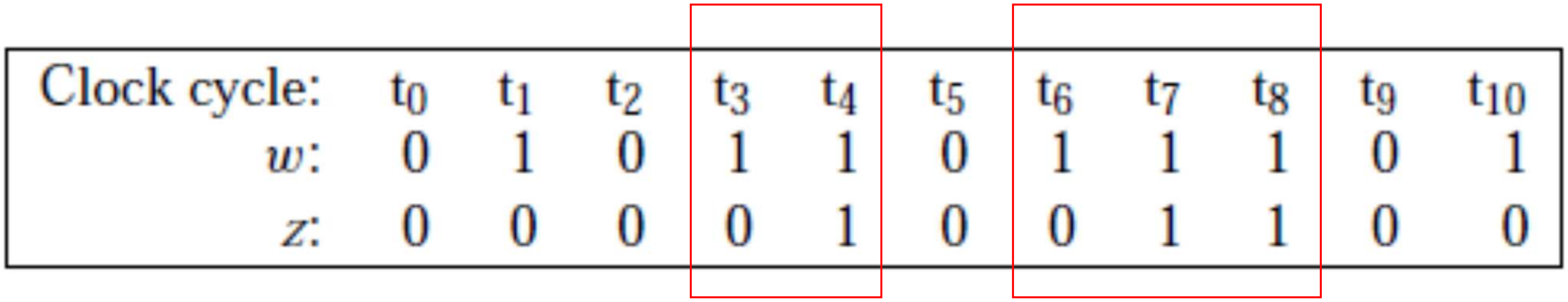


concurrent statements

Example of Moore machine

Sequence detector recognizes sequence "11":

1. The circuit has one input, w, and one output, z.
2. All changes in the circuit occur on the positive edge of a clock signal.
3. The output z is equal to 1 if during two consecutive clock cycles the input w was equal to 1. Otherwise, the value of z is equal to 0.



Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

Sequences of input and output signals

Example of Moore machine

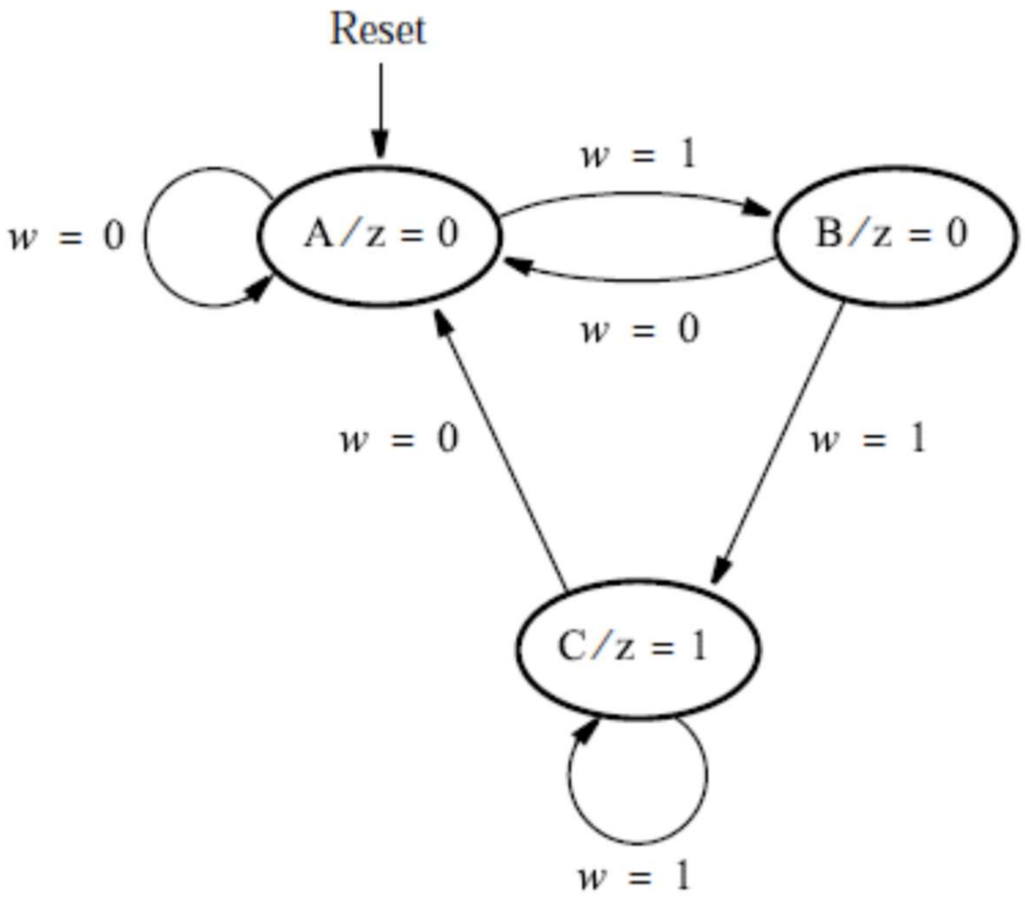
Building state diagram: how many states are needed and which transitions are possible from one state to another

