# EDA322
# Digital Design

Lecture 9:
**Finite State Machines - VHDL**

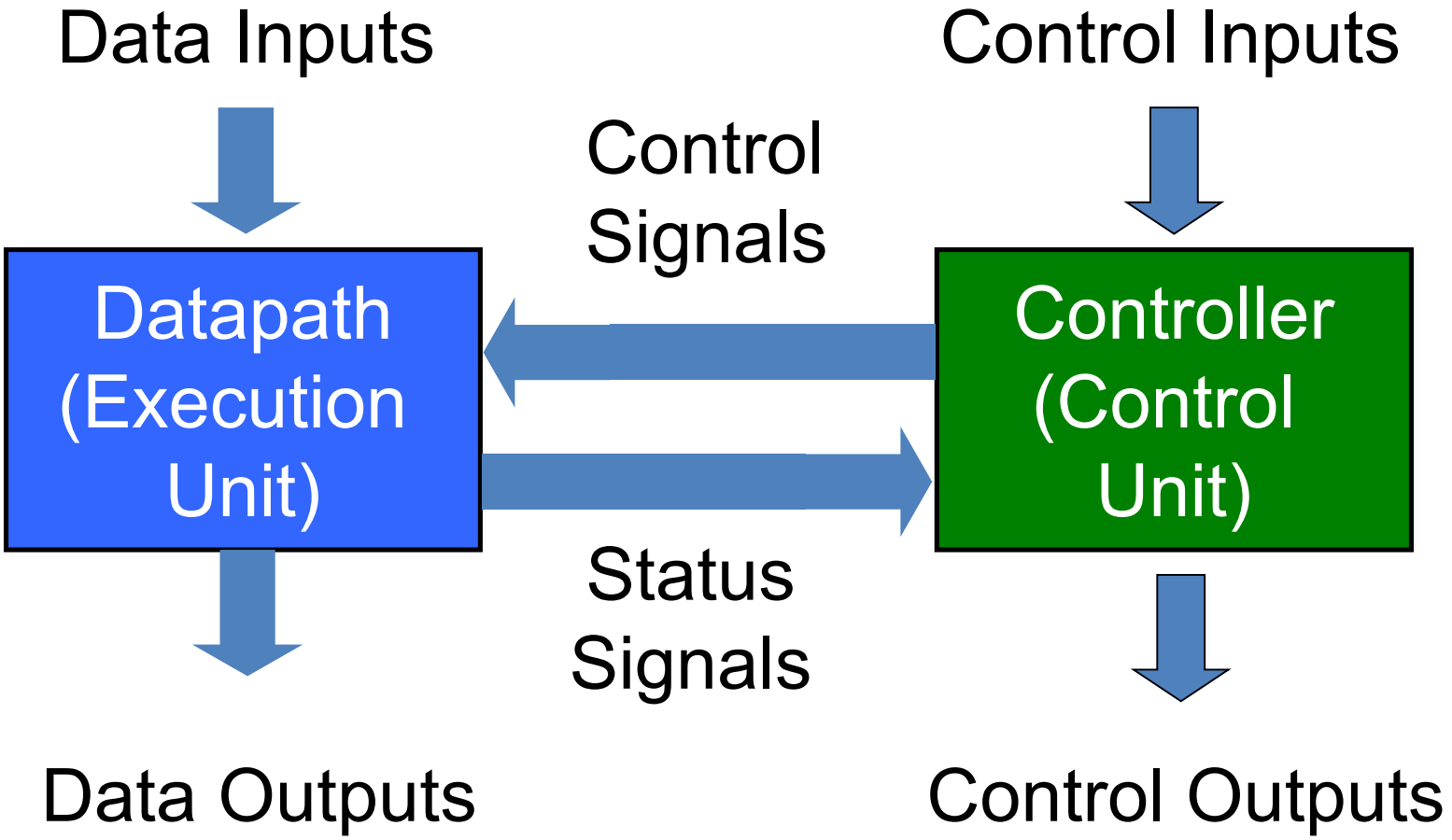
Ioannis Sourdis

# Outline of Lecture 9

- Control vs. Datapath
- More FSM examples
- FSMs in VHDL
  - Moore and Mealy FSMs in VHDL
  - State processes
  - State Coding
  - More VHDL alternatives (Mealy, Moore, registered output)

# Datapath
# vs.
# Controller

# Structure of a Typical Digital System



# Datapath (Execution Unit)

- Manipulates and processes data
- Performs arithmetic and logic operations, shifting, and other data-processing tasks
- Is composed of registers, gates, multiplexers, decoders, adders, comparators, ALUs, etc.
- Provides all necessary resources and interconnects among them to perform specified task
- Interprets control signals from the Controller and generates status signals for the Controller

# Controller (Control Unit)

- Controls data movements in the Datapath by switching multiplexers and enabling or disabling resources
  - Example: enable signals for registers
  - Example: control signals for muxes
- Provides signals to activate various processing tasks in the Datapath
- Determines the sequence the operations performed by Datapath
- Follows Some ‘Program’ or Schedule

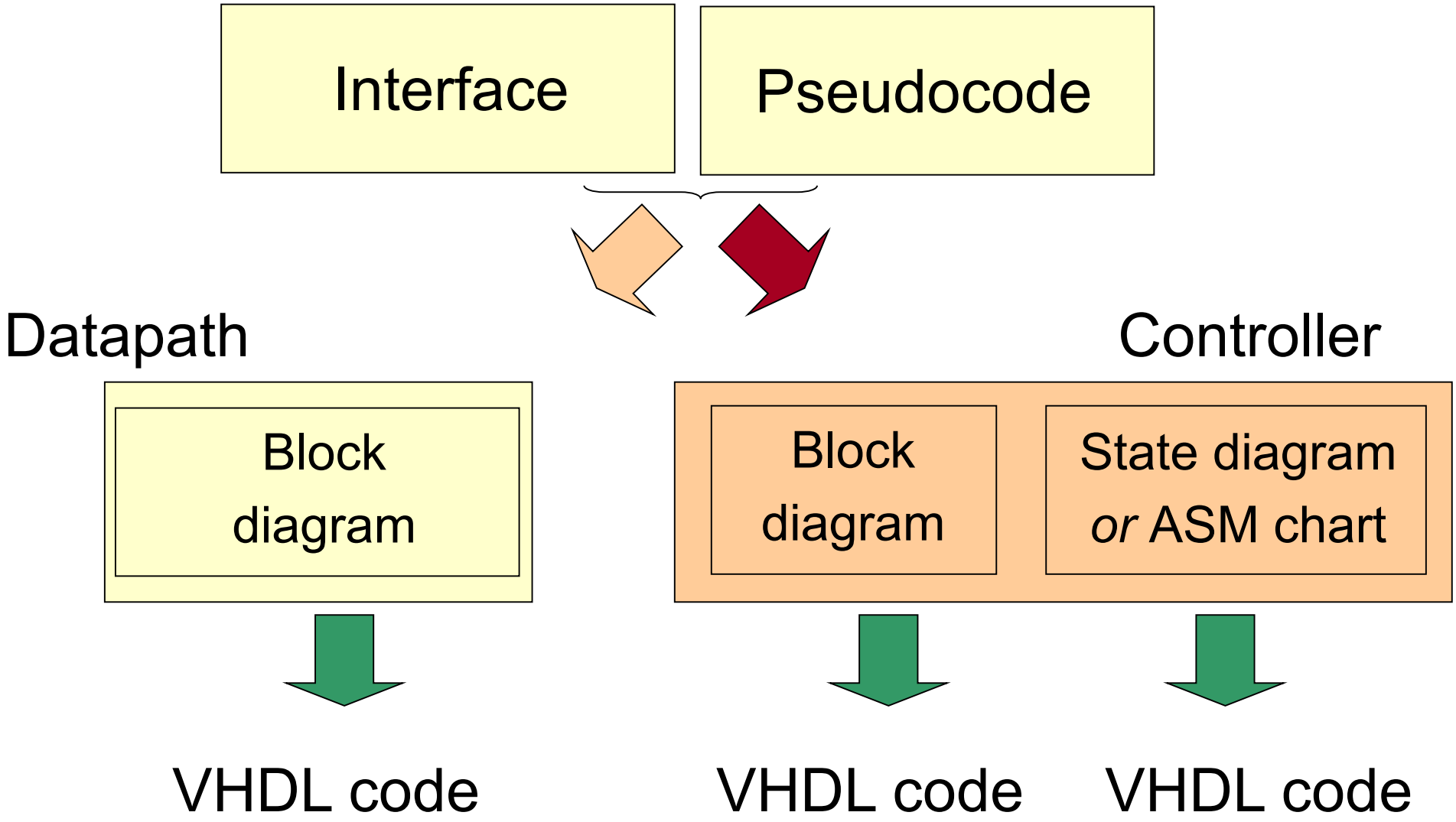
# Controller

- Controller can be programmable or non-programmable
- Programmable
  - Has a program counter which points to next instruction
  - Instructions are held in a RAM or ROM externally
  - Microprocessor is an example of programmable controller
- Non-Programmable
  - Once designed, implements the same functionality
  - Another term is a “hardwired state machine” or “hardwired instructions”

# Finite State Machines

- Digital Systems and especially their Controllers can be described as Finite State Machines (FSMs)
- Finite State Machines can be represented using
  - **State Diagrams and State Tables** - suitable for simple digital systems with a relatively few inputs and outputs
  - **Algorithmic State Machine (ASM) Charts** - suitable for complex digital systems with a large number of inputs and outputs

# Hardware Design with RTL VHDL



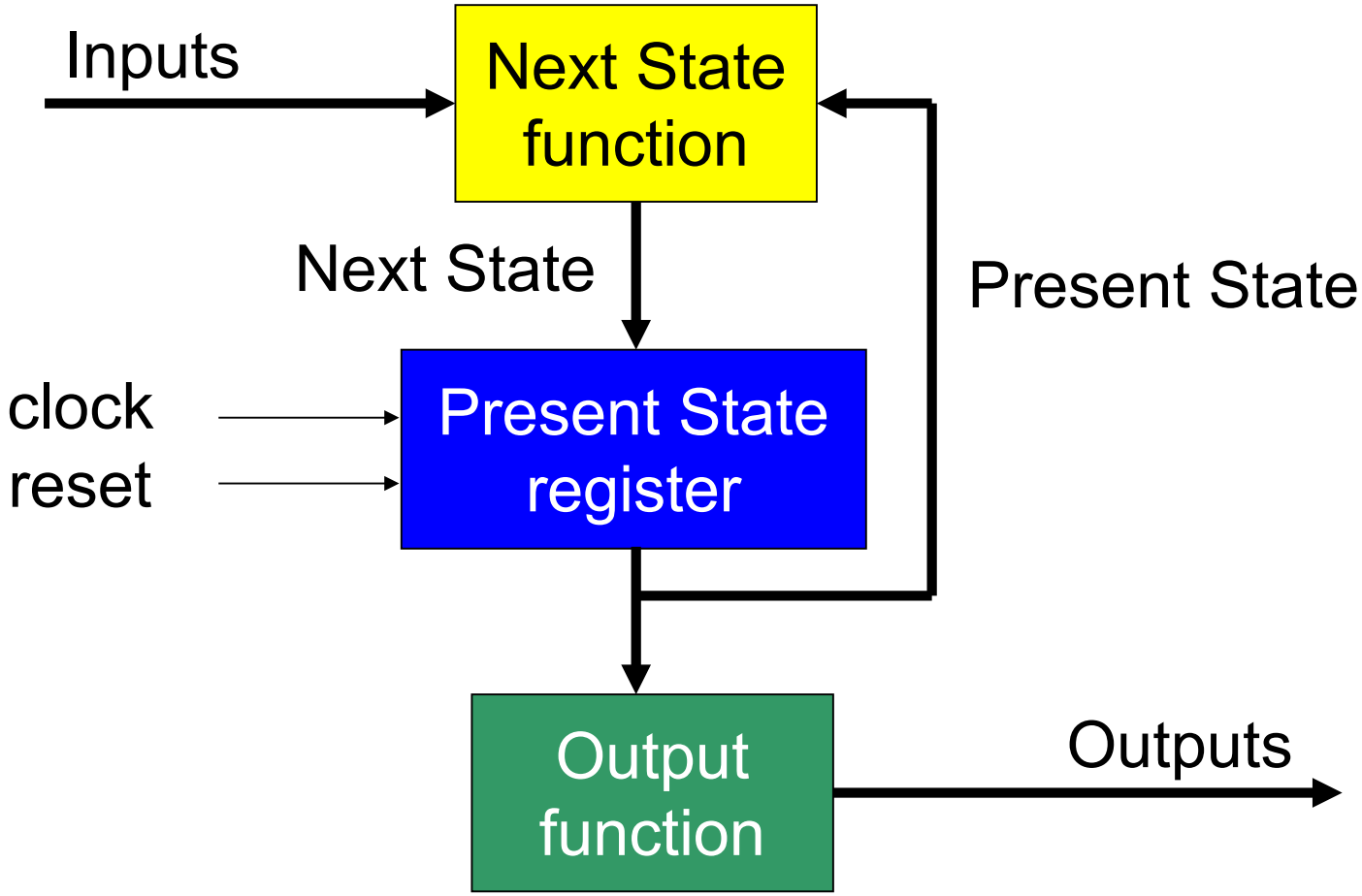
# Finite State Machines Refresher

# Finite State Machines (FSMs)

- A machine/mathematical model with a finite amount of memory to represent its finite set of internal states, has a set of defined inputs and outputs and a set of transitions between selected states
  - Even computers can be viewed as huge FSMs
- Design of FSM
  - Defining states
  - Defining transitions between states
  - Optimization / minimizations
    - **Manual Optimization/Minimization Is Practical for Small FSMs Only**

