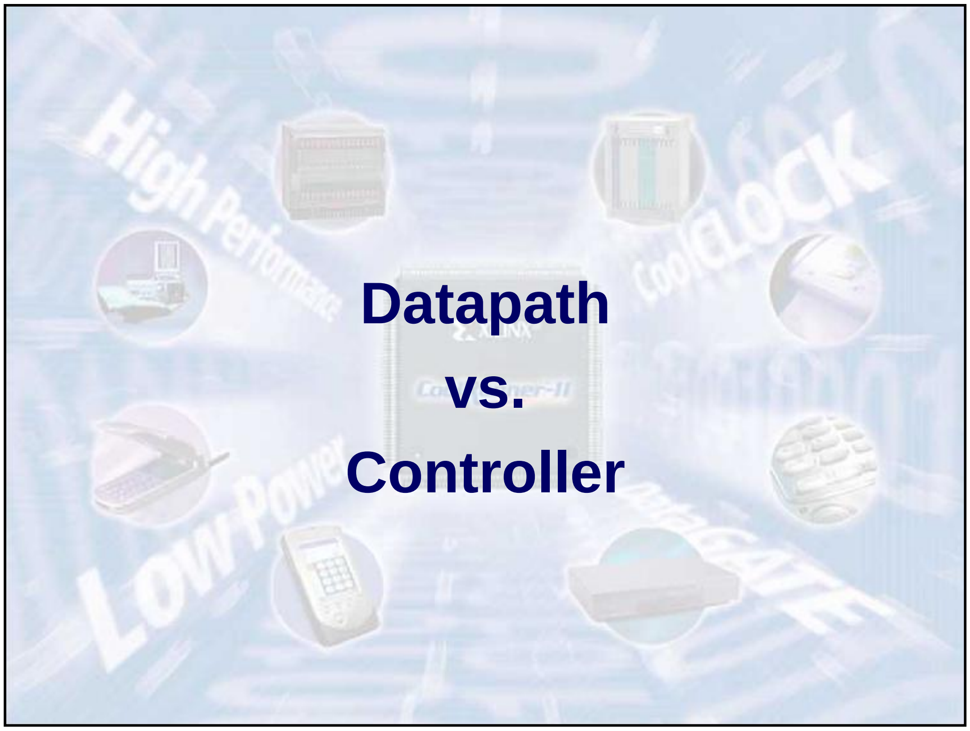
Design of Controllers

Finite State Machines

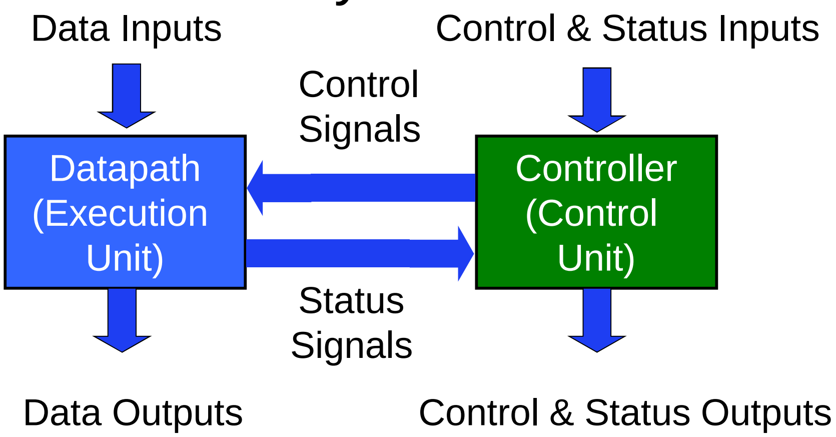


Datapath

vs.

Controller

Structure of a Typical Digital System



Data Inputs

Control & Status Inputs

Control Signals

Datapath (Execution Unit)

Controller (Control Unit)

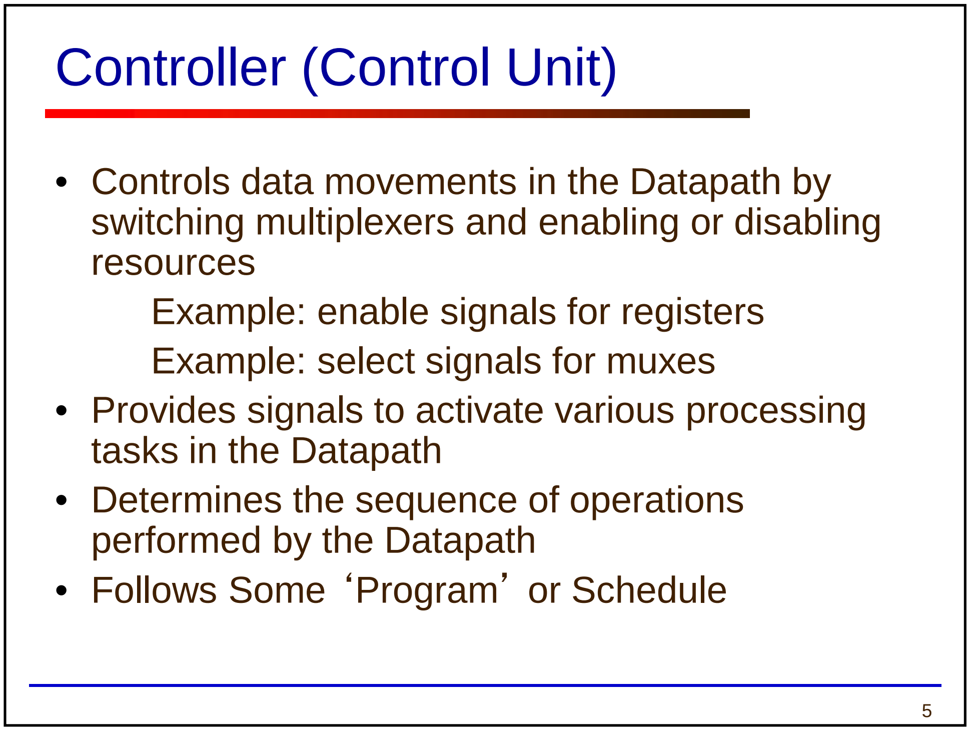
Status Signals

Data Outputs

Control & Status Outputs

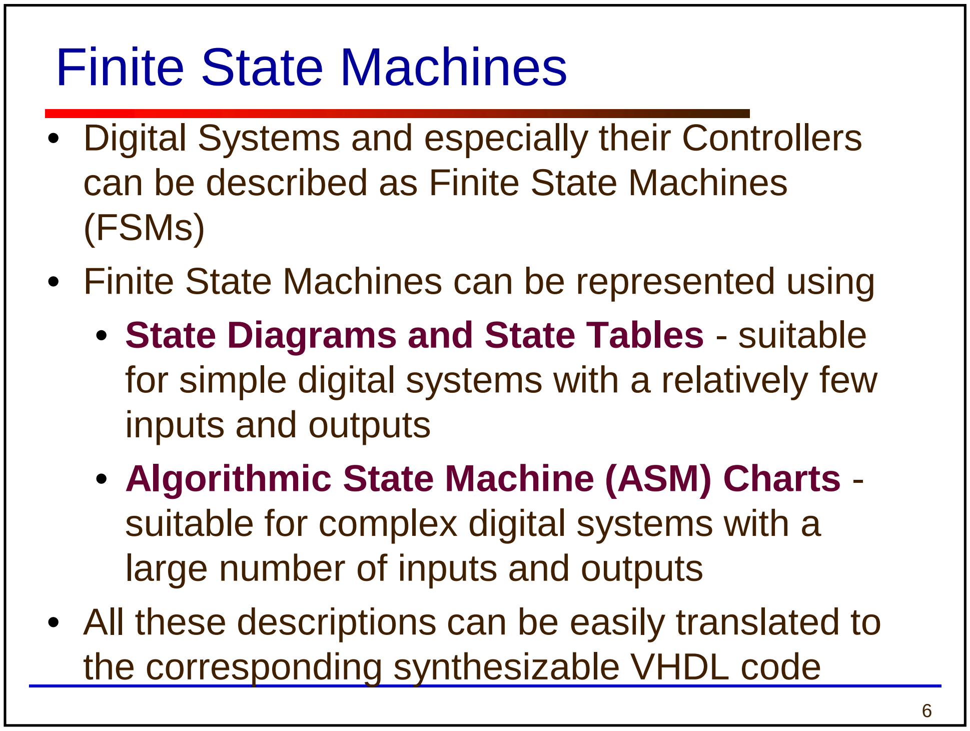
Datapath (Execution Unit)

- Manipulates and processes data
- Performs arithmetic and logic operations, shifting/rotating, and other data-processing tasks
- Is composed of registers, multiplexers, adders, decoders, comparators, ALUs, gates, 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)

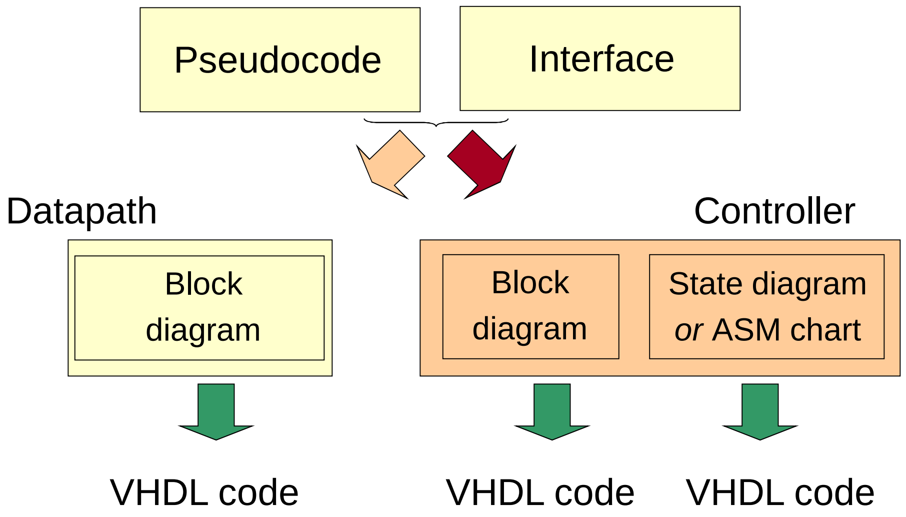
- Controls data movements in the Datapath by switching multiplexers and enabling or disabling resources
  - Example: enable signals for registers
  - Example: select signals for muxes
- Provides signals to activate various processing tasks in the Datapath
- Determines the sequence of operations performed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



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
- All these descriptions can be easily translated to the corresponding synthesizable VHDL code