State table

State diagram of "11" Sequence detector

Example of Moore machine

Building state diagram: how many states are needed and which transitions are possible from one state to another



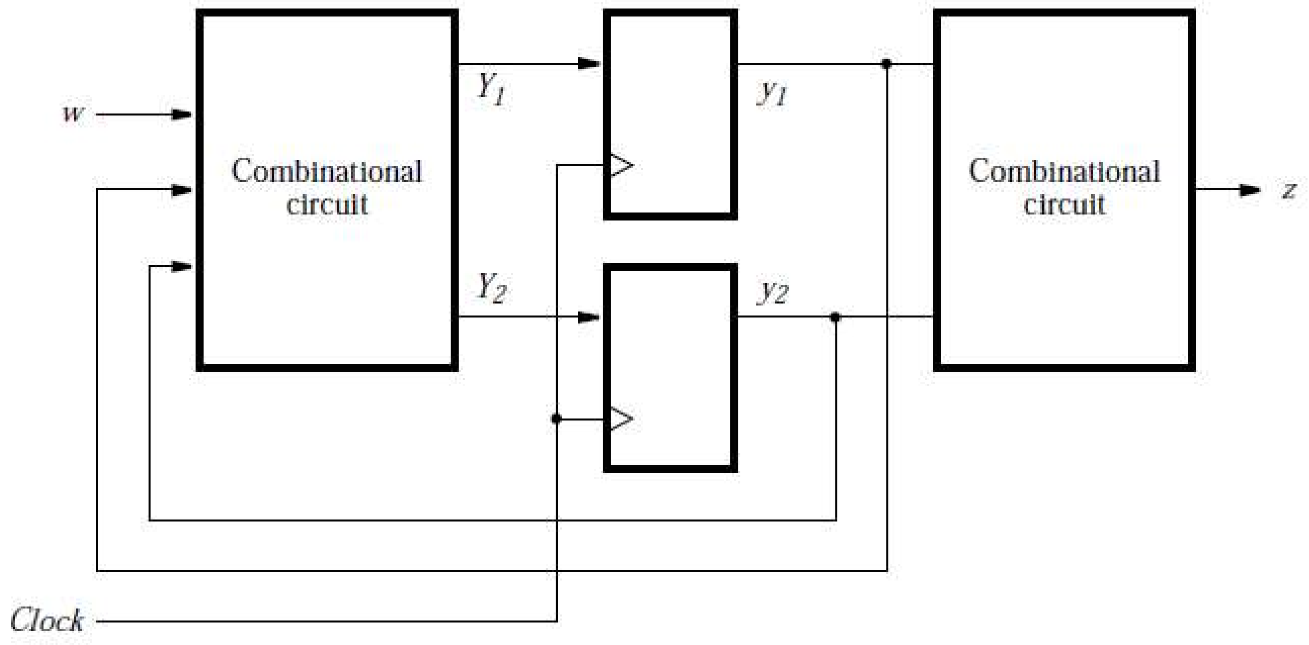
State diagram of "11" Sequence detector

Present state	Next state		Output
	$w=0$	$w=1$	z
A	A	B	0
B	A	C	0
C	A	C	1

State table

Example of Moore machine

Implementation of the circuit



Sequential circuit

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

State-assigned table

VHDL CODE

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY detector11 IS
   PORT ( Clock, nReset, w : IN STD LOGIC ;
               z : OUT STD LOGIC ) ;
   END detector11 ;
ARCHITECTURE Behavior OF detector11 IS
  TYPE State_type IS (A, B, C) ;
  SIGNAL y : State_type ;
BEGIN
-- State transitions
  PROCESS ( nResetn, Clock )
    BEGIN



  END PROCESS ;
   -- Output function

END Behavior ;
```

VHDL CODE

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY detector11 IS
   PORT ( Clock, nReset, w : IN STD LOGIC ;
                z : OUT STD LOGIC ) ;
   END detector11 ;
ARCHITECTURE Behavior OF detector11 IS
  TYPE State_type IS (A, B, C) ;
  SIGNAL y : State_type ;
BEGIN
  PROCESS ( nResetn, Clock )
    BEGIN
      IF Resetn = ’0’ THEN
                y <= A ;
      ELSIF (Clock’EVENT AND Clock = ’1’) THEN
        CASE y IS
          WHEN A =>
              IF w = ’0’ THEN
                y <= A ;
```