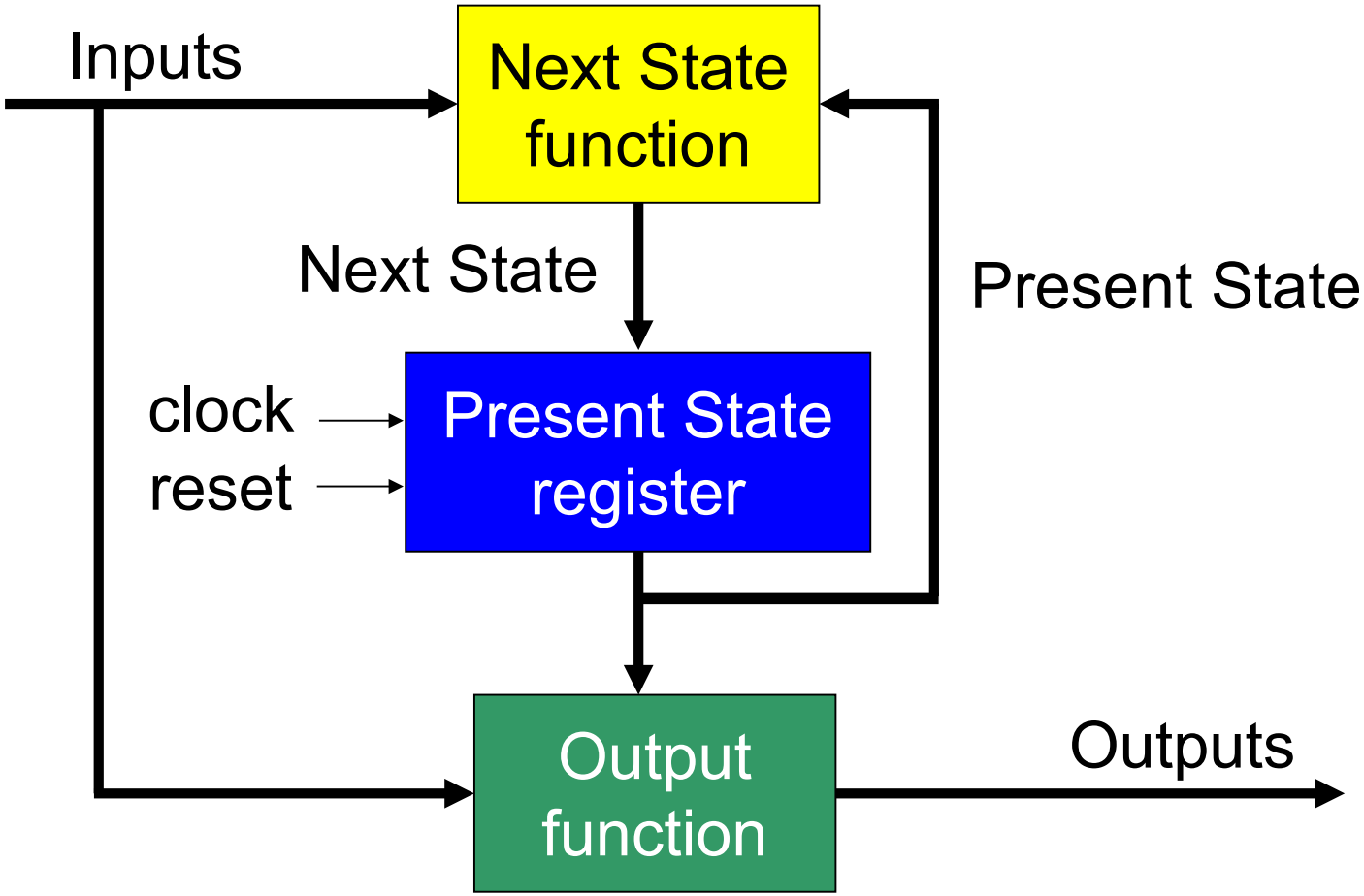
# Moore FSM

- Output Is a Function of a Present State Only



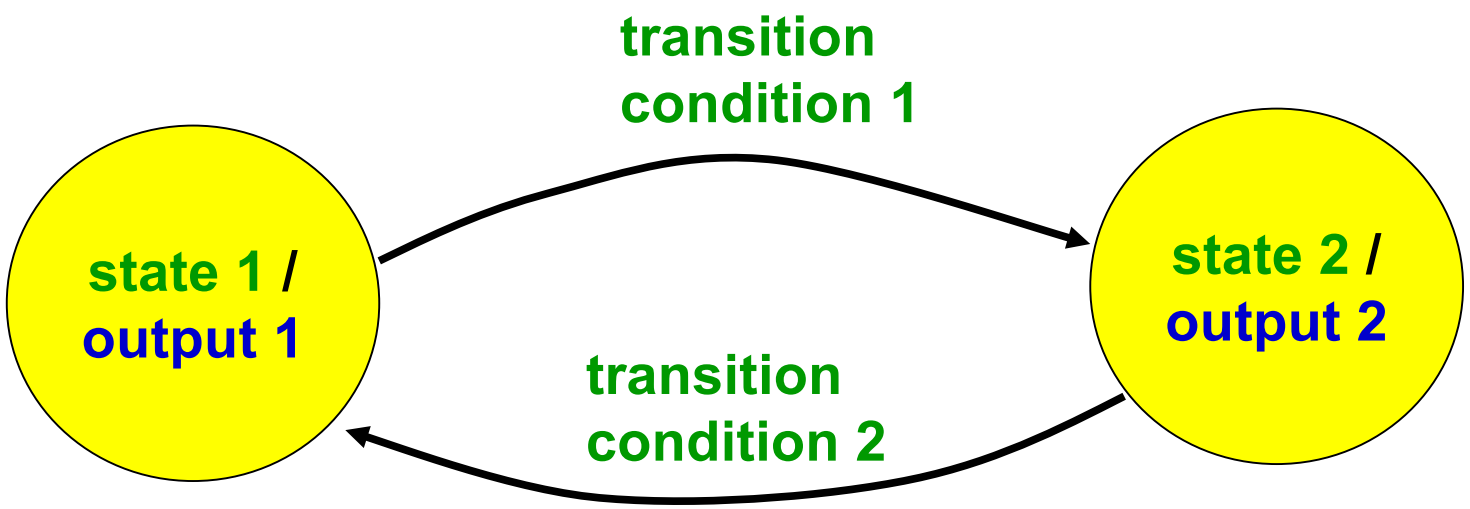
# Mealy FSM

- Output Is a Function of a Present State and Inputs

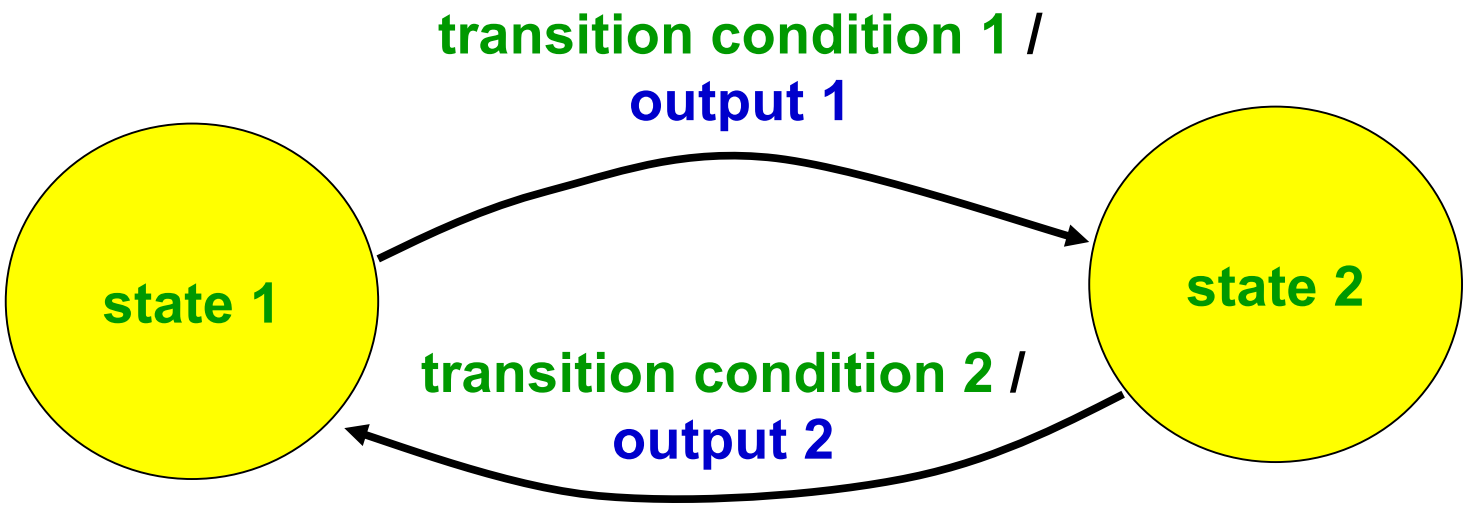


# State Diagrams

# Moore Machine



# Mealy Machine



# Moore vs. Mealy FSM (1)

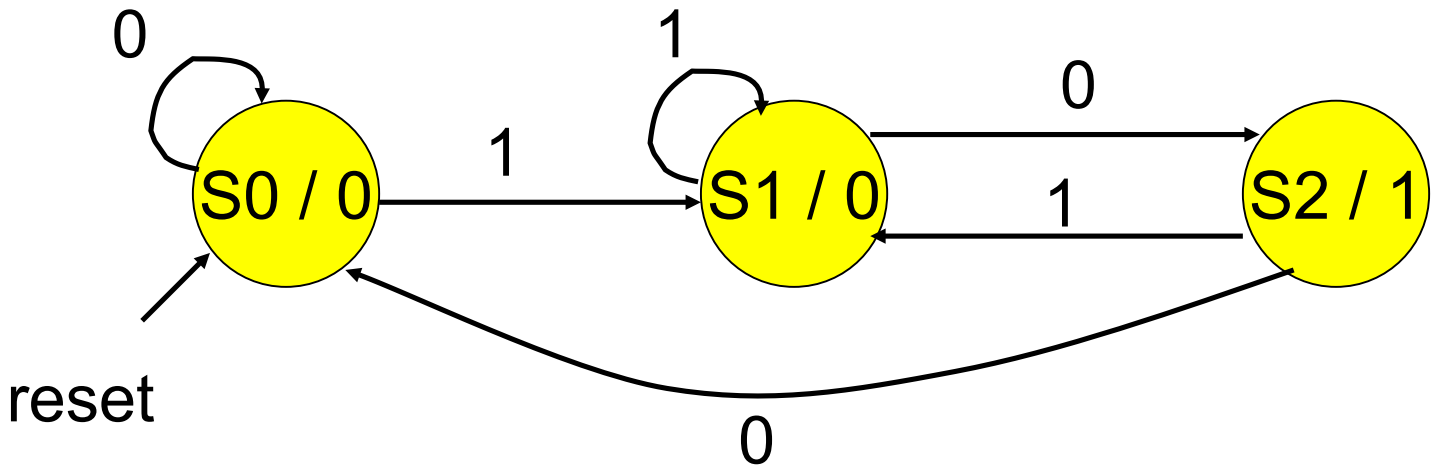
- Moore and Mealy FSMs Can Be Functionally Equivalent
  - Equivalent Mealy FSM can be derived from Moore FSM and vice versa
- Mealy FSM Has Richer Description and Usually Requires Smaller Number of States
  - Smaller circuit area

# 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:

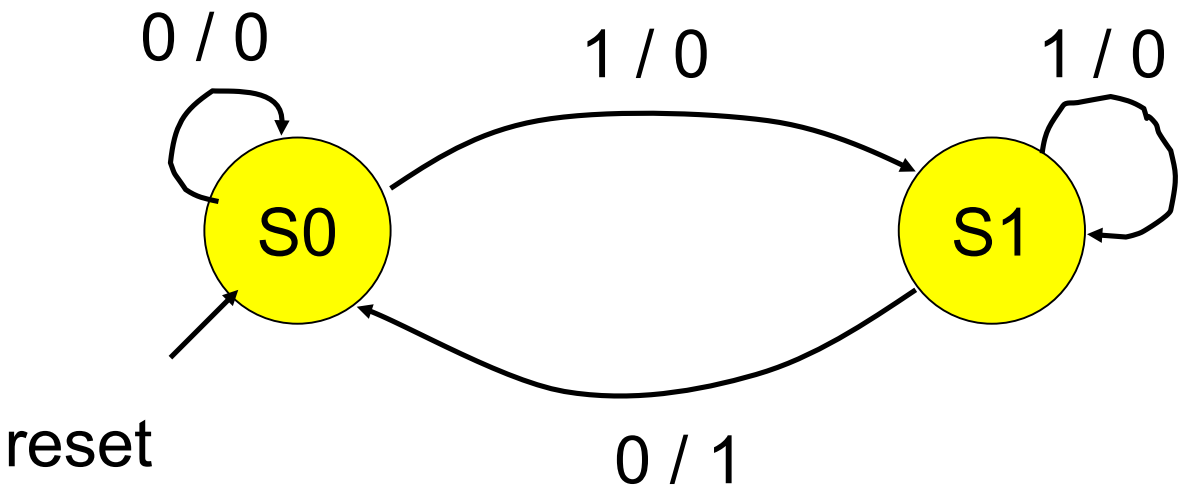
S0: No elements of the sequence observed

S1: “1” observed

S2: “10” observed

# Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence “10”