Hardware Design with RTL VHDL



Pseudocode

Interface

Datapath

Controller

Block diagram

Block diagram

State diagram *or* ASM chart

VHDL code

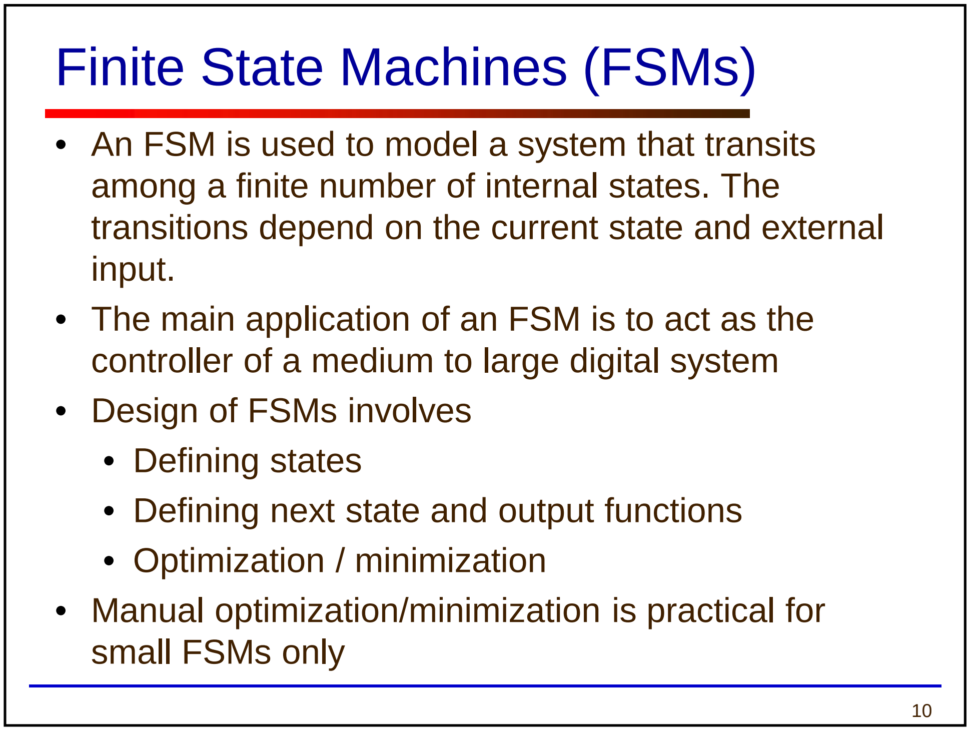
VHDL code

VHDL code

Steps of the Design Process

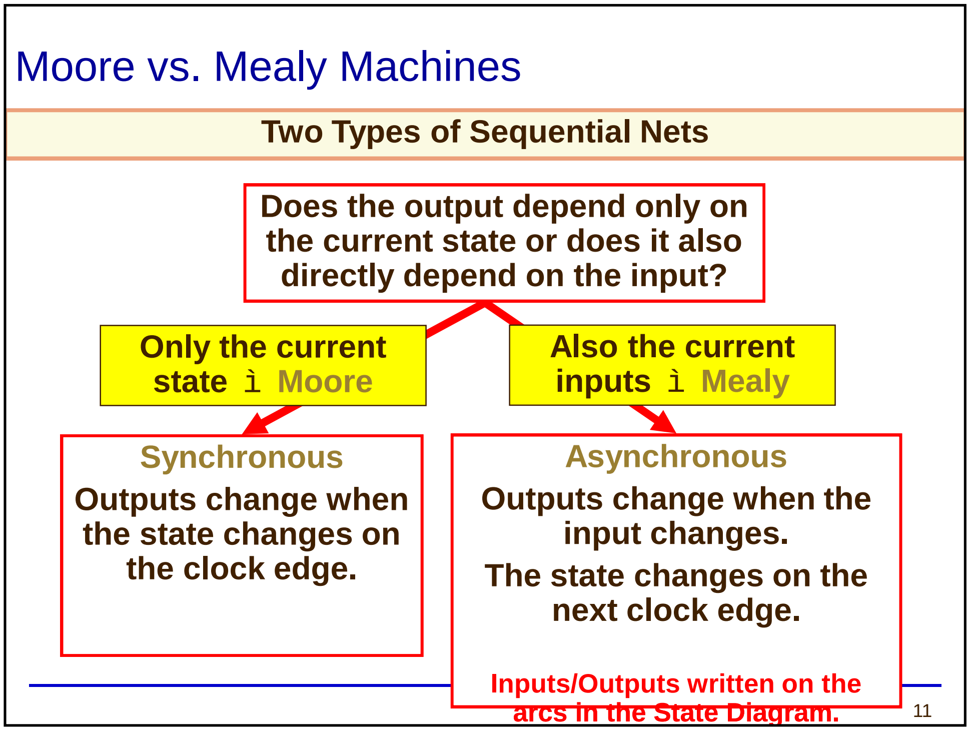
1. Text description
2. Interface
3. Pseudocode
4. Block diagram of the Datapath
5. Interface with the division into Datapath and Controller
6. ASM chart of the Controller
7. RTL VHDL code of the Datapath, Controller, and Top-Level Unit
8. Testbench of the Datapath, Controller, and Top-Level Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing

Finite State Machines Refresher



Finite State Machines (FSMs)

- An FSM is used to model a system that transits among a finite number of internal states. The transitions depend on the current state and external input.
- The main application of an FSM is to act as the controller of a medium to large digital system
- Design of FSMs involves
  - Defining states
  - Defining next state and output functions
  - Optimization / minimization
- Manual optimization/minimization is practical for small FSMs only



Moore vs. Mealy Machines

Two Types of Sequential Nets

Does the output depend only on the current state or does it also directly depend on the input?

Only the current state ì Moore

Also the current inputs ì Mealy

Synchronous

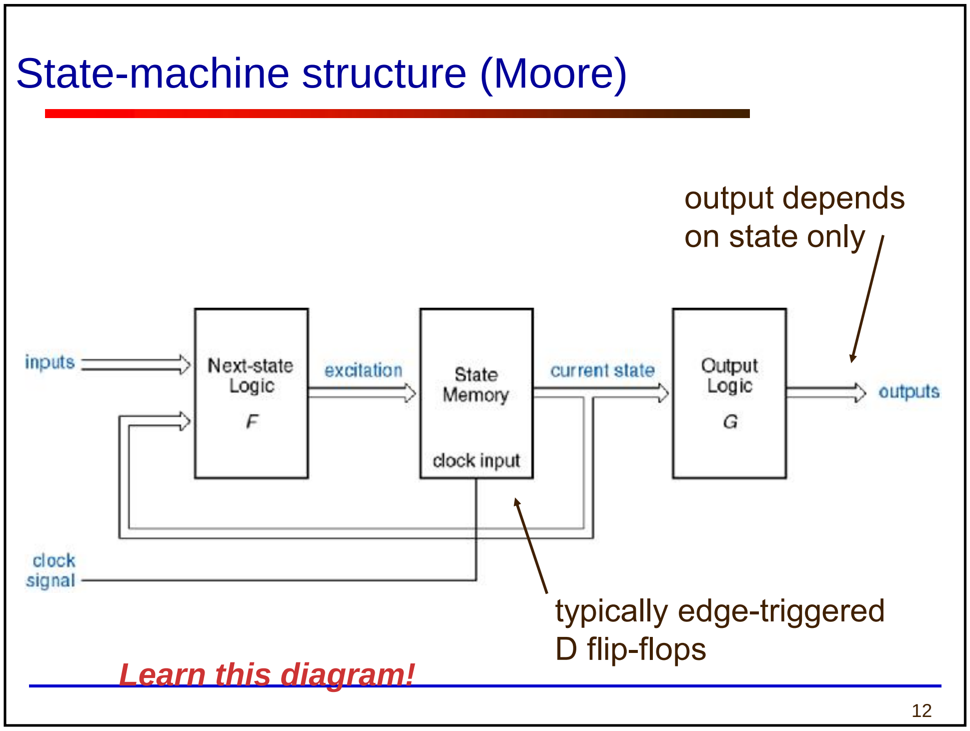
Outputs change when the state changes on the clock edge.

Asynchronous

Outputs change when the input changes.

The state changes on the next clock edge.

Inputs/Outputs written on the arcs in the State Diagram.



State-machine structure (Moore)

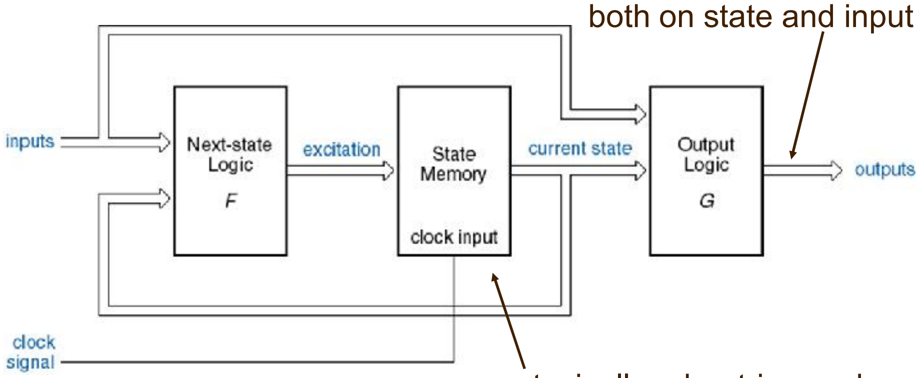
output depends on state only

typically edge-triggered D flip-flops

Learn this diagram!

State-machine structure (Mealy)

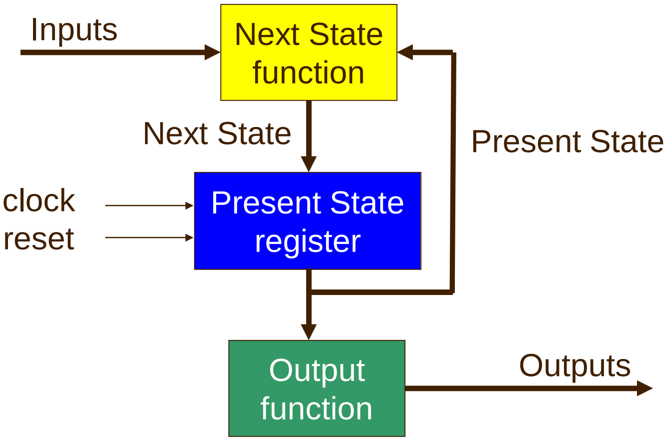
output depends directly both on state and input