VHDL CODE

```
        ELSE
                  y <= B ;
        END IF ;
    WHEN B =>
      IF w =  ’0’ THEN
        y <= A ;
      ELSE
        y <= C ;
       END IF ;
    WHEN C =>
       IF w = ’0’ THEN
          y <= A ;
       ELSE
          y <= C ;
       END IF ;
     END CASE ;
   END IF ;
  END PROCESS ;
    z <= ’1’ WHEN y = C ELSE ’0’ ;
 END Behavior ;
```

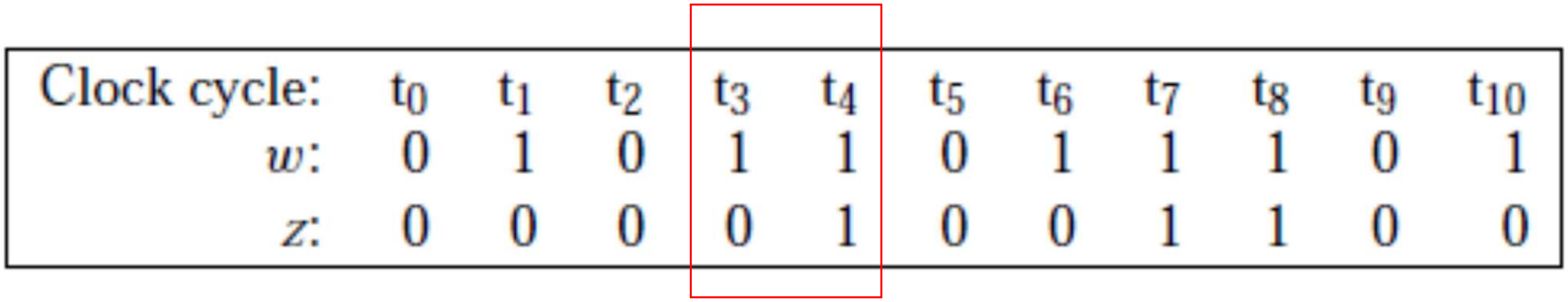
QUIZ

Write testbench and use ModelSim to simulate the above detector?

Example of Mealy machine

Sequence detector recognizes sequence “11”:

1. The circuit has one input, w, and one output, z.
2. All changes in the circuit occur on the positive edge of a clock signal.
3. The output z is equal to 1 if during two consecutive clock cycles the input w was equal to 1. Otherwise, the value of z is equal to 0.



Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

Sequences of input and output signals

Example of Mealy machine

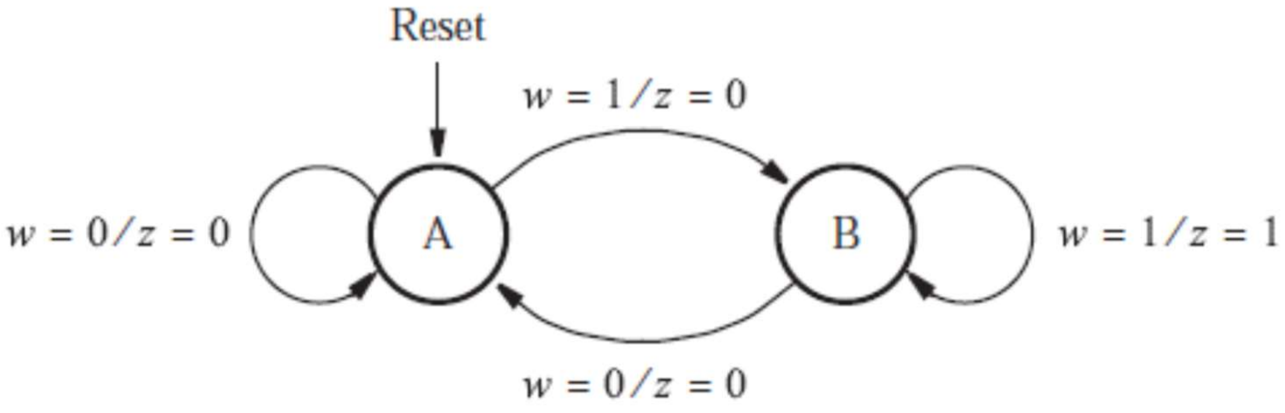
Building state diagram: how many states are needed and which transitions are possible from one state to another

State diagram of "11" Sequence detector

State table

Example of Mealy machine

Building state diagram: how many states are needed and which transitions are possible from one state to another