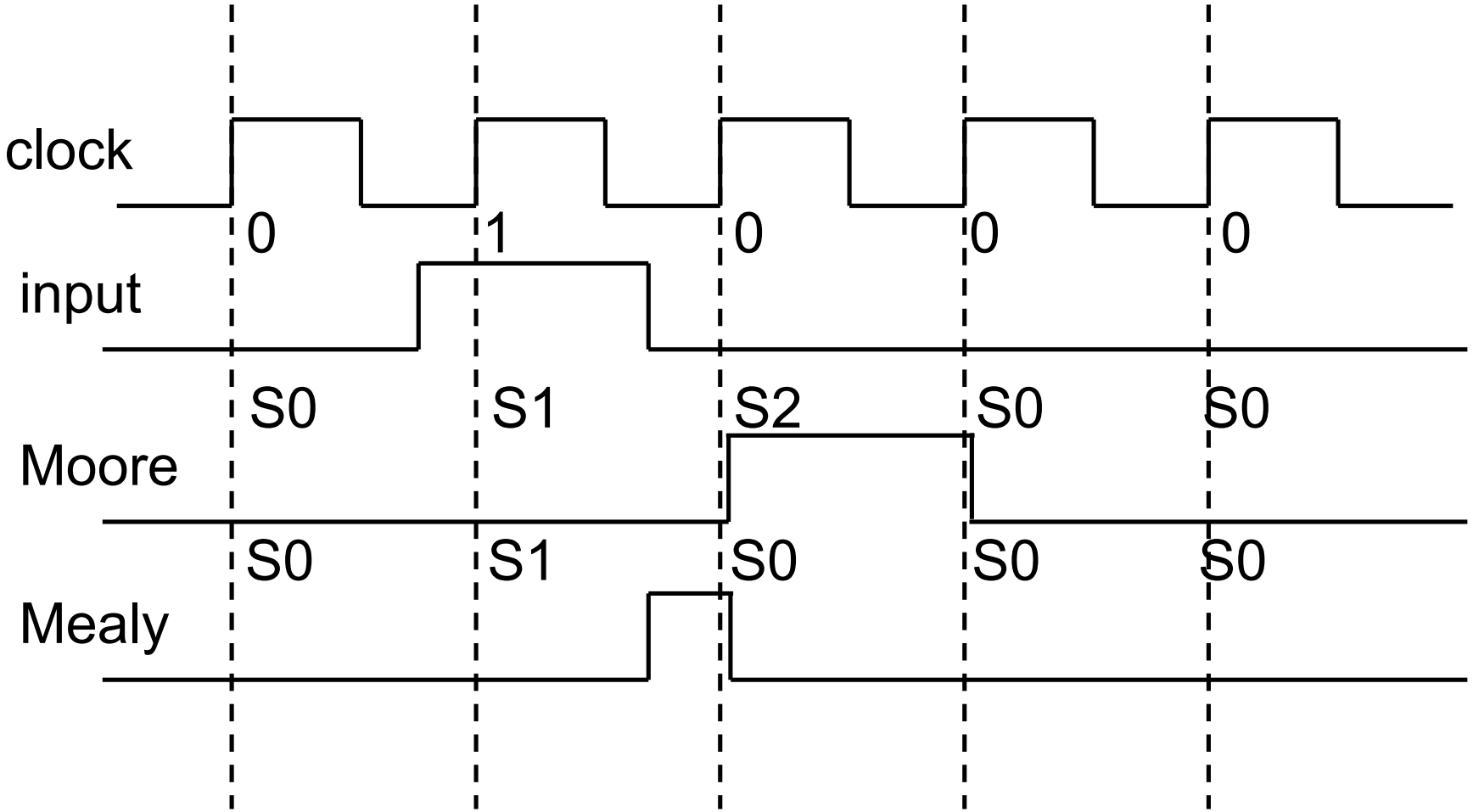



Meaning of states:

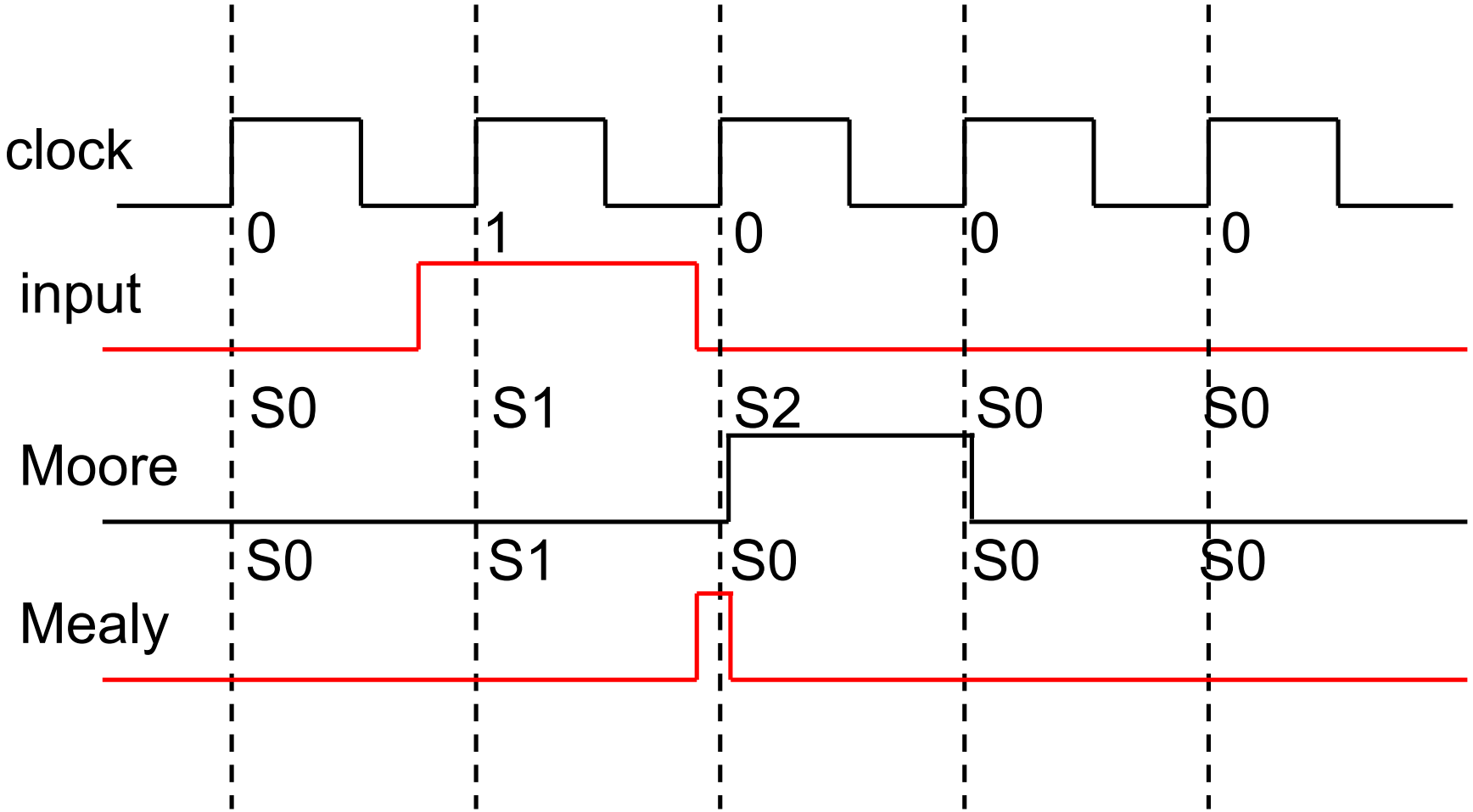
S0: No elements of the sequence observed