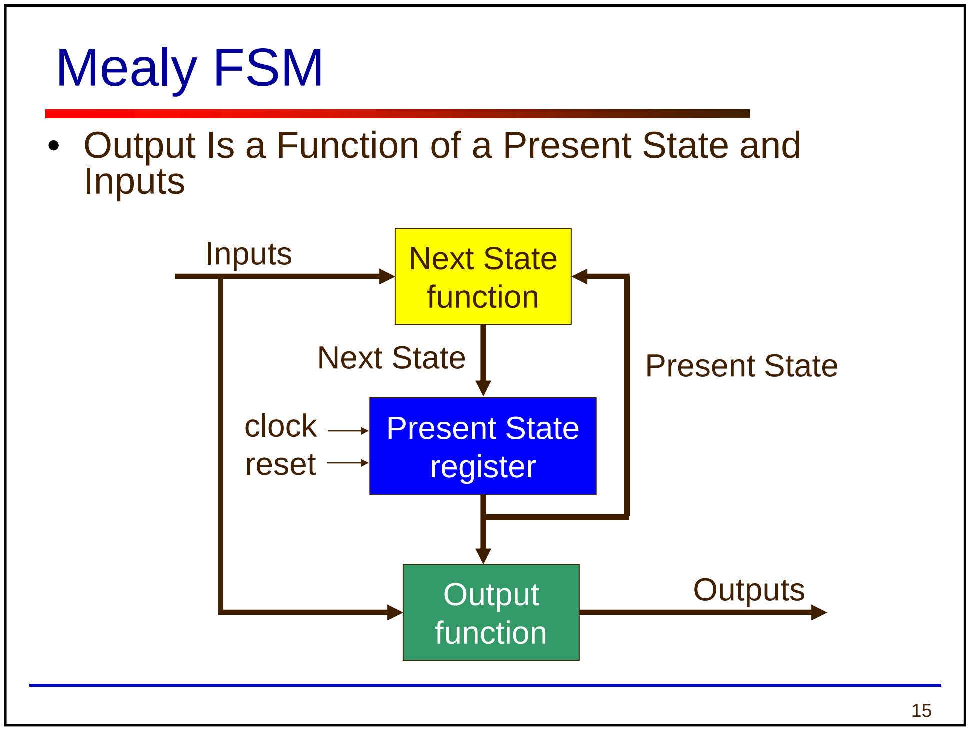


typically edge-triggered DFFs

Moore FSM

- Output Is a Function of a Present State Only





Mealy FSM

- Output Is a Function of a Present State and Inputs

Inputs

Next State function

Next State

Present State

clock

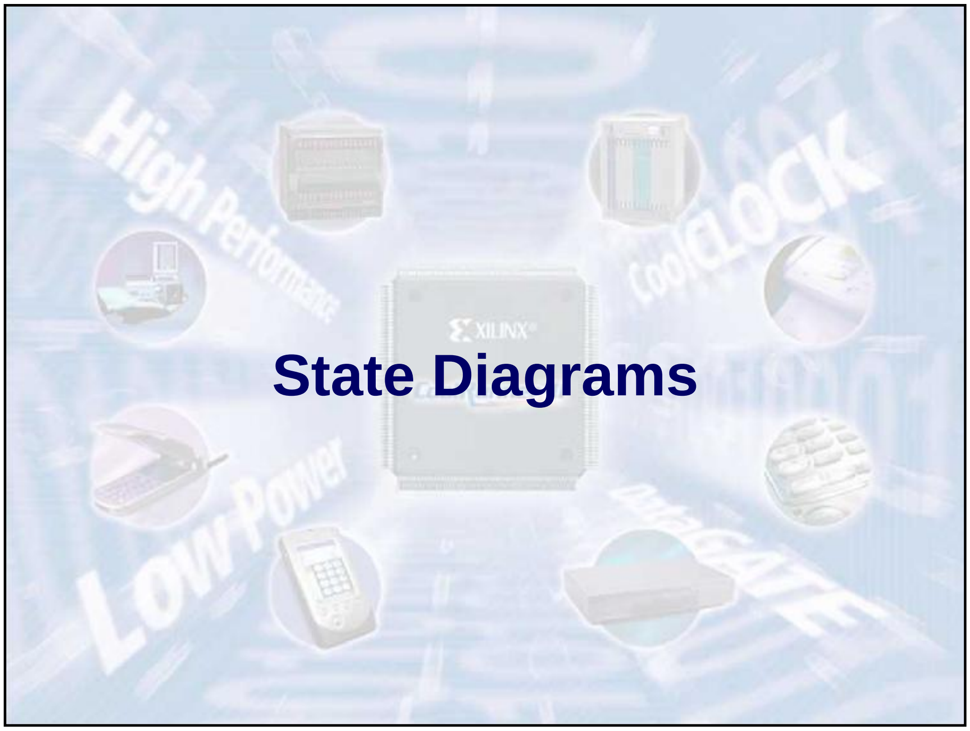
reset

Present State register

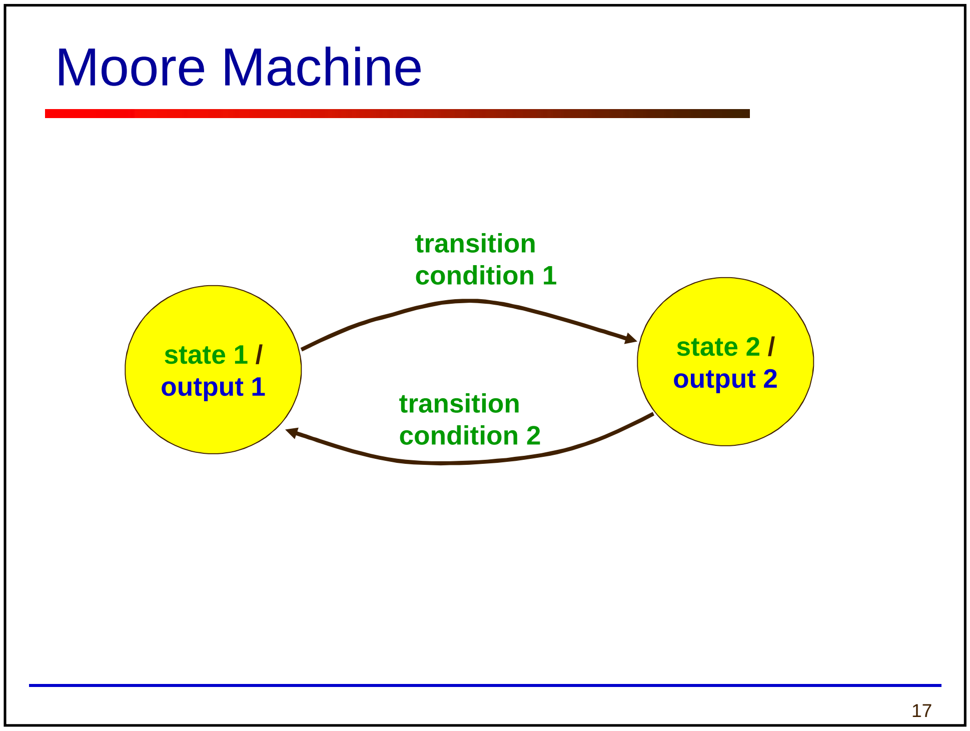
Output function

Outputs

15



State Diagrams



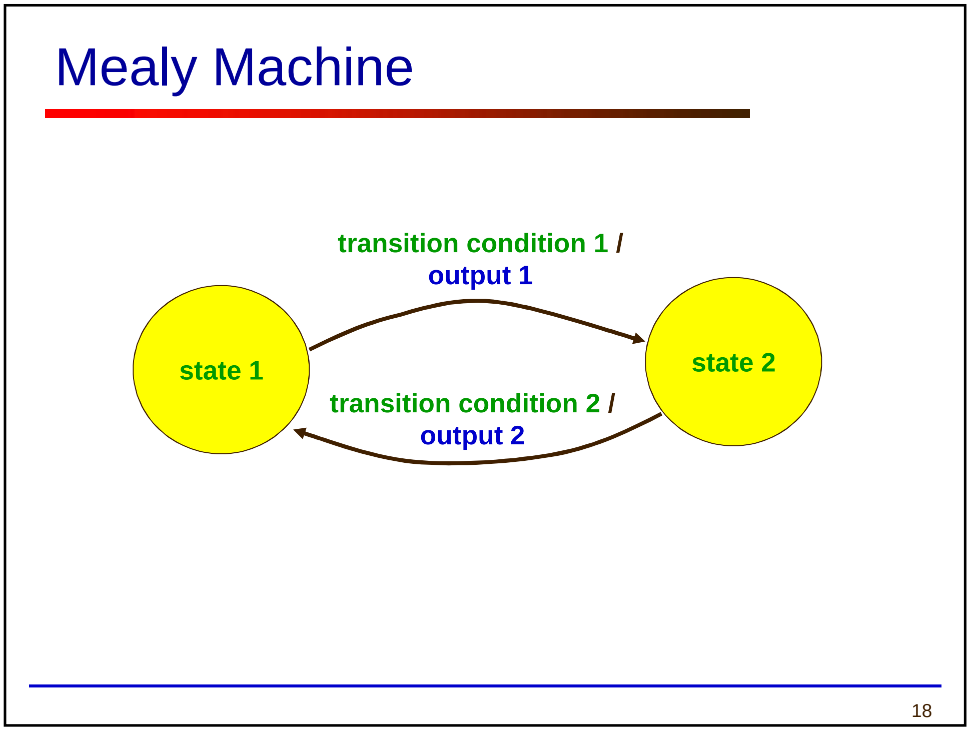
Moore Machine

transition condition 1

state 1 / output 1

state 2 / output 2

transition condition 2



Mealy Machine

transition condition 1 / output 1

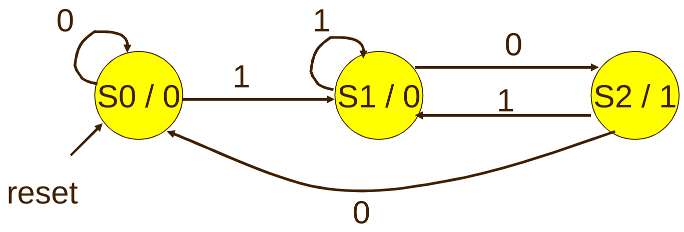
state 1

state 2

transition condition 2 / output 2

Moore FSM - Example 1

- Moore FSM that Recognizes Sequence “10”



Meaning of states:

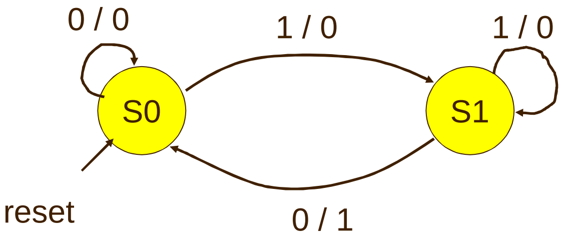
S0: No elements of the sequence observed

S1: “1” observed

S2: “10” observed

Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence “10”

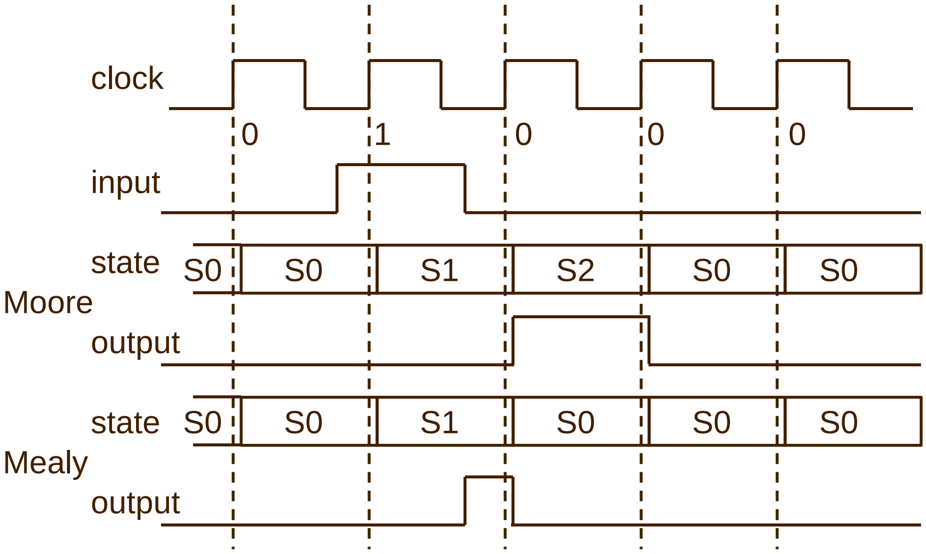


Meaning of states:

S0: No elements of the sequence observed

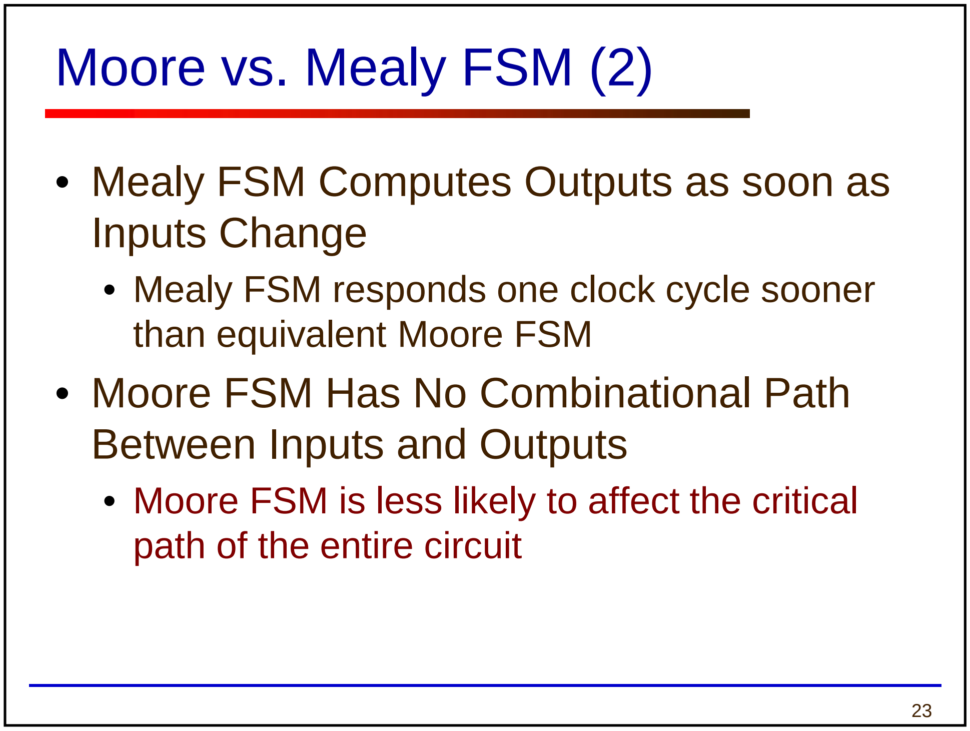
S1: “1” observed

Moore & Mealy FSMs – Example 1



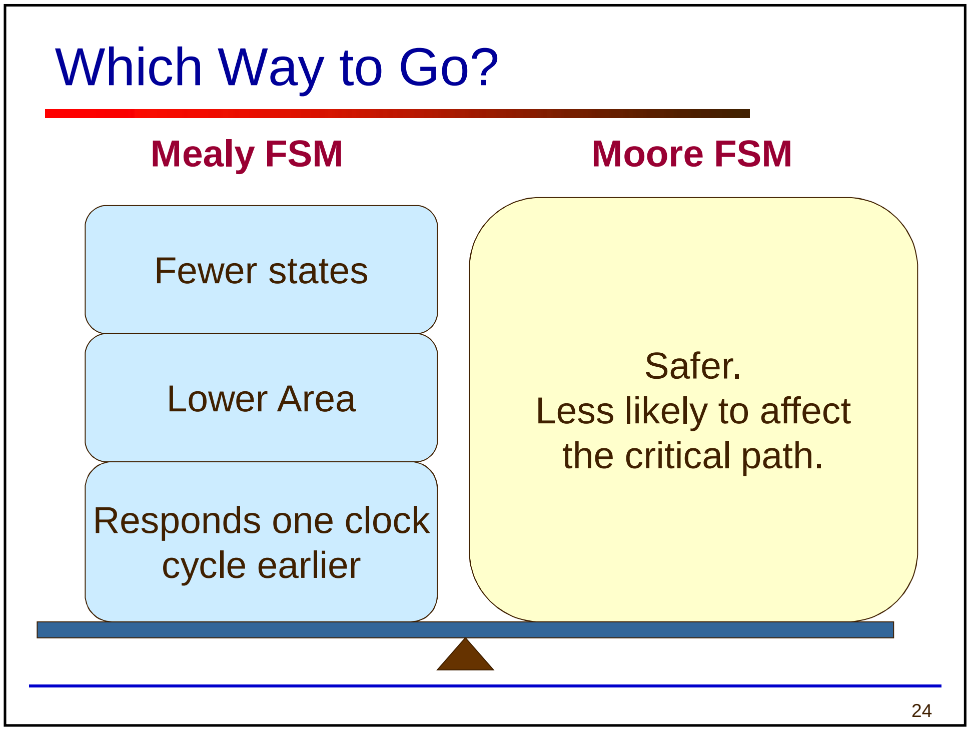
Moore vs. Mealy FSM (1)

- 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 less likely to affect the critical path of the entire circuit

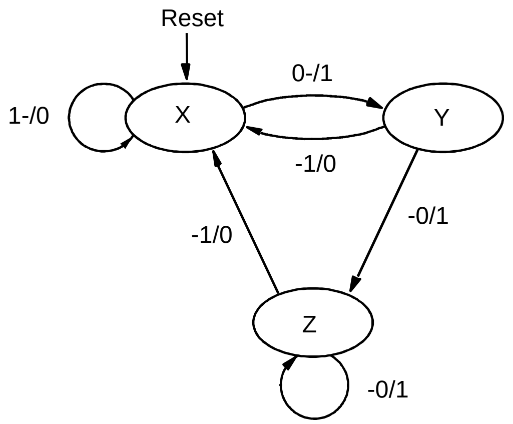


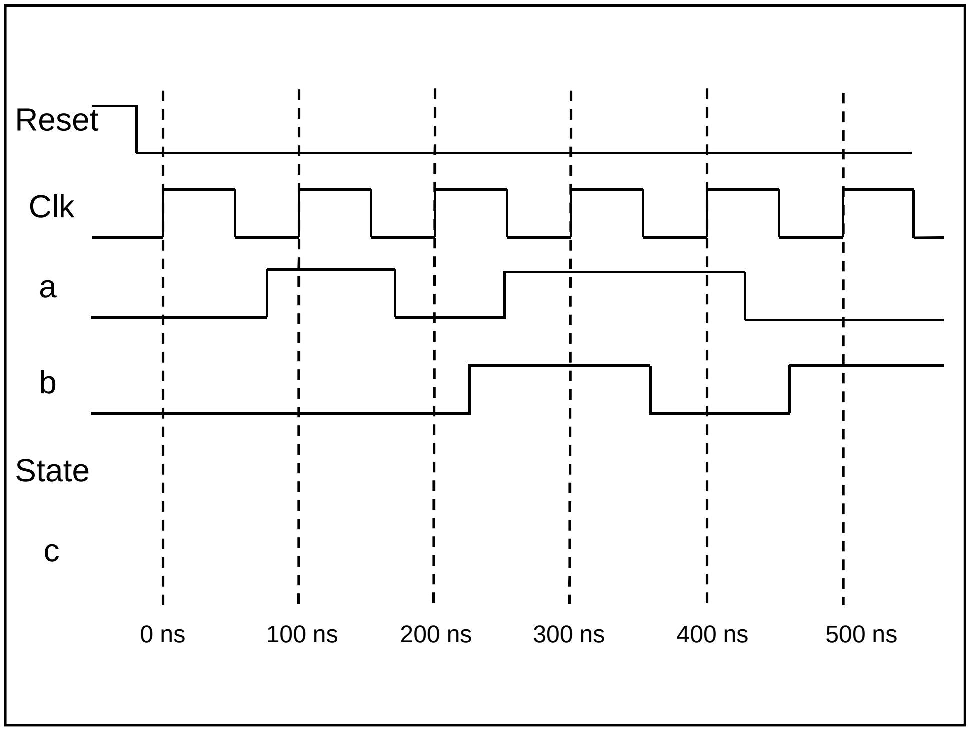
Which Way to Go?

Mealy FSM
Moore FSM
Fewer states
Lower Area
Responds one clock cycle earlier
Safer.
Less likely to affect the critical path.

Problem 1

Assuming state diagram given on the next slide, supplement timing waveforms given in the answer sheet with the correct values of signals **State** and **c**, in the interval from 0 to 575 ns.





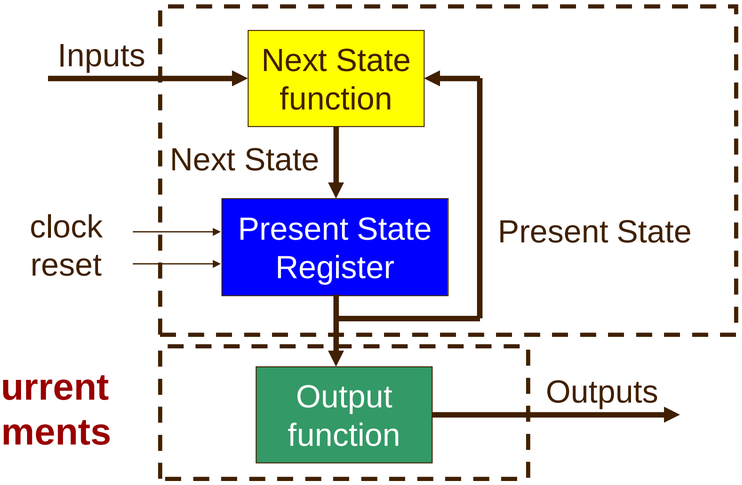
Finite State Machines in VHDL

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
  - Output function described using rules for combinational logic, i.e. as concurrent statements or a process with all inputs in the sensitivity list

Moore FSM

process(clock, reset)



Inputs

Next State function

Next State

clock

reset

Present State Register

Present State

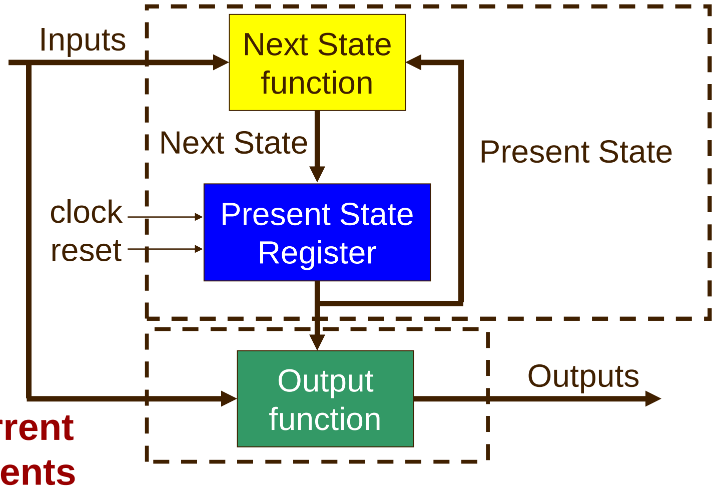
Output function

Outputs

concurrent statements

Mealy FSM

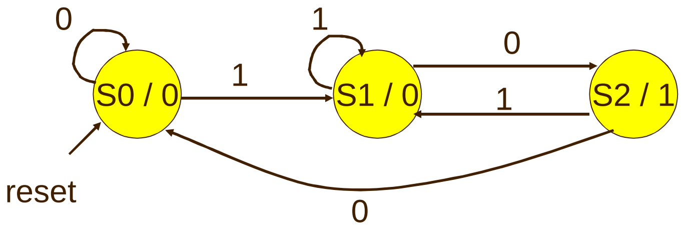
process(clock, reset)



concurrent statements

Moore FSM - Example 1

- Moore FSM that Recognizes Sequence “10”



Moore FSM in VHDL (1)