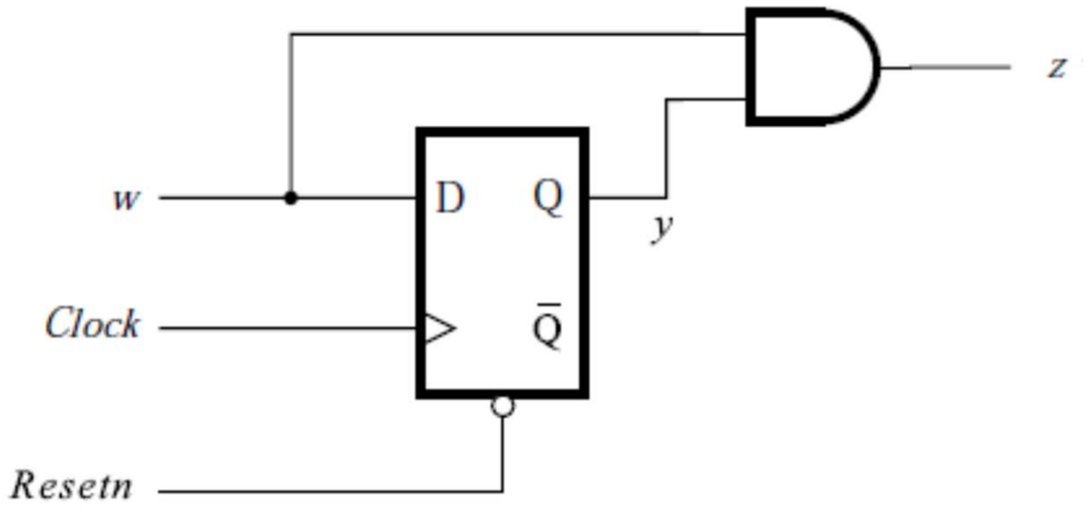
State diagram of "11" Sequence detector

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

State table

Example of Mealy machine

Implementation of the circuit



Sequential circuit

	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

State-assigned table

VHDL CODE

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY detector11 IS
   PORT ( Clock, nReset, w : IN STD LOGIC ;
                z : OUT STD LOGIC ) ;
   END simple ;
ARCHITECTURE Behavior OF detector11 IS
  TYPE State_type IS (A, B) ;
  SIGNAL y : State_type ;
BEGIN
-- State transitions
  PROCESS ( nResetn, Clock )
    BEGIN



  END PROCESS ;
   -- Output function

END Behavior ;
```

VHDL CODE

```
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
ENTITY detector11 IS
   PORT ( Clock, Resetn, w : IN STD LOGIC ;
                  z : OUT STD LOGIC ) ;
   END detector11 ;
ARCHITECTURE Behavior OF detector11 IS
    TYPE State type IS (A, B) ;
    SIGNAL y : State type ;
 BEGIN
   PROCESS ( Resetn, Clock )  -- for state transition
     BEGIN
        IF Resetn = ’0’ THEN
            y <= A ;
        ELSIF (Clock’EVENT AND Clock = ’1’) THEN
           CASE y IS
              WHEN A =>
                 IF w = ’0’ THEN  y <= A ;
                 ELSE
                                   y <= B ;
                 END IF ;
```

VHDL CODE

```
      WHEN B =>
        IF w  = '0' THEN
                y <= A ;
        ELSE
                 y <= B ;
        END IF ;
    END CASE ;
  END IF ;
 END PROCESS ;
PROCESS ( y, w ) -- for output transition
 BEGIN
   CASE y IS
      WHEN A =>
         z <= '0' ;
      WHEN B =>
         z <= w ;
     END CASE ;
 END PROCESS ;
END Behavior ;
```

QUIZ

Write testbench and use ModelSim to simulate the above detector?

Writing Testbenches

```
ENTITY detector11_tb IS END ENTITY;
--
ARCHITECTURE bev OF detector11_tb IS
  SIGNAL w, clock, nReset, z : std_logic := '0';
  COMPONENT detector11 IS
  PORT ( Clock, nReset, w : IN STD LOGIC ;
             z : OUT STD LOGIC ) ;
  END COMPONENT ;

BEGIN
  DUT1: detector110
      PORT MAP (Clock, nReset, w, z);
  rst <= '1' AFTER 31 NS, '0' AFTER 54 NS;
  clock <= NOT clock AFTER 7 NS WHEN NOW<=165 NS ELSE '0';
  PROCESS BEGIN
    WAIT FOR 23 NS; w <= ‘0';
    WAIT FOR 21 NS; w <= ‘1';
    WAIT FOR 19 NS; w <= '1';
    WAIT FOR 31 NS; w <= '0';
    WAIT;
  END PROCESS;
  PROCESS (w) BEGIN
    REPORT "Signal w changed to:"& std_logic'IMAGE(w)& " at " & TIME'IMAGE(NOW)  SEVERITY NOTE;
  END PROCESS;
END ARCHITECTURE bev;
```