S1: “1” observed

# Moore & Mealy FSMs – Example 1



# Moore & Mealy FSMs – Example 1



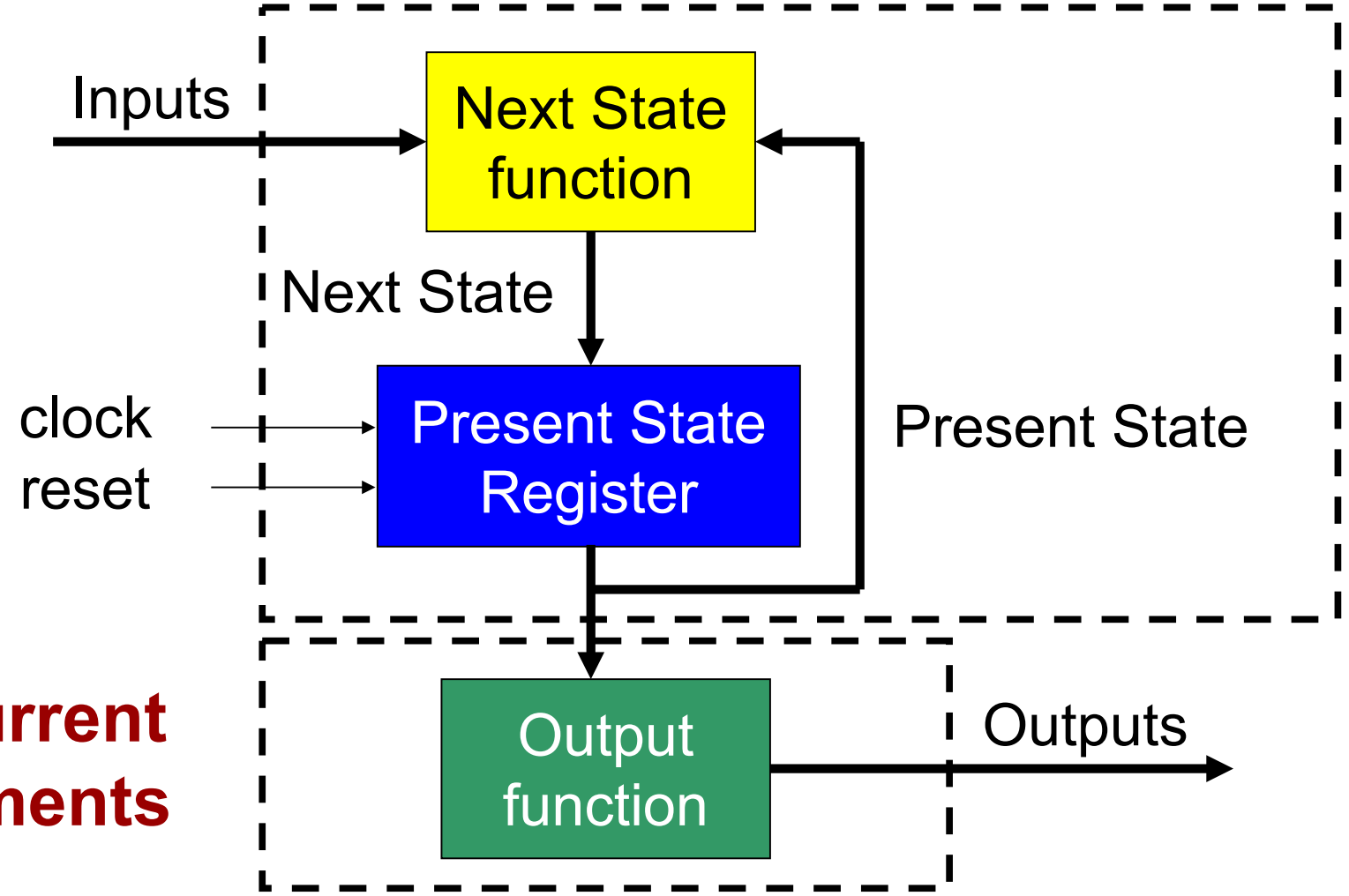
# Finite State Machines in VHDL

# FSMs in VHDL

- 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

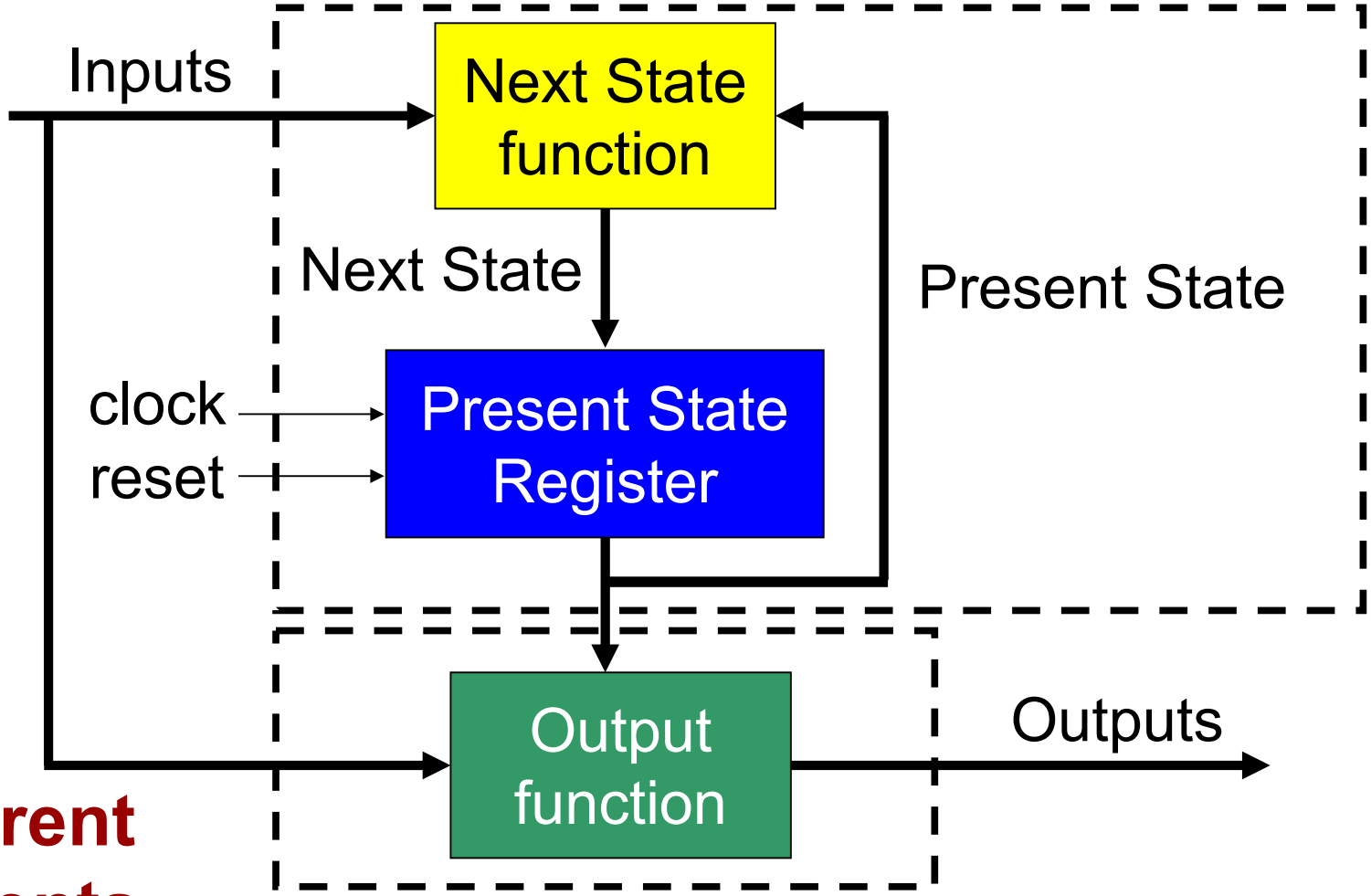
**process(clock, reset)**




**concurrent statements**

# Mealy FSM

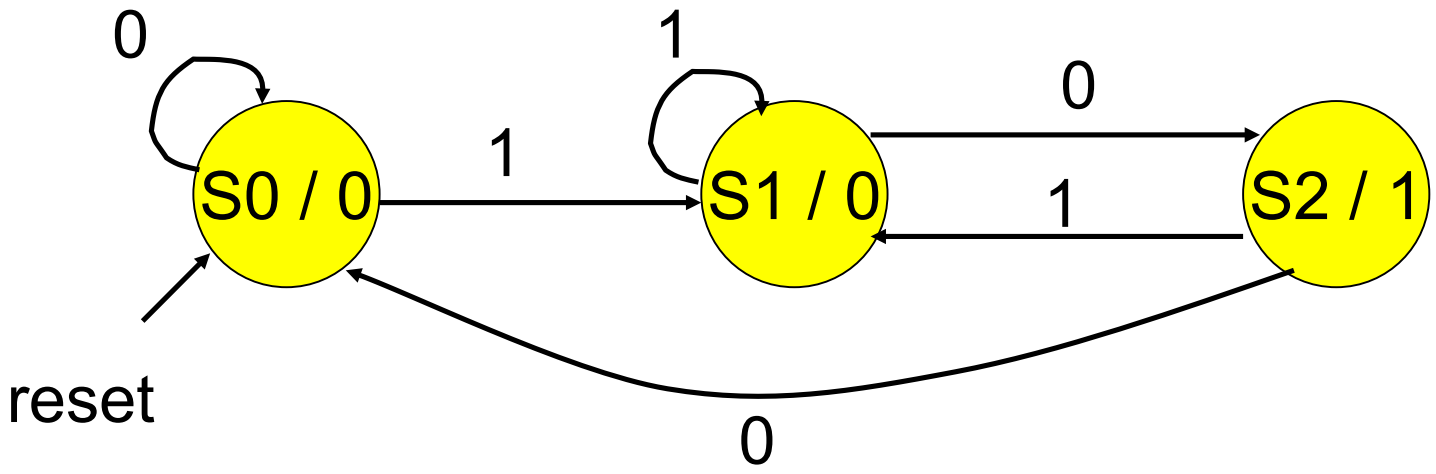
**process(clock, reset)**




**concurrent statements**

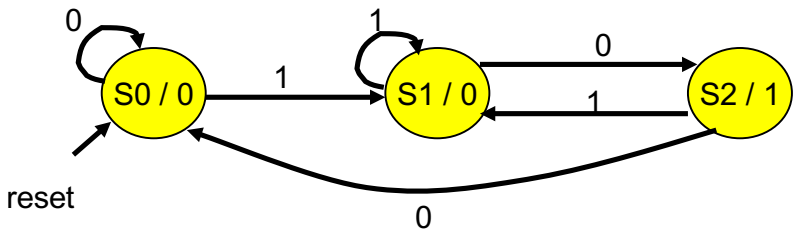
# Moore FSM - Example 1

- Moore FSM that Recognizes Sequence “10”



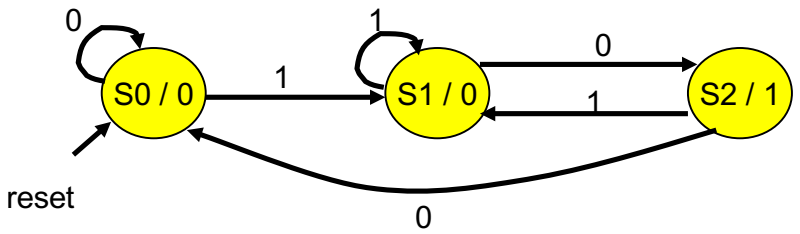
# Moore FSM in VHDL (1)

```
TYPE state IS (S0, S1, S2);
SIGNAL Moore_state: state;

U_Moore: PROCESS (clock, reset)
BEGIN
      IF(reset = '1') THEN
         Moore_state <= S0;
      ELSIF (clock = '1' AND clock'event) THEN
         CASE Moore_state IS
            WHEN S0 =>
             IF input = '1' THEN
                Moore_state <= S1;
             ELSE
               Moore_state <= S0;
             END IF;
```



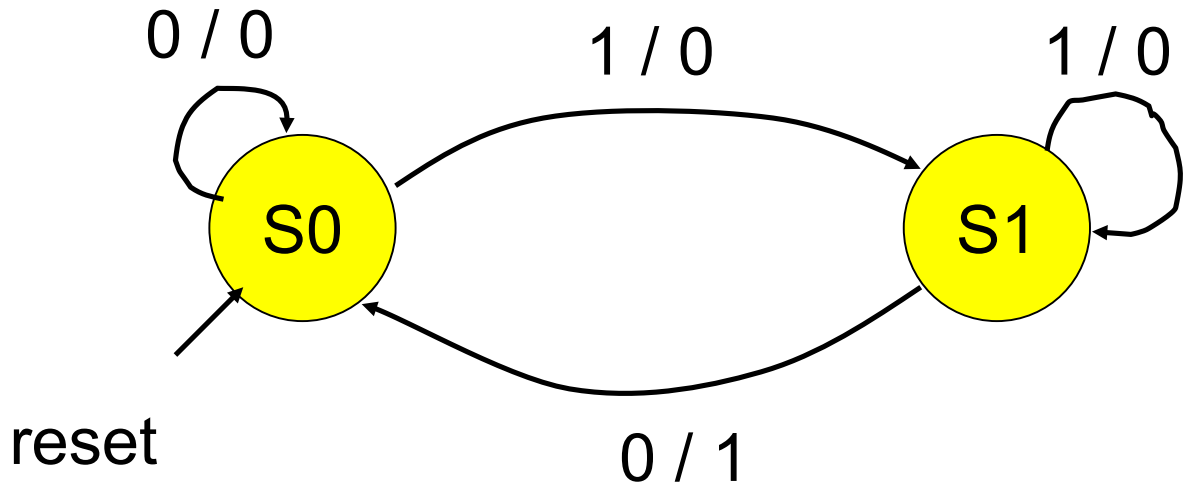
# Moore FSM in VHDL (2)

```
            WHEN S1 =>
                    IF input = '0' THEN
                    Moore_state <= S2;
                ELSE
                    Moore_state <= S1;
                END IF;
             WHEN S2 =>
                    IF input = '0' THEN
                    Moore_state <= S0;
                     ELSE
                    Moore_state <= S1;
                END IF;
        END CASE;
    END IF;
END PROCESS;

Output <= '1' WHEN Moore_state = S2 ELSE '0';
```



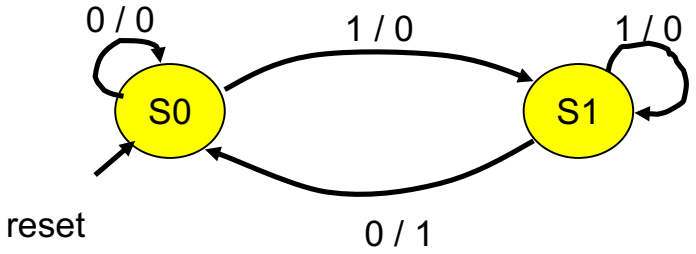
# Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence “10”



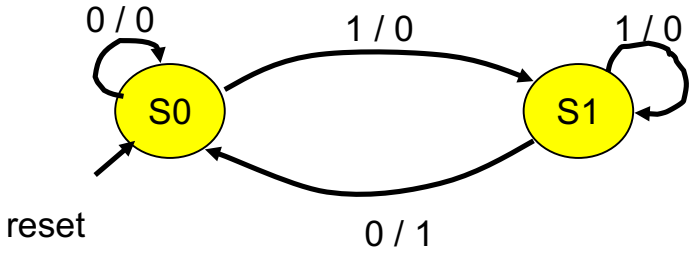
# Mealy FSM in VHDL (1)

```
TYPE state IS (S0, S1);
SIGNAL Mealy_state: state;

U_Mealy: PROCESS(clock, reset)
BEGIN
      IF(reset = '1') THEN
         Mealy_state <= S0;
      ELSIF (clock = '1' AND clock'event) THEN
         CASE Mealy_state IS
               WHEN S0 =>
                  IF input = '1' THEN
                     Mealy_state <= S1;
                  ELSE
                     Mealy_state <= S0;
                  END IF;
```



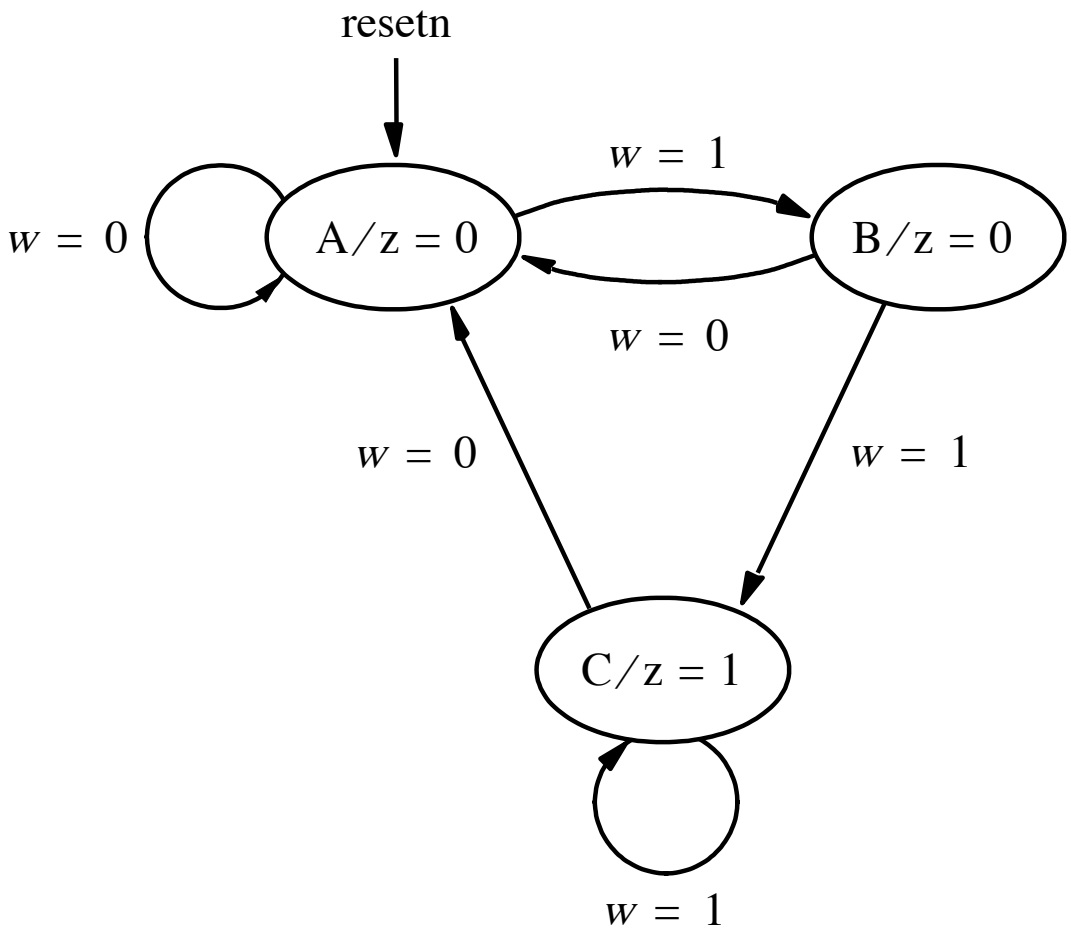
# Mealy FSM in VHDL (2)

```
            WHEN S1 =>
              IF input = '0' THEN
                Mealy_state <= S0;
              ELSE
                Mealy_state <= S1;
              END IF;
        END CASE;
    END IF;
END PROCESS;
```




```
Output <= '1' WHEN (Mealy_state = S1 AND input = '0') ELSE '0';
```