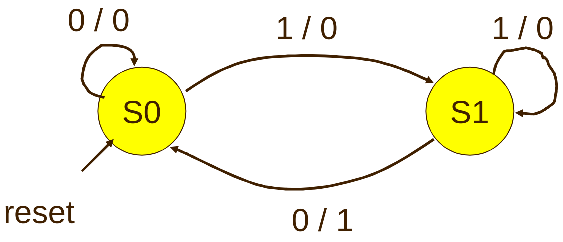
```
TYPE state IS (S0, S1, S2);
SIGNAL Moore_state: state;

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”



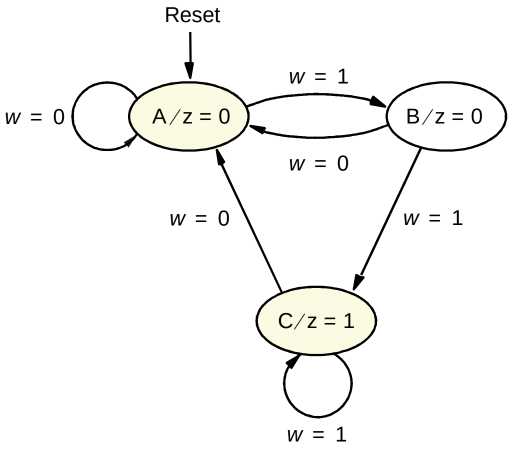
Mealy FSM in VHDL (1)

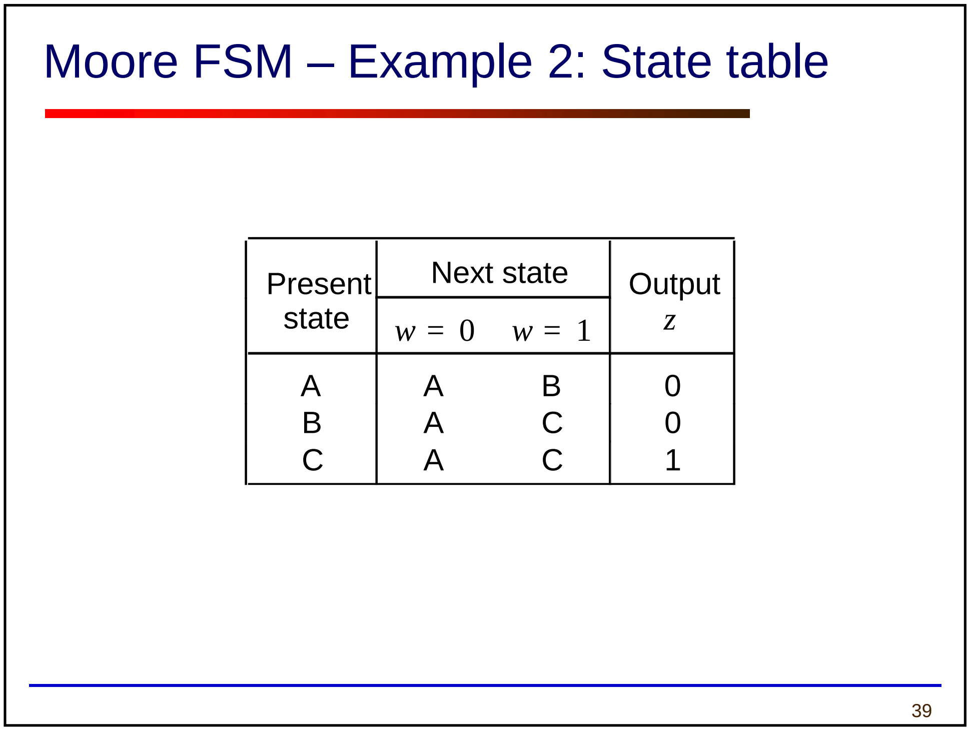
```
TYPE state IS (S0, S1);
SIGNAL Mealy_state: state;

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’;
```

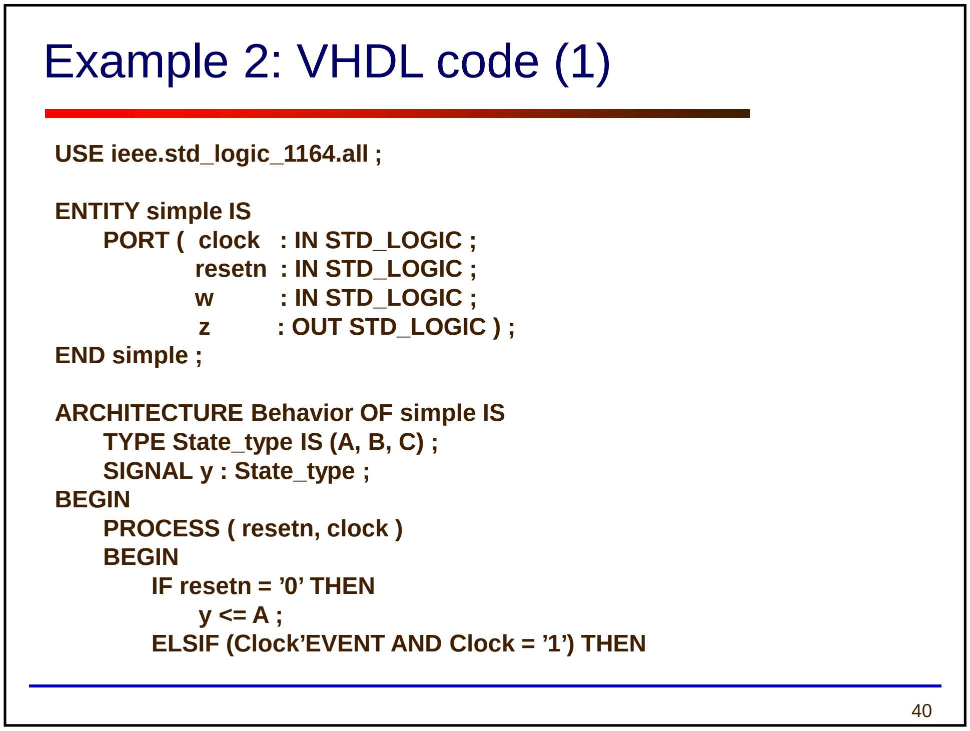
Moore FSM – Example 2: State diagram





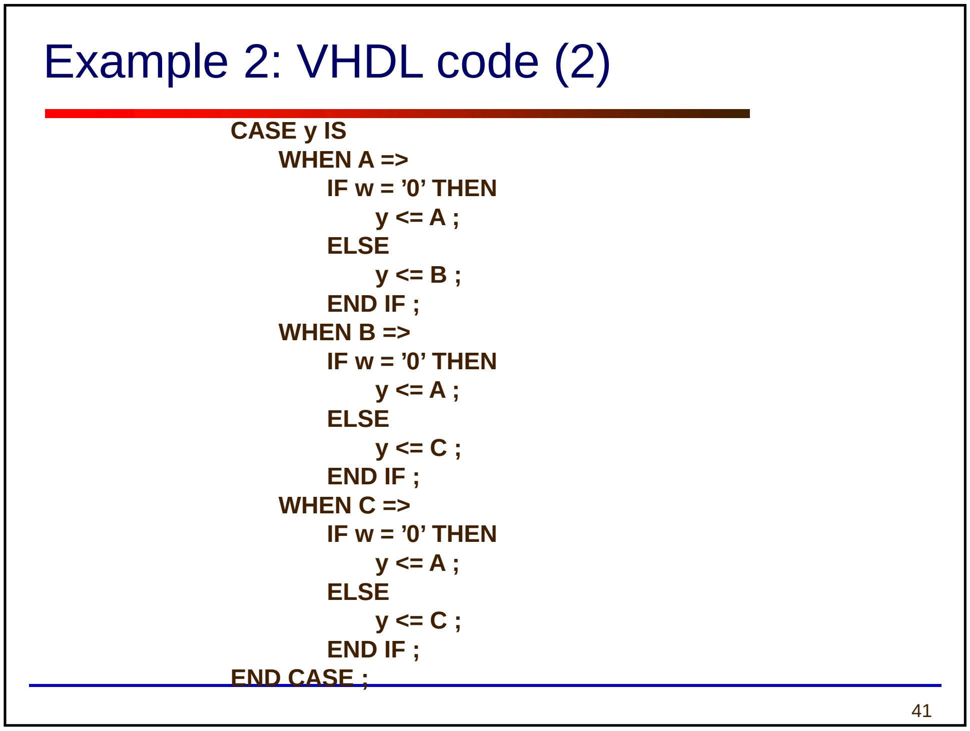
Moore FSM – Example 2: State table

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



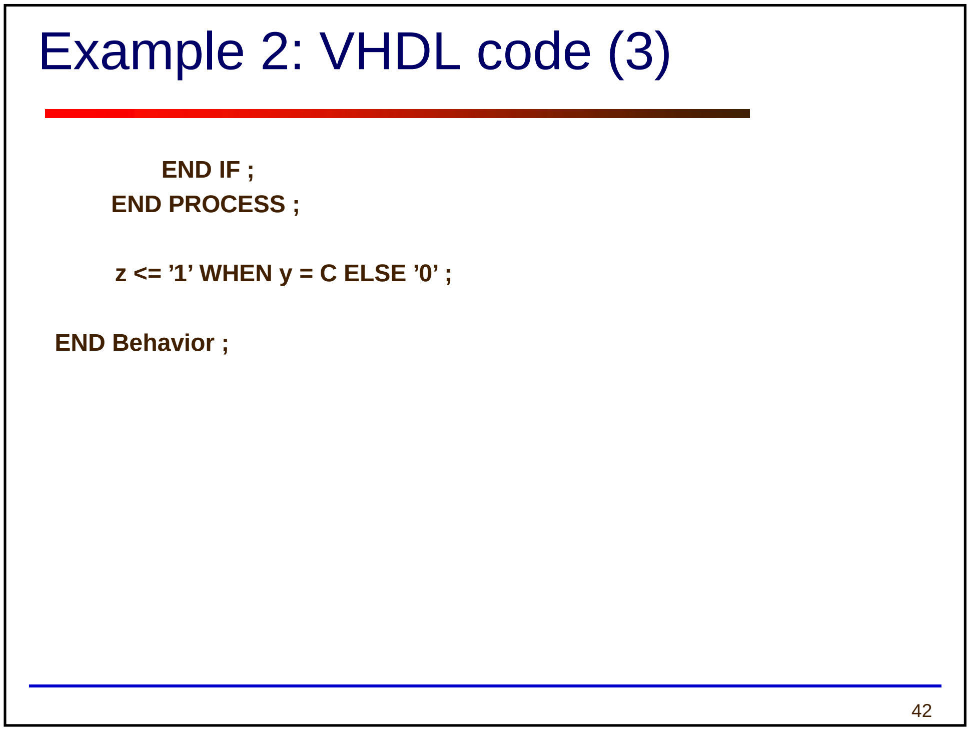
Example 2: VHDL code (1)

```
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
```



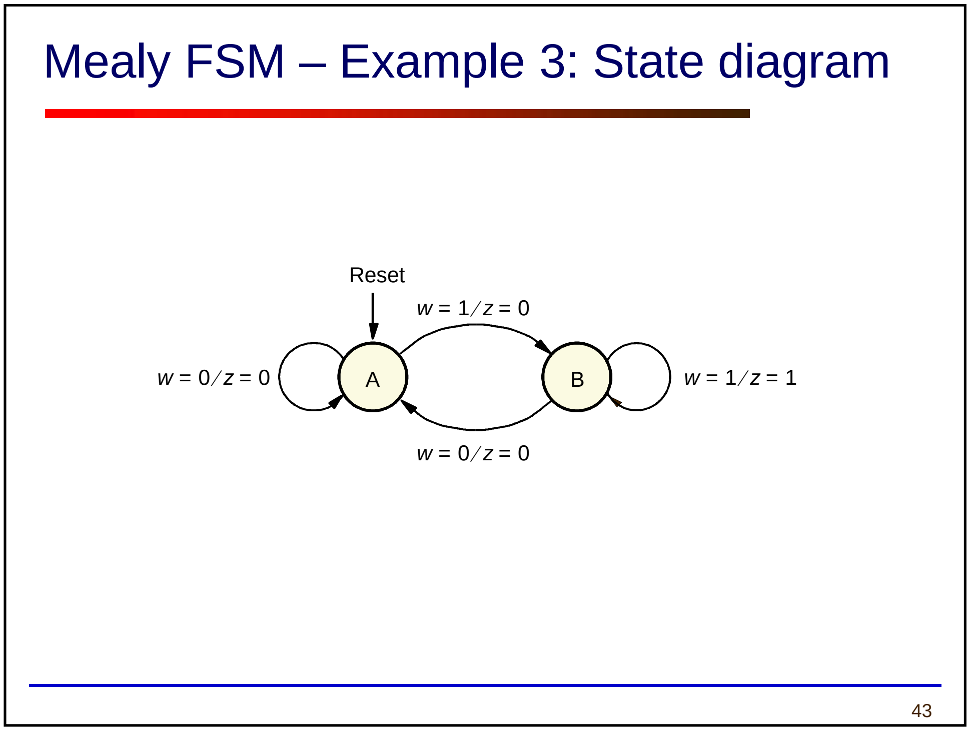
Example 2: VHDL code (2)