# Moore FSM – Example 2: State diagram

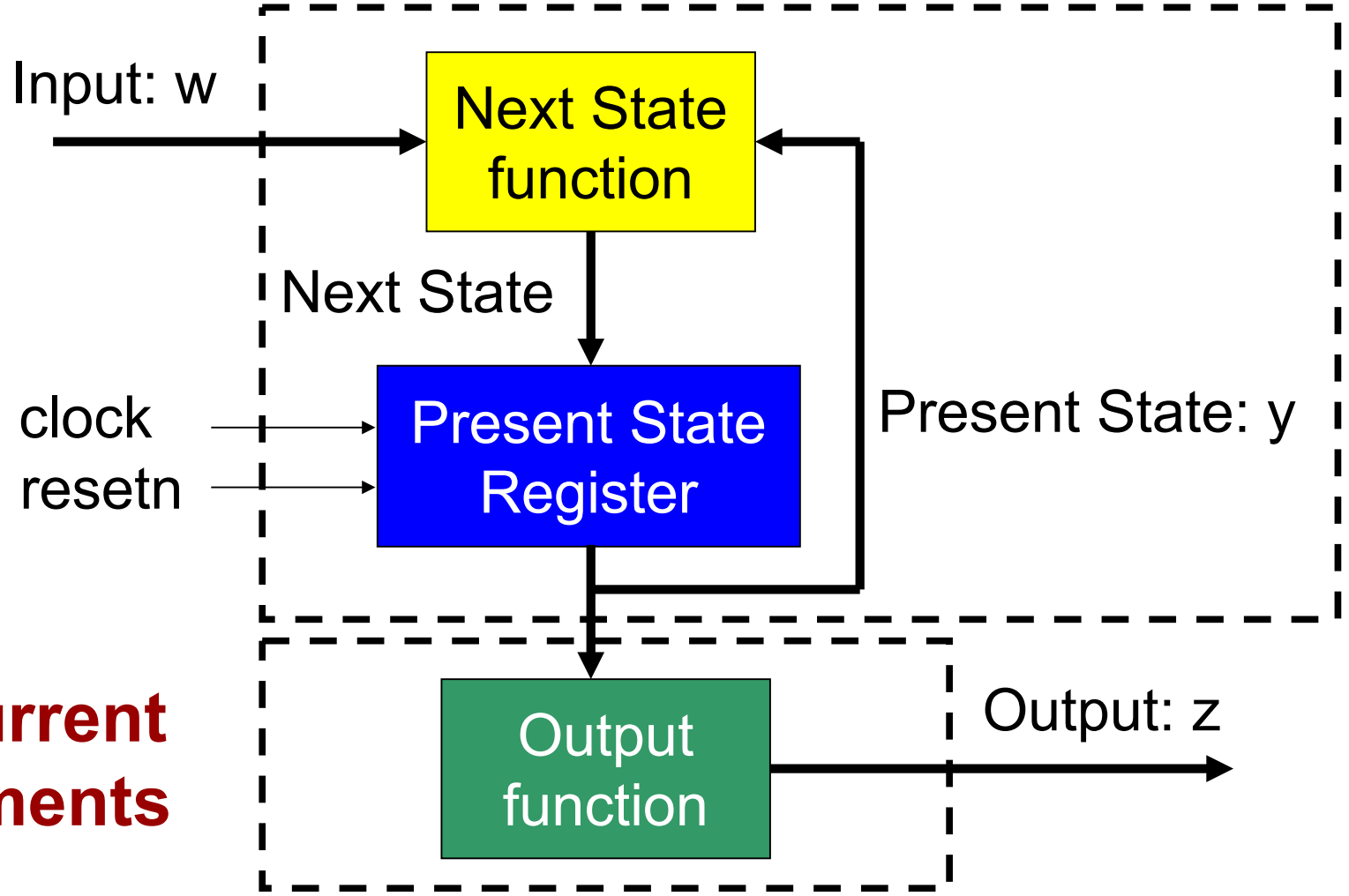


# Moore FSM – Example 2: State table

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

# Moore FSM

**process(clock, reset)**




**concurrent statements**

# Moore FSM – Example 2: VHDL code (1)

```
USE ieee.std_logic_1164.all ;

ENTITY simple IS
      PORT (  clock   : IN STD_LOGIC ;
              resetn  : IN STD_LOGIC ;
              w       : IN STD_LOGIC ;
              z       : OUT STD_LOGIC ) ;
END simple ;

ARCHITECTURE Behavior OF simple IS
      TYPE State_type IS (A, B, C) ;
      SIGNAL y : State_type ;
BEGIN
      PROCESS ( resetn, clock )
      BEGIN
            IF resetn = '0' THEN
                  y <= A ;
            ELSIF (Clock'EVENT AND Clock = '1') THEN
```

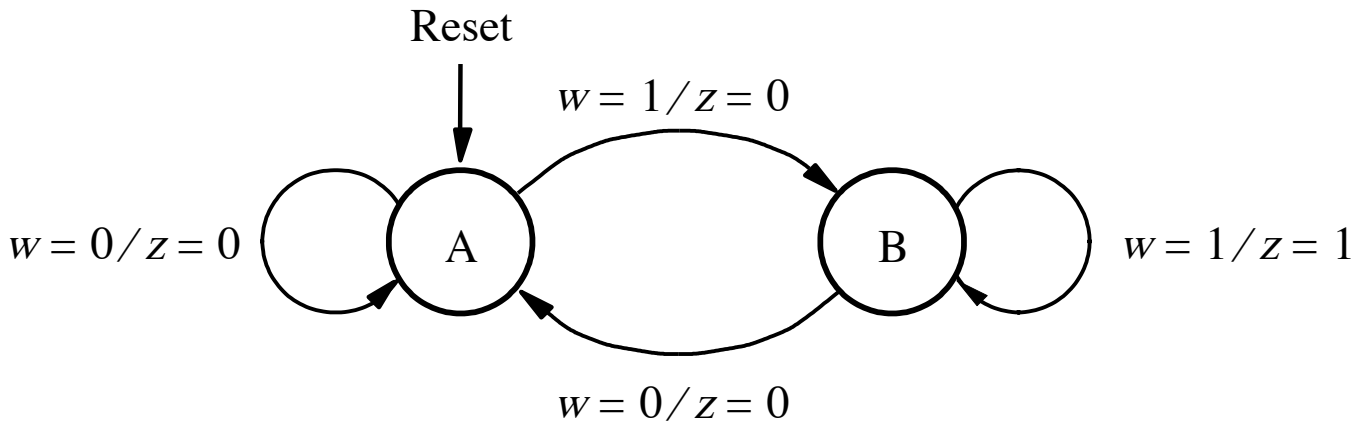
# Moore FSM – Example 2: VHDL code (2)

```
CASE y IS
        WHEN A =>
                IF w = '0' THEN
                        y <= A ;
                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 ;
```

# Moore FSM – Example 2: VHDL code (3)

```
        END IF ;
    END PROCESS ;

    z <= '1' WHEN y = C ELSE '0' ;

END Behavior ;
```

# Mealy FSM – Example 2: State diagram



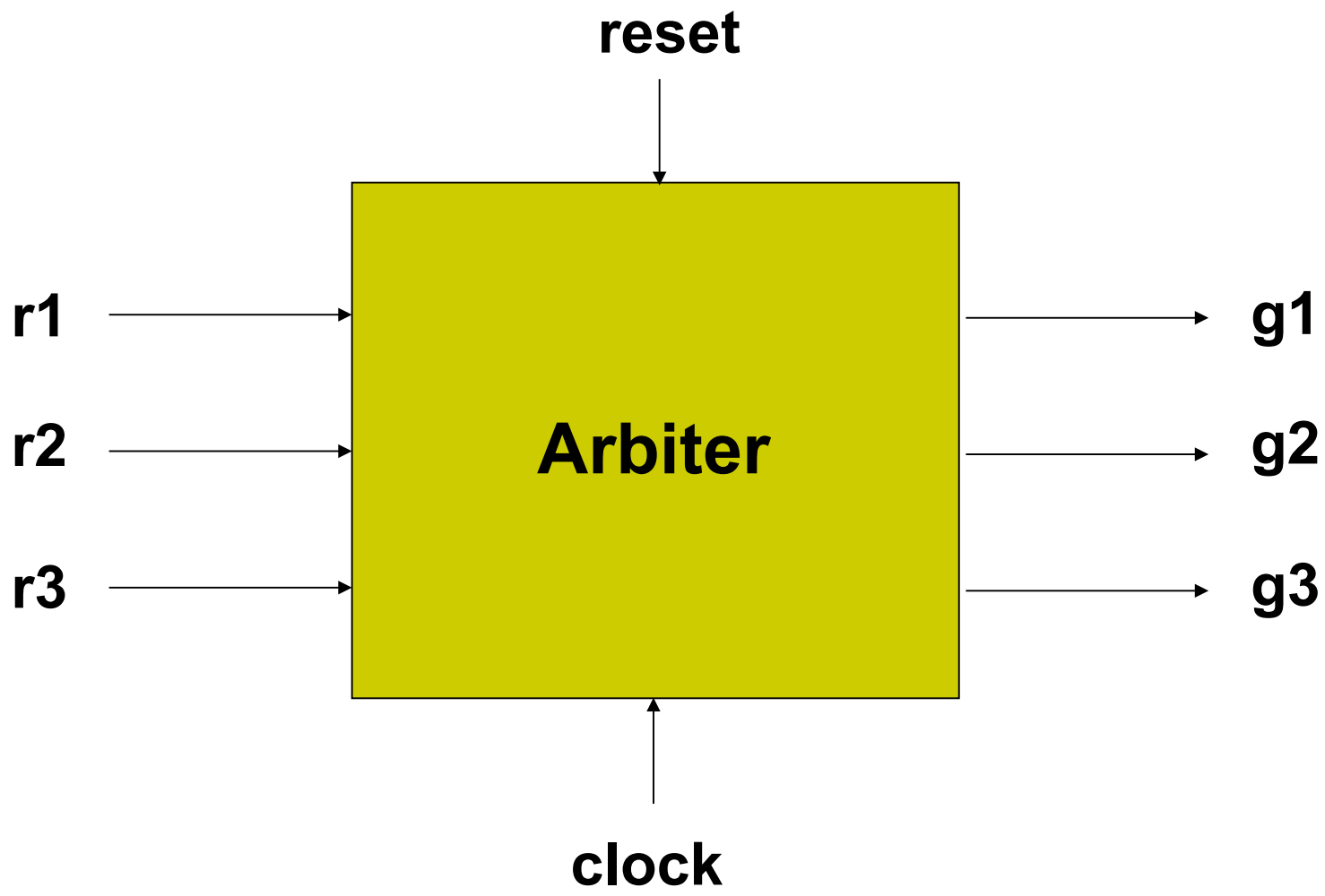
# Example 2: VHDL code (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY Mealy IS
        PORT (   clock     : IN          STD_LOGIC ;
                 resetn    : IN          STD_LOGIC ;
                 w         : IN          STD_LOGIC ;
                   z       : OUT     STD_LOGIC ) ;
END Mealy ;

ARCHITECTURE Behavior OF Mealy IS
        TYPE State_type IS (A, B) ;
        SIGNAL y : State_type ;
BEGIN
        PROCESS ( resetn, clock )
        BEGIN
                IF resetn = '0' THEN
                        y <= A ;
                ELSIF (clock'EVENT AND clock = '1') THEN
```

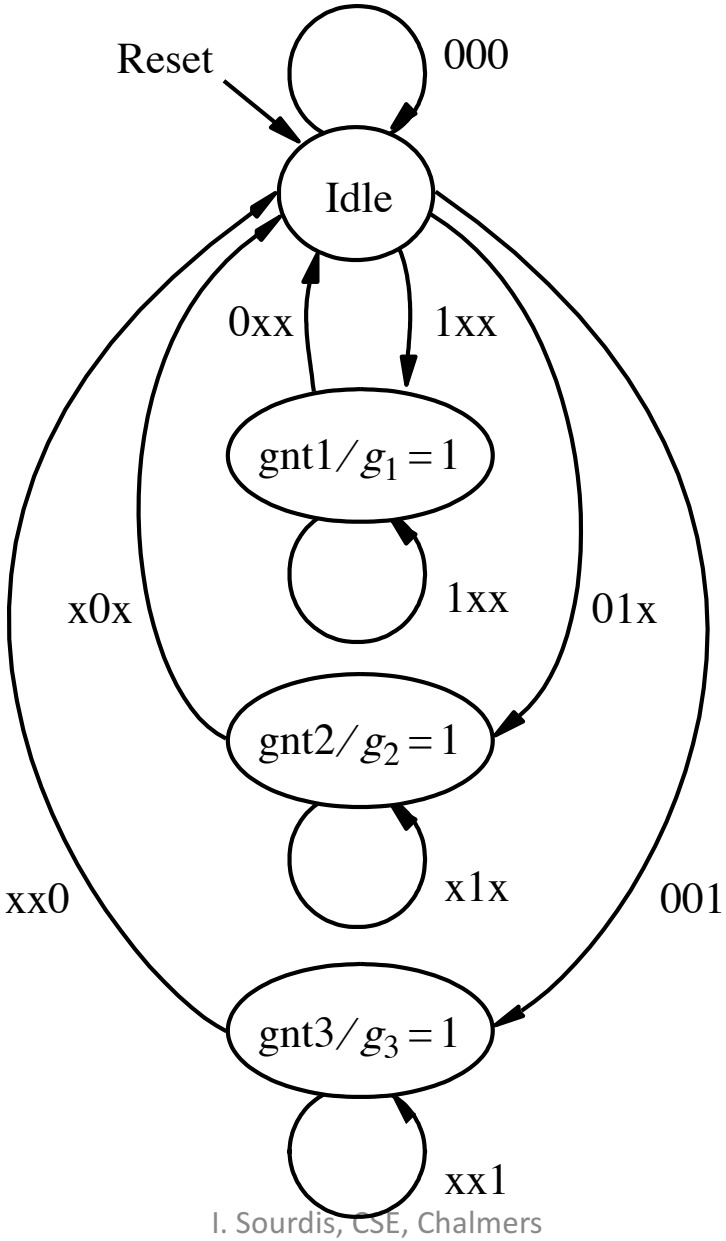
# Example 2: VHDL code (2)

```
            CASE y IS
                WHEN A =>
                    IF w = '0' THEN
                    y <= A ;
                        ELSE
                    y <= B ;
                        END IF ;
                WHEN B =>
                        IF w = '0' THEN
                    y <= A ;
                        ELSE
                    y <= B ;
                        END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;

    z <= '1' WHEN (y = B) AND (w='1') ELSE '0' ;

END Behavior ;
```

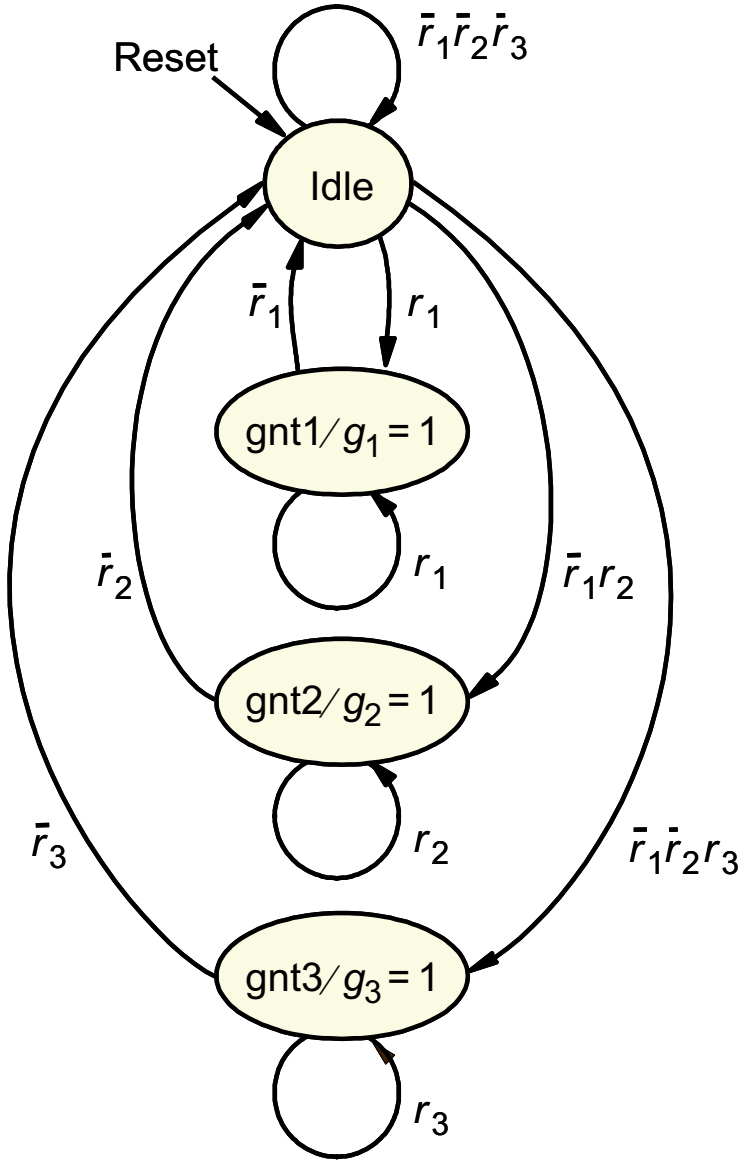
# Control Unit Example: Arbiter (1)



# Control Unit Example: Arbiter (2)



# Control Unit Example: Arbiter (3)



# Example 3: VHDL code (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY arbiter IS
   PORT ( Clock, Resetn   : IN     STD_LOGIC ;
                r         : IN     STD_LOGIC_VECTOR(1 TO 3) ;
                g         : OUT    STD_LOGIC_VECTOR(1 TO 3) ) ;
END arbiter ;

ARCHITECTURE Behavior OF arbiter IS
   TYPE State_type IS (Idle, gnt1, gnt2, gnt3) ;
   SIGNAL y : State_type ;
```

# Example 3: VHDL code (2)

```
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
            IF Resetn = '0' THEN y <= Idle ;
            ELSIF (Clock'EVENT AND Clock = '1') THEN
                    CASE y IS
                            WHEN Idle =>
                                    IF r(1) = '1' THEN y <= gnt1 ;
                                    ELSIF r(2) = '1' THEN y <= gnt2 ;
                                    ELSIF r(3) = '1' THEN y <= gnt3 ;
                                    ELSE y <= Idle ;
                                    END IF ;
                            WHEN gnt1 =>
                                    IF r(1) = '1' THEN y <= gnt1 ;
                                    ELSE y <= Idle ;
                                    END IF ;
                            WHEN gnt2 =>
                                    IF r(2) = '1' THEN y <= gnt2 ;
                                    ELSE y <= Idle ;
                                    END IF ;
```

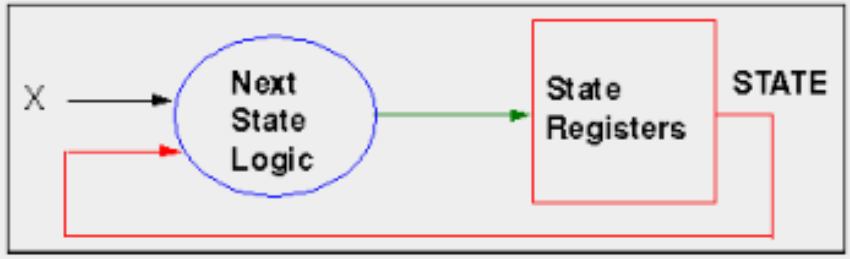
# Example 3: VHDL code (3)

```
                    WHEN gnt3 =>
                        IF r(3) = '1' THEN y <= gnt3 ;
                        ELSE y <= Idle ;
                        END IF ;
                END CASE ;
            END IF ;
        END PROCESS ;

        g(1) <= '1' WHEN y = gnt1 ELSE '0' ;
        g(2) <= '1' WHEN y = gnt2 ELSE '0' ;
        g(3) <= '1' WHEN y = gnt3 ELSE '0' ;
END Behavior ;
```

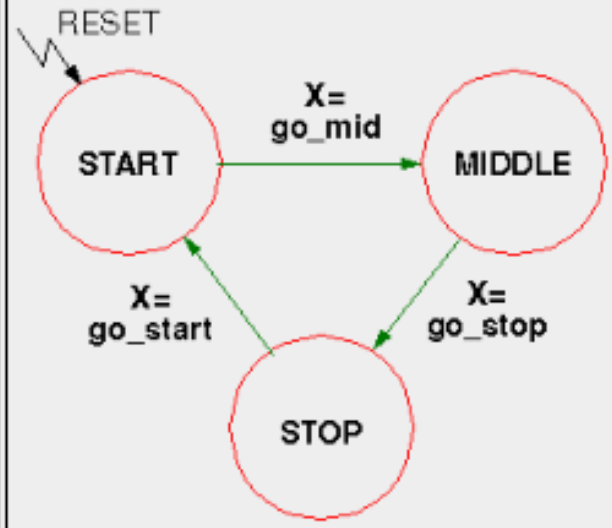
# FSMs with VHDL

- State Processes
- State Coding
- Additional ways to implement FSMs
  - Moore
  - Mealy
  - Registered Output

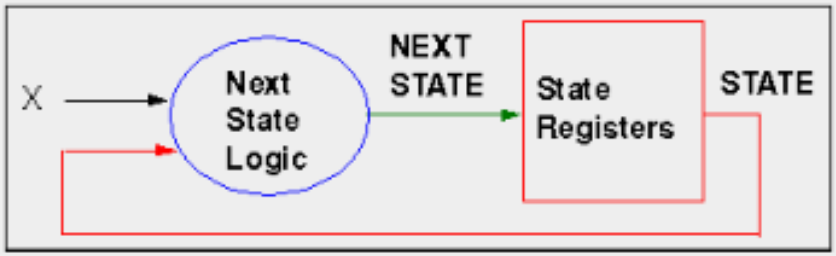
# One "State" Process



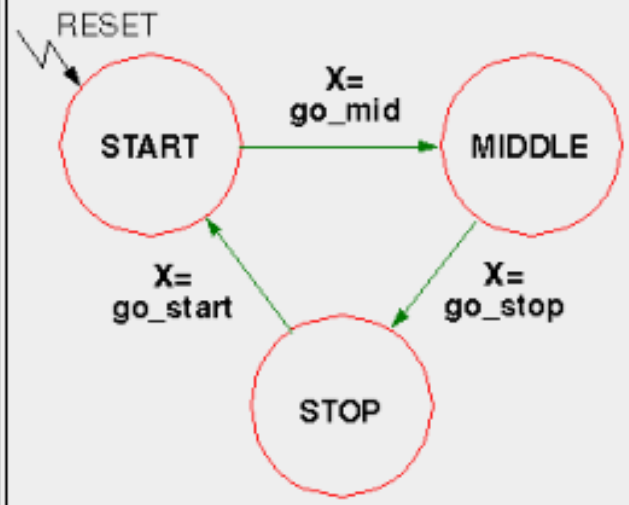

```
FSM_FF: process (CLK, RESET)
begin
   if RESET='1' then
      STATE <= START ;
   elsif CLK'event and CLK='1' then
      case STATE is
         when START  => if X=GO_MID then
                        STATE <= MIDDLE ;
                     end if ;
         when MIDDLE => if X=GO_STOP then
                        STATE <= STOP ;
                     end if ;
         when STOP   => if X=GO_START then
                        STATE <= START ;
                     end if ;
         when others => STATE <= START ;
      end case ;
   end if ;
end process FSM_FF ;
```



# Two "State" Processes




```
FSM_FF: process (CLK, RESET)    begin
   if RESET='1' then
       STATE <= START  ;
   elsif CLK'event and CLK='1' then
       STATE  <=  NEXT_STATE  ;
   end if;
end process FSM_FF ;

FSM_LOGIC: process ( STATE , X)
begin
    NEXT_STATE <= STATE ;
   case  STATE  is
        when  START   => if  X=GO_MID  then
                    NEXT_STATE <= MIDDLE  ;
                end if ;
        when  MIDDLE  => ...
        when  others  =>  NEXT_STATE <= START  ;
      end case ;
end process FSM_LOGIC ;
```



# How Many Processes?

- Structure and Readability
  - Asynchronous combinatoric ≠ synchronous storing elements
    => 2 processes
  - FSM states change with special input changes
    => 1 process more comprehensible
  - Graphical FSM (without output equations) resembles one state process
    => 1 process
- Simulation
  - Error detection easier with two state processes
    => 2 processes
- Synthesis
  - 2 state processes can lead to smaller generic netlist
    and therefore to better synthesis results
    => 2 processes

# State Encoding

- type STATE_TYPE is ( **START, MIDDLE, STOP** ) ;
  signal STATE : STATE_TYPE ;

- START -> " **00** "
  MIDDLE -> " **01** "
  STOP -> " **10** "

- START -> " **001** "
  MIDDLE -> " **010** "
  STOP -> " **100**

- State encoding responsible for safety of FSM

- Default encoding: **binary**

- Speed optimized default encoding: **one hot**

# Extension of Case Statement

- type STATE_TYPE is (START, MIDDLE, STOP) ;
  signal STATE : STATE_TYPE ;
  . . .

```
case STATE is
      when START    => · · ·
      when MIDDLE  => · · ·
      when STOP     => · · ·

      when others     => · · ·

end case ;
```

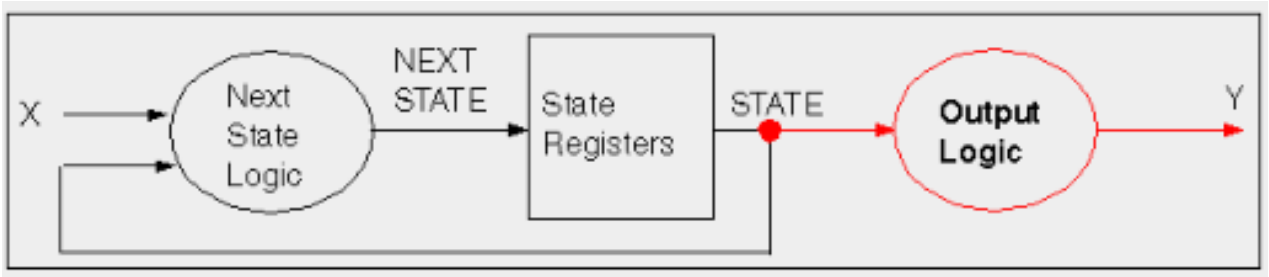
- Adding the "when others" choice

# Hand Coding

```
subtype STATE_TYPE is
                std_logic_vector(1 downto 0) ;

  signal STATE : STATE_TYPE ;
  constant START  : STATE_TYPE := "01";
  constant MIDDLE : STATE_TYPE := "11";
  constant STOP   : STATE_TYPE := "00";
  …
    case STATE is
        when START   => …
        when MIDDLE  => …
        when STOP    => …
        when others  => …
    end case ;
```

- Defining constants
- Control of encoding
- Safe FSM
- Simulatable
- Portable design
- More effort

# FSM: Moore




**The output vector is a function of the state vector: Y = f(S)**

- **Three Processes**

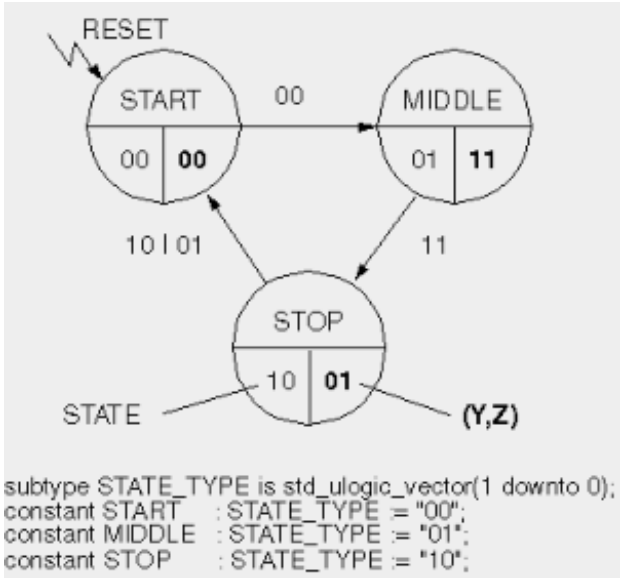
```
architecture RTL of MOORE is
    ...
begin
    REG: -- Clocked Process
    CMB: -- Combinational Process

    OUTPUT: process (STATE)
    begin
       -- Output Logic
    end process OUTPUT ;

end RTL ;
```

- **Two Processes**

```
architecture RTL of MOORE is
    ...
begin
    REG: process (CLK, RESET)
    begin
       -- State Registers Inference with
Next State Logic
    end process REG ;

    OUTPUT: process (STATE)
    begin
       -- Output Logic
    end process OUTPUT ;

end RTL ;
```