```
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 ;
```

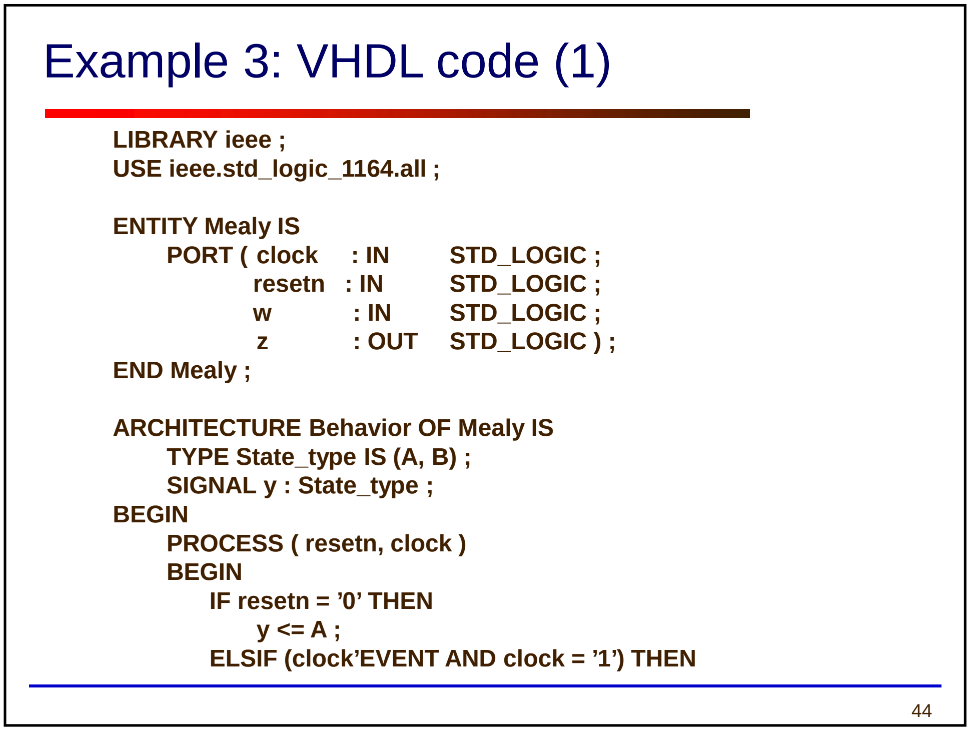


Example 2: VHDL code (3)

```
            END IF ;
        END PROCESS ;

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

END Behavior ;
```

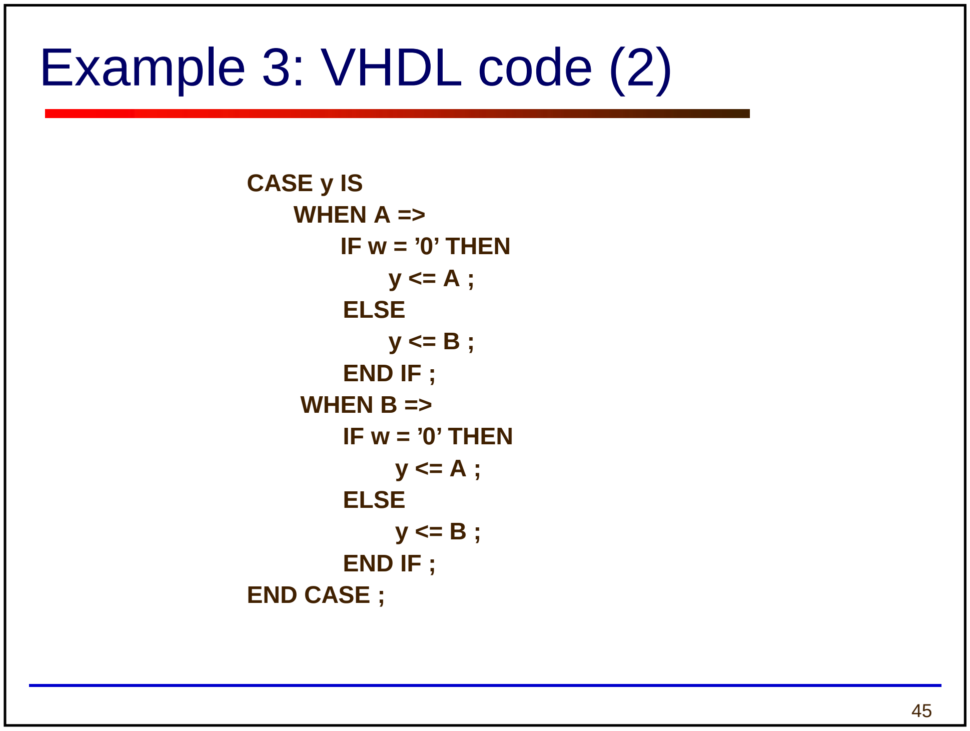


Mealy FSM – Example 3: State diagram



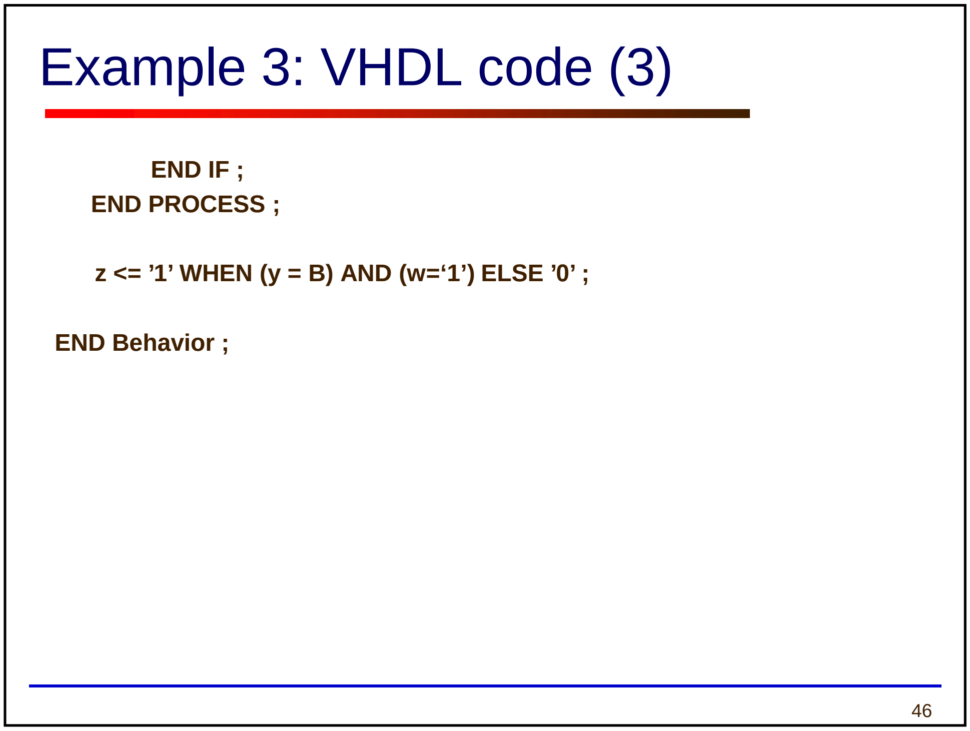
Example 3: VHDL code (1)

```
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 3: VHDL code (2)

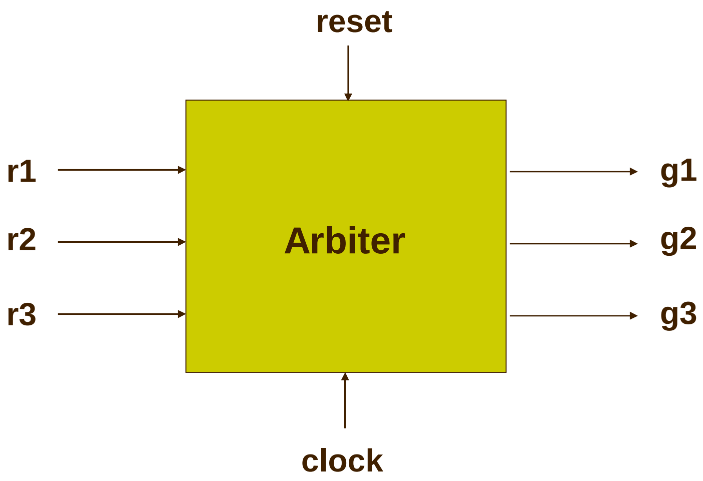
```
CASE y IS
    WHEN A =>
        IF w = ’0’ THEN
            y <= A ;
        ELSE
            y <= B ;
        END IF ;
     WHEN B =>
        IF w = ’0’ THEN
            y <= A ;
        ELSE
            y <= B ;
        END IF ;
END CASE ;
```



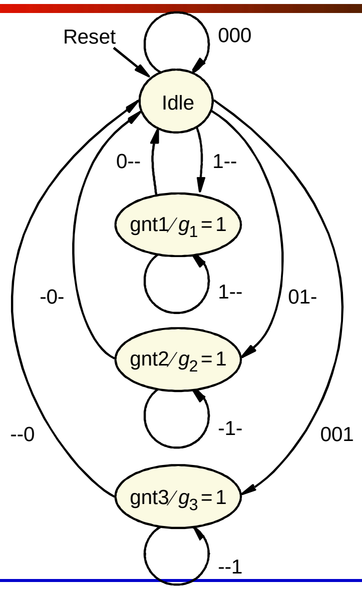
Example 3: VHDL code (3)

```
        END IF ;
    END PROCESS ;

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

END Behavior ;
```

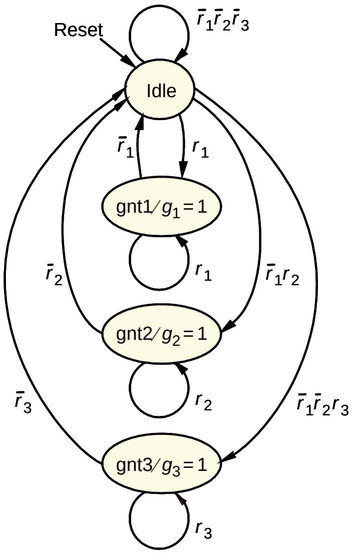
Control Unit Example: Arbiter (1)



Control Unit Example: Arbiter (2)