# Moore: 3 Process Example

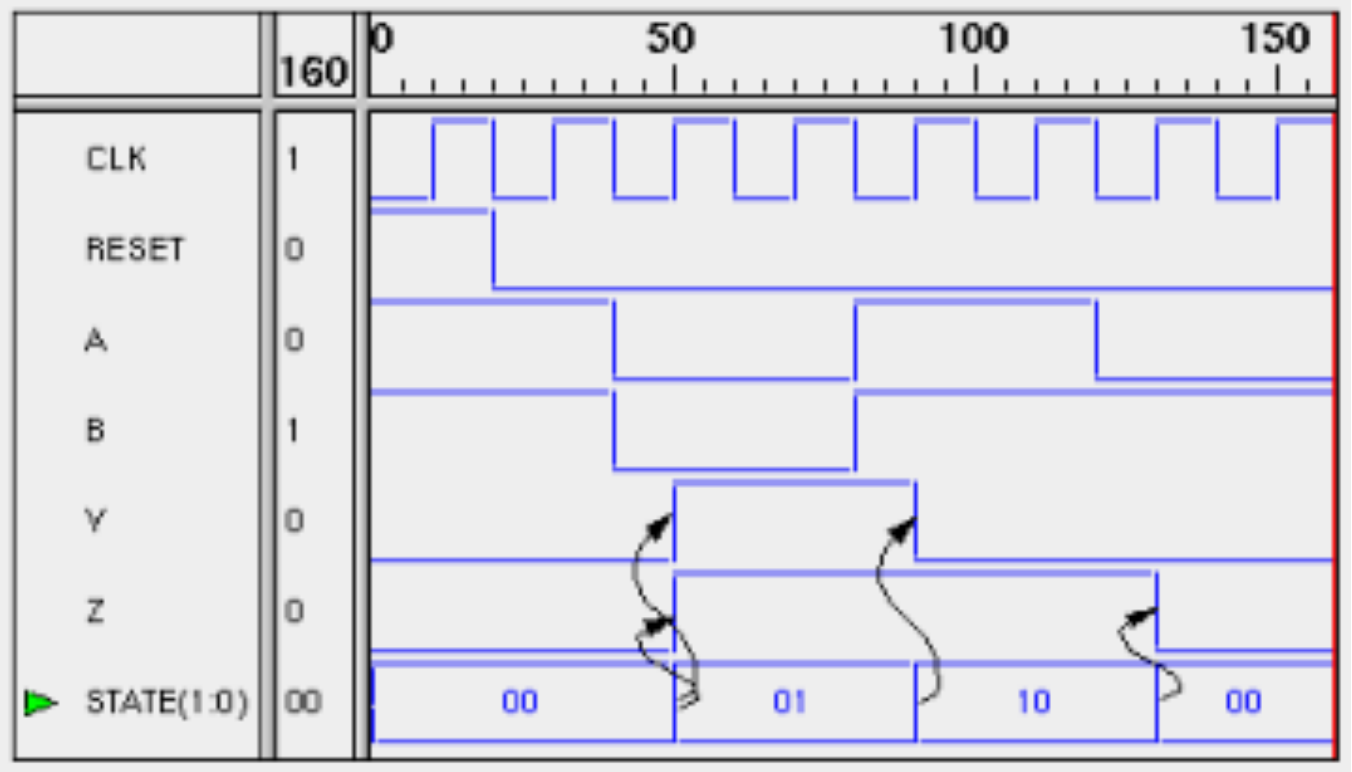
```
architecture RTL of MOORE_TEST is
      signal STATE,NEXTSTATE : STATE_TYPE ;
   begin

     REG: process (CLK, RESET) begin
       if RESET='1' then    STATE <= START ;
       elsif CLK`event and CLK='1' then
          STATE <= NEXTSTATE ;
       end if ;
    end process REG ;
```



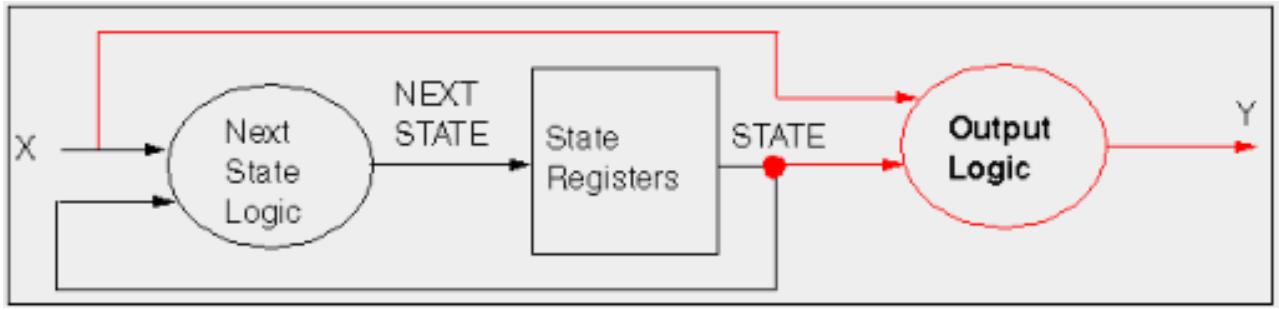

```
subtype STATE_TYPE is std_ulogic_vector(1 downto 0);
constant START   : STATE_TYPE := "00";
constant MIDDLE  : STATE_TYPE := "01";
constant STOP    : STATE_TYPE := "10";
```

```
  CMB: process (A,B,STATE) begin
     NEXT_STATE <= STATE;
     case STATE is
      when START => if (A or B)='0' then
                          NEXTSTATE <= MIDDLE ;
                end if ;
      when MIDDLE => if (A and B)='1' then
           NEXTSTATE <= STOP ;
                end if ;
      when STOP    => if (A xor B)='1' then
           NEXTSTATE <= START ;
                end if ;
      when others => NEXTSTATE <= START ;
     end case ;
  end process CMB ;

    -- concurrent signal assignments for output
    Y <= '1' when STATE=MIDDLE else '0' ;
    Z <= '1' when STATE=MIDDLE or STATE=STOP else '0' ;
end RTL ;
```

# Waveform Moore Example




- (Y,Z) changes simultaneously with STATE => Moore machine




**The output vector is a function of the state vector and the input vector:Y =f(X,S)**

- **Three Processes**

```
architecture RTL of MEALY is
    ...
begin
    REG: -- Clocked Process

    CMB: -- Combinational Process

    OUTPUT: process (STATE, X)
    begin
        -- Output Logic
    end process OUTPUT ;
end RTL ;
```

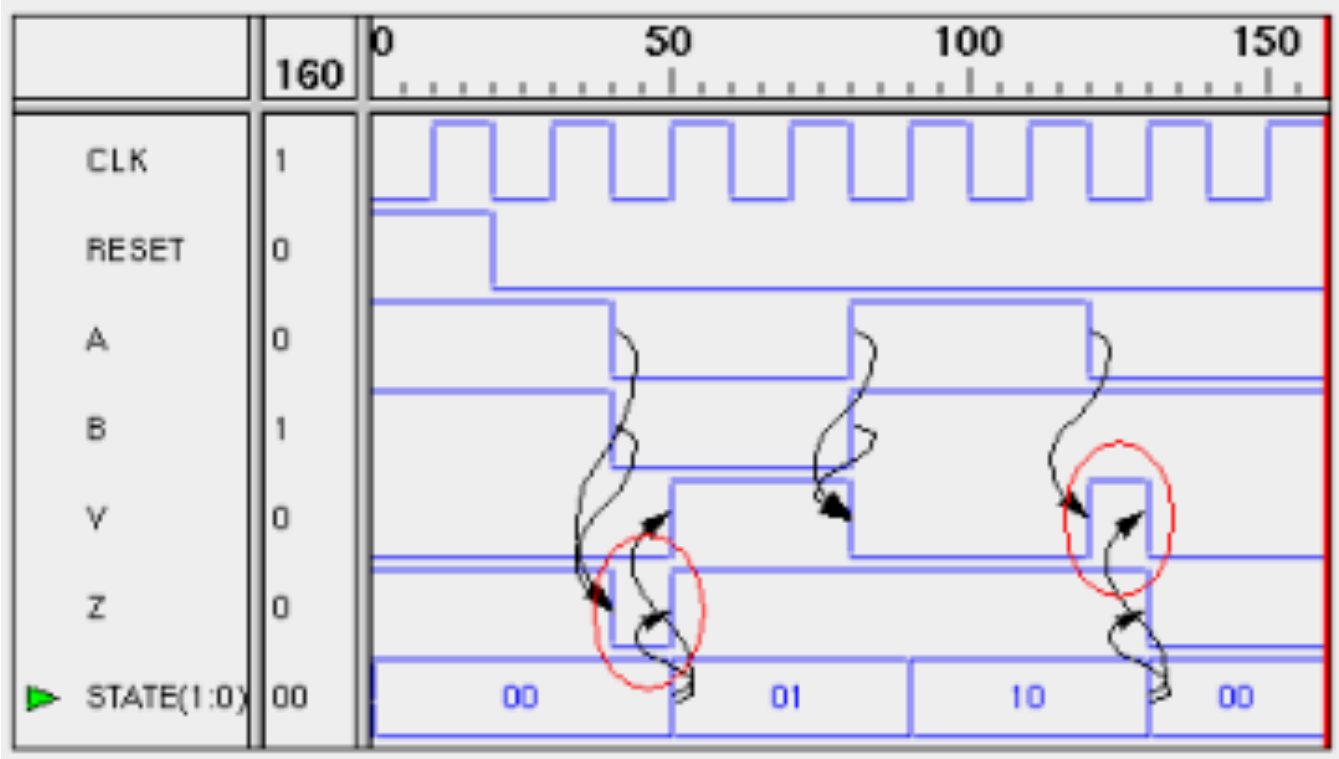
- **Two Processes**

```
architecture RTL of MEALY is
    ...
begin
    MED: process (CLK, RESET)
    begin
        -- State Registers Inference with
Next State Logic
    end process MED ;

    OUTPUT: process (STATE, X)
    begin
        -- Output Logic
    end process OUTPUT ;

end RTL ;
```

# Mealy Example

```
architecture RTL of MEALY_TEST is
        signal STATE,NEXTSTATE : STATE_TYPE ;
    begin
        REG: · · ·   -- clocked STATE process
        CMB: · · ·   -- Like Moore Examples

        OUTPUT: process (STATE, A, B)
        begin
          case STATE is
            when START   =>
                              Y <= '0' ;
                              Z <= A and B ;
            when MIDLLE  =>
                              Y <= A nor B ;
                              Z <= '1' ;
            when STOP    =>
                              Y <= A nand B ;
                              Z <= A or B ;
            when others  =>
                              Y <= '0' ;
                              Z <= '0' ;
          end case;
        end process OUTPUT;
    end RTL ;
```

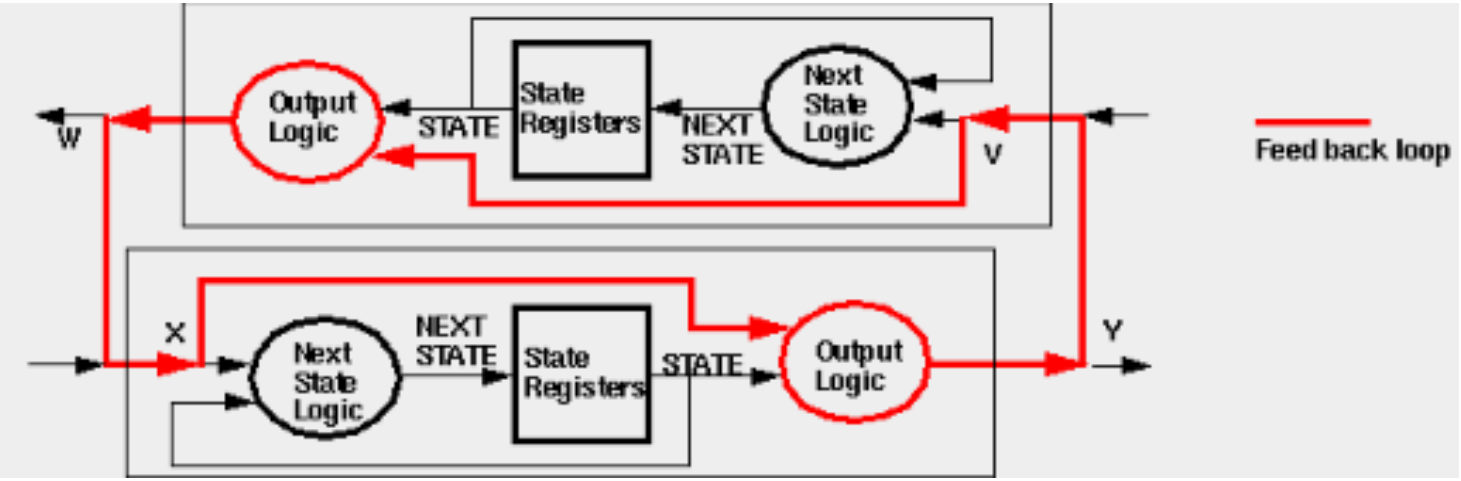
# Waveform Moore Example




- (Y,Z) changes with input => Mealy machine
- Note the "spikes" of Y and Z in the waveform
  - FSM has to be modeled carefully in order to avoid spikes in normal operation.