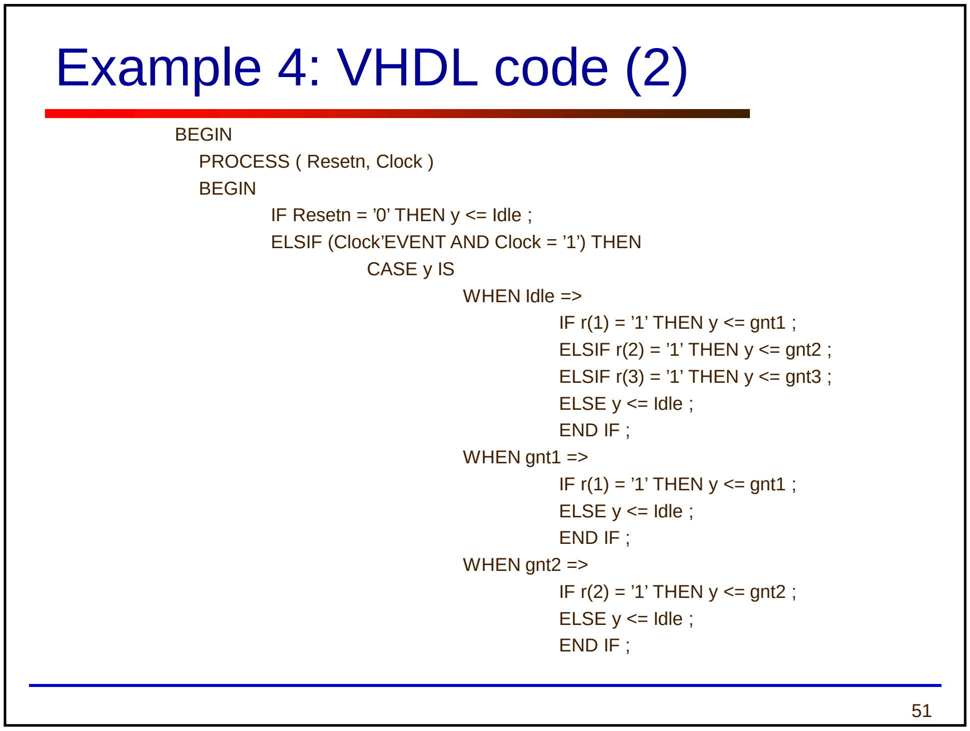


Control Unit Example: Arbiter (3)



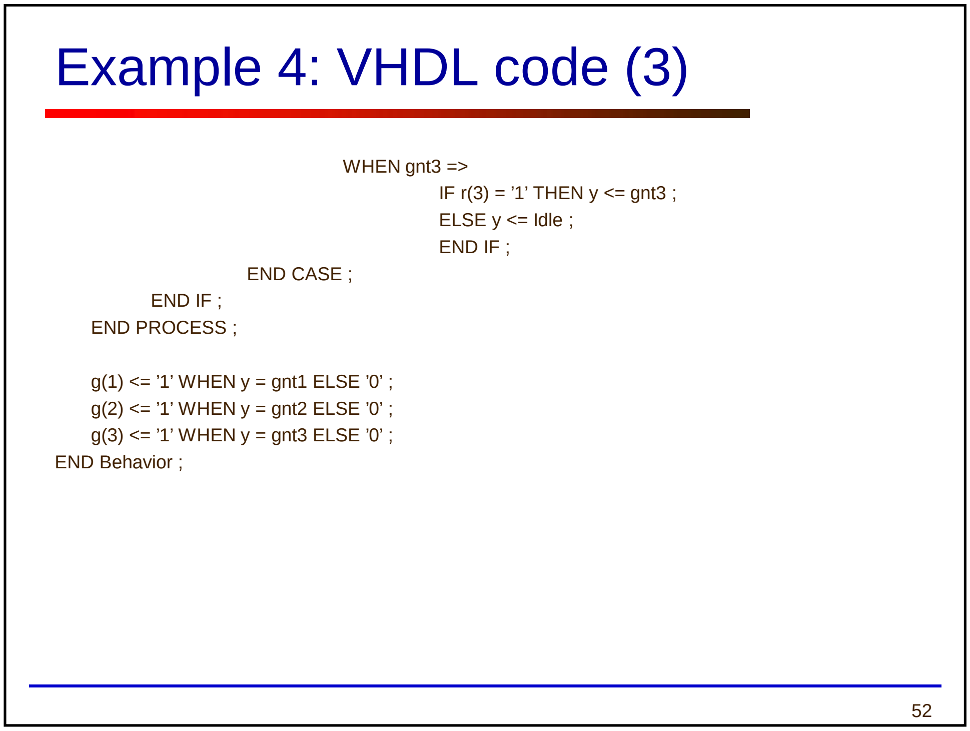
Example 4: VHDL code (1)

```
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 4: 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 ;
```

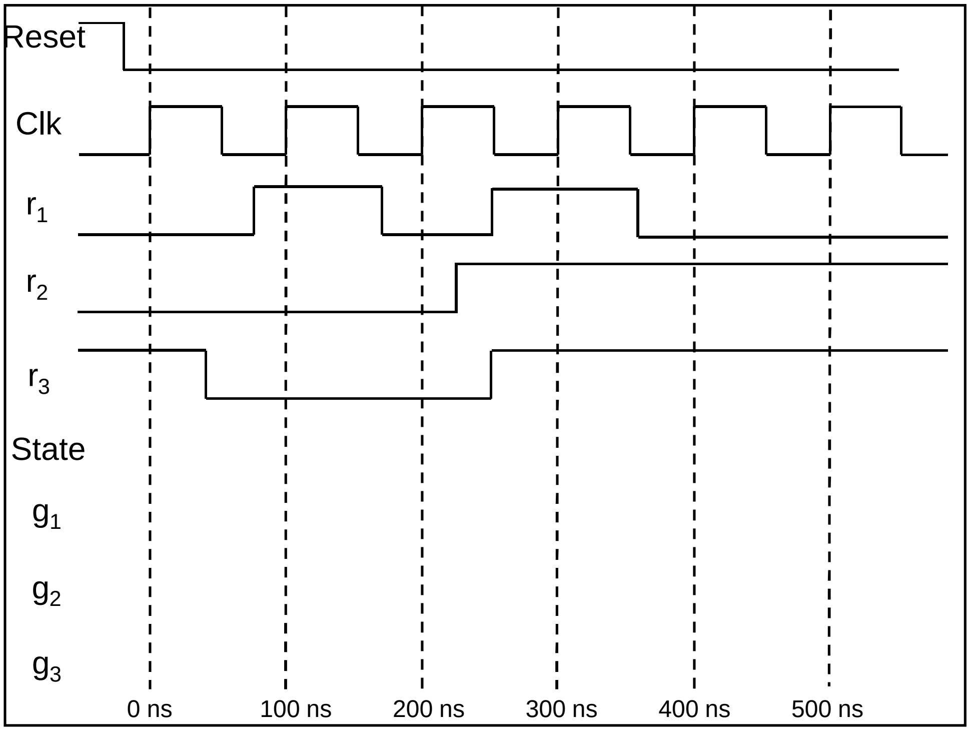


Example 4: VHDL code (3)