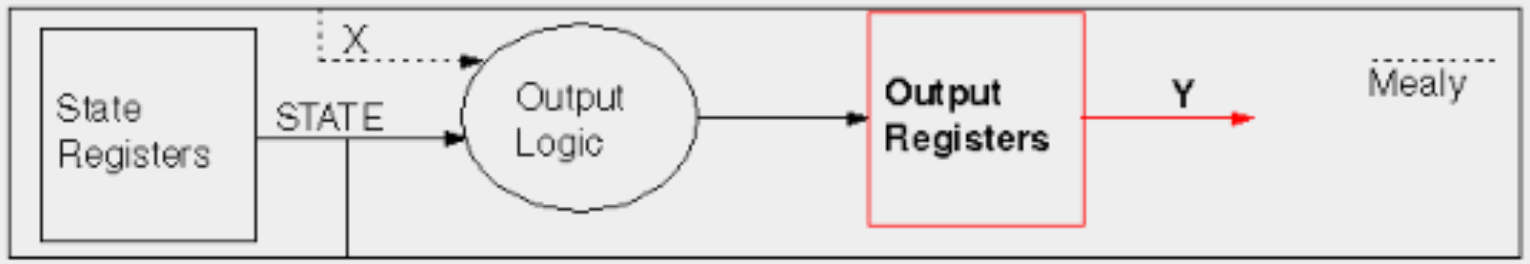
# Modelling Aspects

- Moore is preferred because of safe operation
- Mealy more flexible, but danger of
  - Spikes
  - Unnecessary long paths (maximum clock period)
  - Combinational feed back loops

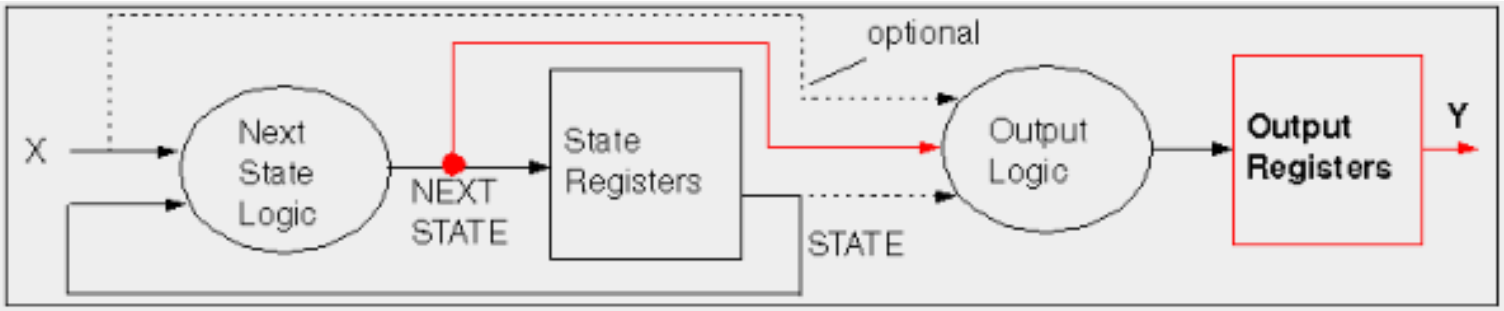


# Registered Output

- Avoiding long paths and uncertain timing
- With one additional clock period

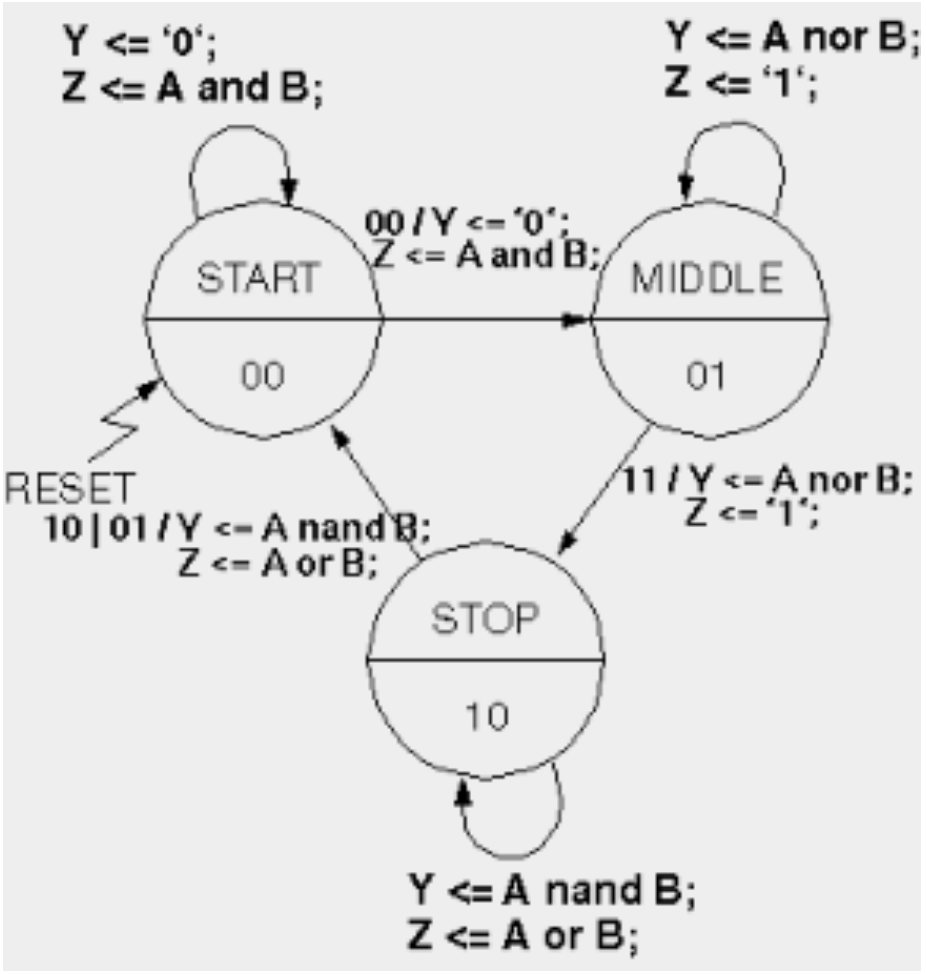



- Without additional clock period (Mealy)

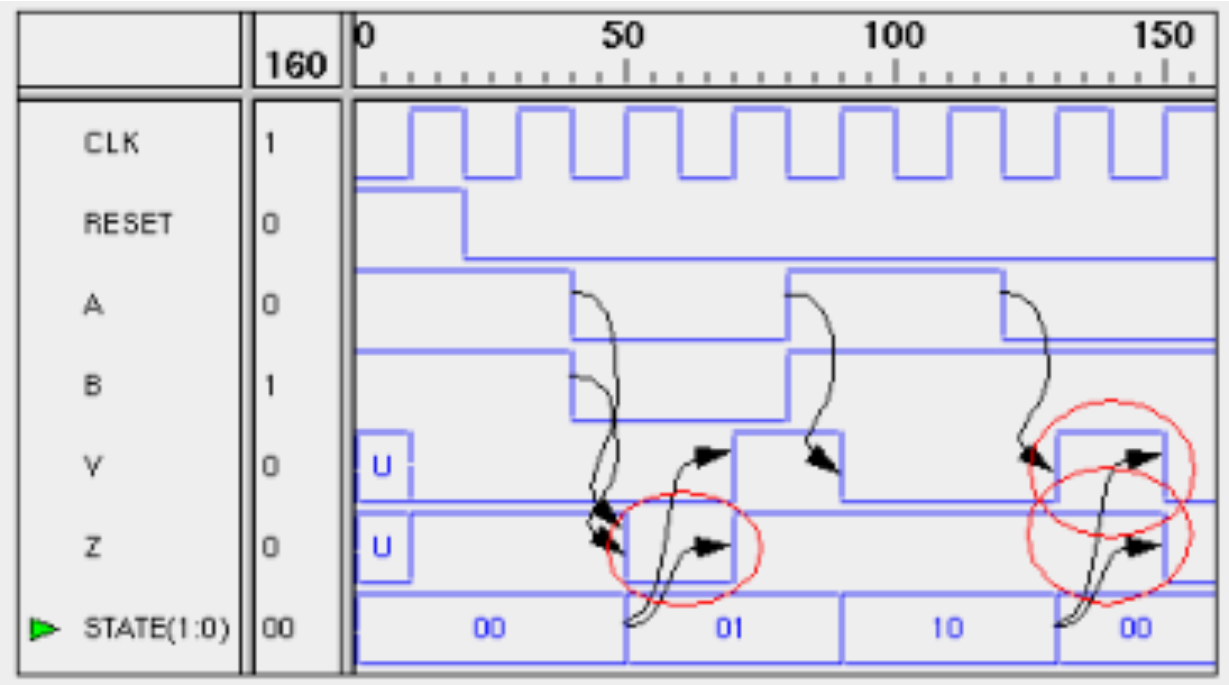


# Registered Output Example (1)

```
architecture RTL of REG_TEST is
        signal Y_I , Z_I : std_ulogic ;
        signal STATE,NEXTSTATE : STATE_TYPE ;
      begin

        REG: · · ·   -- clocked STATE process

        CMB: · · ·   -- Like other Examples

        OUTPUT: process (STATE, A, B)
        begin
          case STATE is
            when START   =>
                                Y_I<= `0` ;
                                Z_I<= A and B ;
            . . .
        end process OUTPUT


       -- clocked output process
        OUTPUT_REG: process(CLK)    begin
          if CLK'event and CLK='1' then
           Y <= Y_I ;
           Z <= Z_I ;
          end if ;
        end process OUTPUT_REG ;
      end RTL ;
```



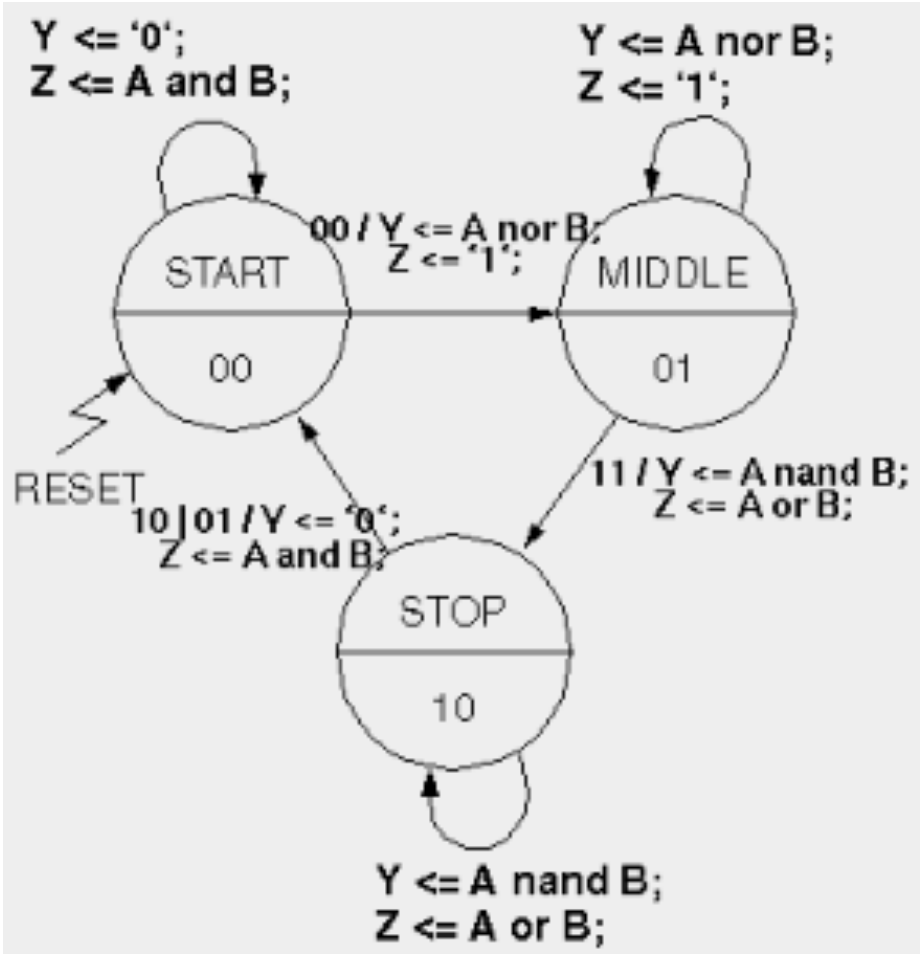
# Waveform Registered Output Example (1)



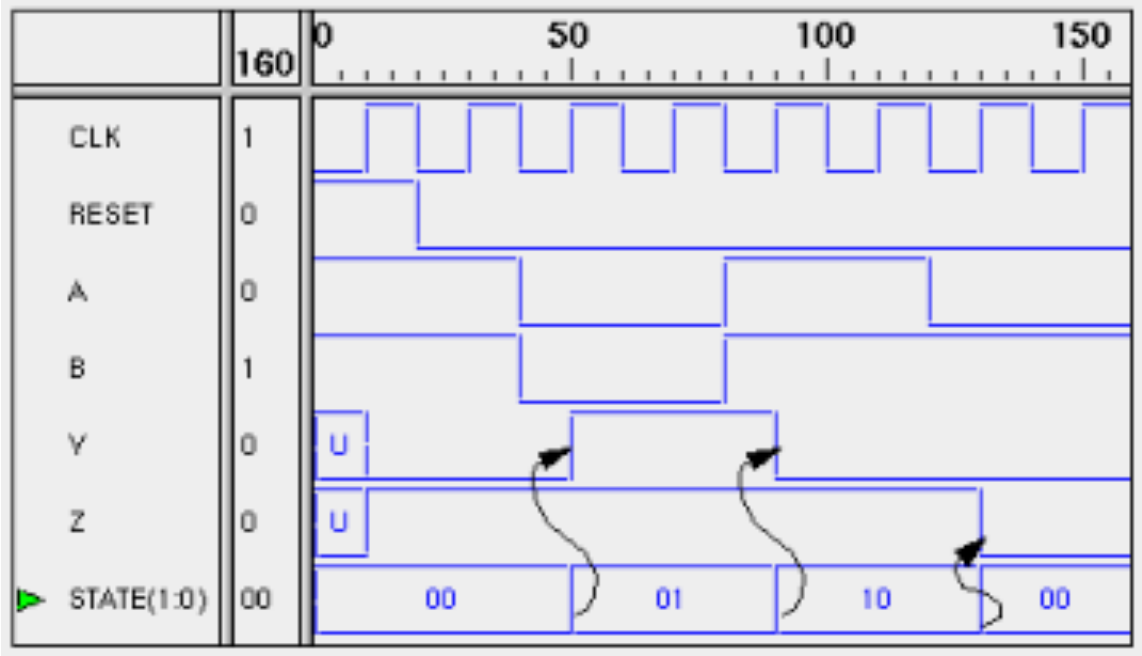

- One clock period delay between STATE and output changes.
- Input changes with clock edge result in an output change. (Danger of unmeant values)

# Registered Output Example (2)

```
architecture RTL of REG_TEST2 is
        signal Y_I , Z_I : std_ulogic ;
        signal STATE,NEXTSTATE : STATE_TYPE ;
    begin

        REG: · · ·   -- clocked STATE process

        CMB: · · ·   -- Like other Examples

        OUTPUT: process ( NEXTSTATE , A, B)
        begin
          case NEXTSTATE is
            when START   =>
                                Y_I<= `0` ;
                                Z_I<= A and B ;
            . . .
        end process OUTPUT

        OUTPUT_REG: process(CLK)
        begin
          if CLK'event and CLK='1' then
           Y <= Y_I ;
           Z <= Z_I ;
          end if ;
        end process OUTPUT_REG ;
    end RTL ;
```



# Waveform Registered Output Example (2)




- No delay between STATE and output changes.
- "Spikes" of original Mealy machine are gone!

# Summary of Lecture 9

- Control vs. Datapath
- More FSM examples
- FSMs in VHDL
  - Moore and Mealy FSMs in VHDL
  - State processes
  - State Coding
  - More VHDL alternatives (Mealy, Moore, registered output)

- Book (complimentary to the slides):
  - Sections 14.6

- Next Lecture 10:
  - Testbenches VHDL