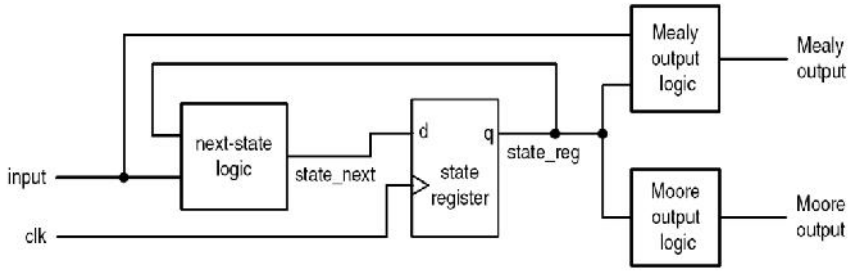
```
                                    WHEN gnt3 =>
                                                IF r(3) = '1' THEN y <= gnt3 ;
                                                ELSE y <= Idle ;
                                                END IF ;
                        END CASE ;
           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

Problem 2

Assuming ASM chart given on the next slide, supplement timing waveforms given in the answer sheet with the correct values of signals **State, g1, g2, g3**, in the interval from 0 to 575 ns.



Generalized FSM

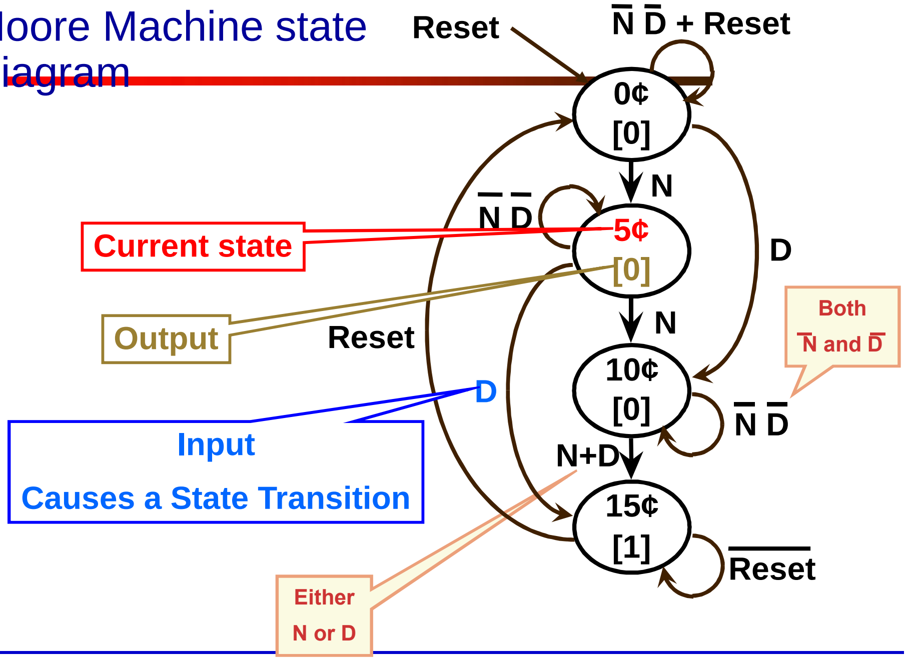


Based on RTL Hardware Design by P. Chu

Moore Machine State Diagram

- **Design a vending machine that has only one coin slot and takes nickels (5¢) and dimes(10¢), and sells candies costing 15¢ each.**
- **If you pay with 2 dimes, it keeps the change.**
- **Transitions occur when you insert a coin.**
- **Must hit reset to restart the machine after vending.**

Moore Machine state Diagram


Reset
$\overline{N}\,\overline{D}$ + Reset
0¢
[0]
N
$\overline{N}\,\overline{D}$
5¢
[0]
D
Current state
Output
Reset
N
Both
$\overline{N}$ and $\overline{D}$
10¢
[0]
D
Input
Causes a State Transition
$\overline{N}\,\overline{D}$
N+D
15¢
[1]
$\overline{\text{Reset}}$
Either
N or D

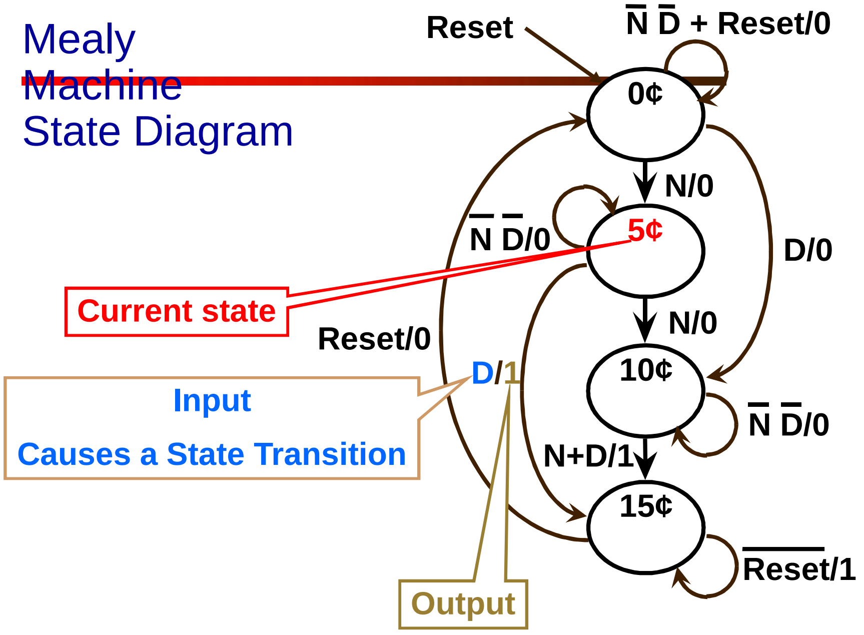
State encoding

- Remember I said that encoding the states was not always obvious.
- Here, the states are 0c, 5c, 10c, 15c.
- You could use 4 F/Fs to encode the actual numbers.
- You could also use 2 F/Fs to encode the number in multiples of 5c.
- 0c ->0, 5c-> 1, 10c->2, 15c->3.
- This costs less to build.
- It also has no forbidden states

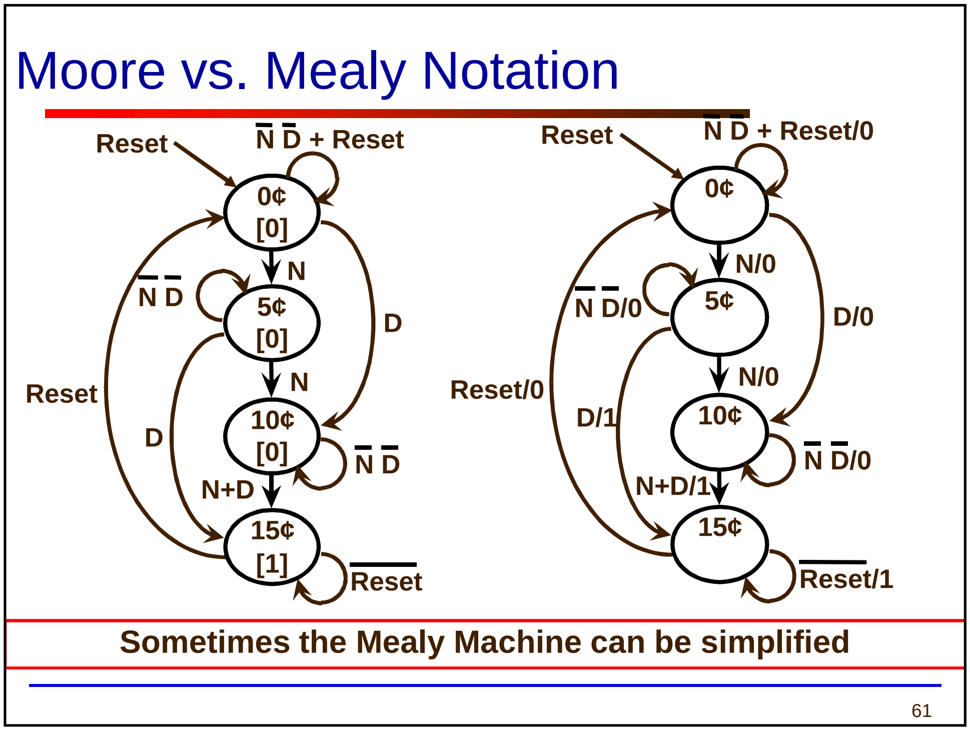
Exercise

- Write the state diagram for a payphone that takes nickels, dimes, and quarters.
- Phone calls cost 25c.
- No change is given.
- However if you hit a ‘next call’ button, when you have credit, you can use that to pay for the next call.

Mealy ~~Machine~~ State Diagram



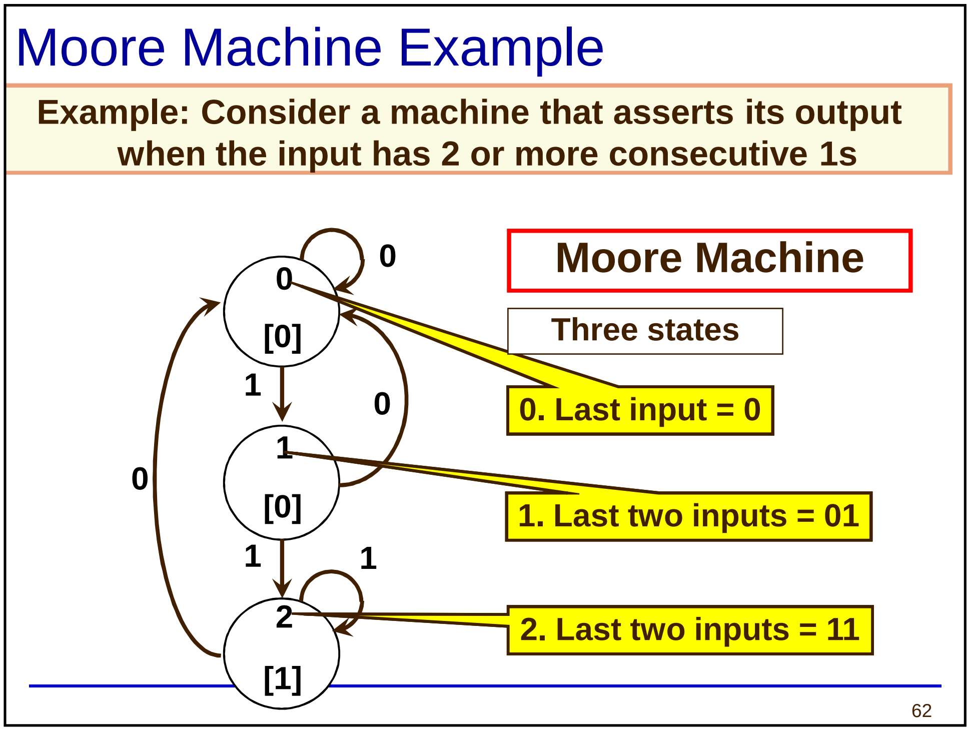
Reset
$\overline{N}\,\overline{D}$ + Reset/0
0¢
N/0
$\overline{N}\,\overline{D}$/0
5¢
D/0
Current state
Reset/0
N/0
D/1
10¢
Input
Causes a State Transition
$\overline{N}\,\overline{D}$/0
N+D/1
15¢
$\overline{\text{Reset}}$/1
Output



Moore vs. Mealy Notation

Reset
$\overline{N}\,\overline{D}$ + Reset
0¢ [0]
N
$\overline{N}\,\overline{D}$
5¢ [0]
D
N
Reset
D
10¢ [0]
$\overline{N}\,\overline{D}$
N+D
15¢ [1]
$\overline{\text{Reset}}$
Reset
$\overline{N}\,\overline{D}$ + Reset/0
0¢
N/0
$\overline{N}\,\overline{D}$/0
5¢
D/0
N/0
Reset/0
D/1
10¢
$\overline{N}\,\overline{D}$/0
N+D/1
15¢
$\overline{\text{Reset}}$/1

Sometimes the Mealy Machine can be simplified



Moore Machine Example

Example: Consider a machine that asserts its output when the input has 2 or more consecutive 1s

Moore Machine

Three states

0. Last input = 0

1. Last two inputs = 01

2. Last two inputs = 11

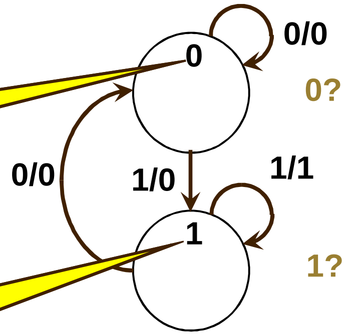
Mealy Machine Example

Example: Consider a machine that asserts its output when the input has 2 or more consecutive 1s

Mealy Machine

Two states

Interpretation of states

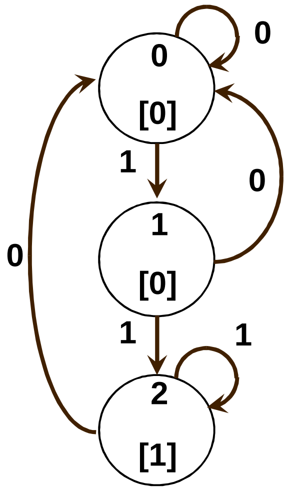


0. Last input = 0.
Output = 0, for either input

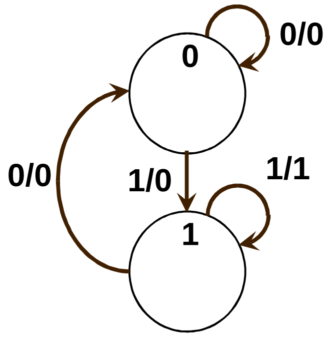
1. Last input = 1.
Output = 1 for input =1,
Output = 0 for input = 0.

Compare Moore vs. Mealy

Moore Machine



Mealy Machine

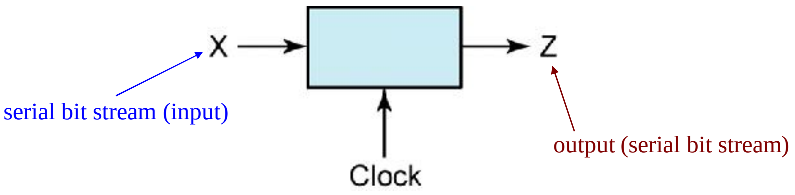


Designers prefer Mealy, because of the fewer states.

Example: A sequence detector (Mealy)

To illustrate the design of a clocked **Mealy** sequential circuit, we will design a sequence detector.

The circuit is of the form:



serial bit stream (input)

output (serial bit stream)

Example: A sequence detector (Mealy)

Suppose we want to design the sequence detector so that any input sequence ending in 101 will produce an output of Z = 1 coincident with the last 1.

The circuit does not reset when a 1 output occurs.

A typical input sequence and the corresponding output sequence are:

X =	0	0	1	1	0	1	1	0	0	1	0	1	0	1	0	0
Z =	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0
(time:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15)

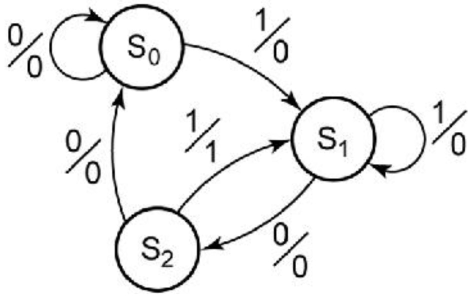
Example: A sequence detector (Mealy)

Initially, we do not know how many flip-flops will be required, so we will designate the circuit states as S_0, S_1, etc.

We will start with a reset state designated S_0.

If a 0 input is received, the circuit can stay in S_0 because the input sequence we are looking for does not start with a 0.

Example: A sequence detector (Mealy)



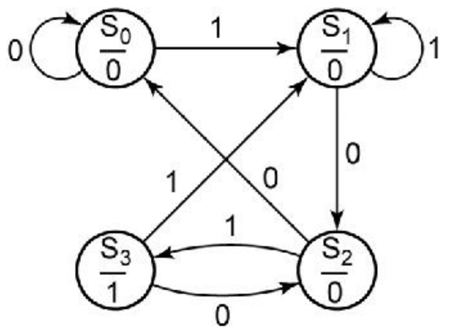
State Graph for the Mealy Machine

Example: A sequence detector (Moore)

The procedure for finding the state graph for a Moore machine is similar to that used for a Mealy machine, except that the output is written with the state.

We will rework the previous example as a **Moore** machine: the circuit should produce an output of 1 only if an input sequence ending in 101 has occurred.

Example: A sequence detector (Moore)



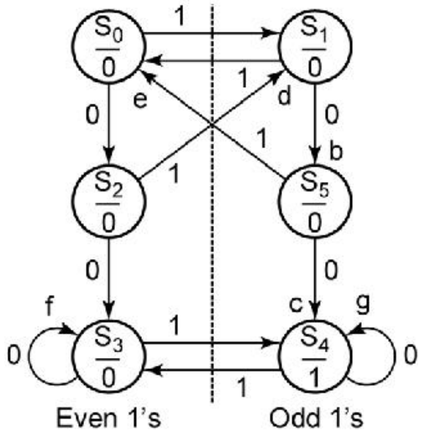
State Graph for the Moore Machine

Example: Another FSM (Moore)

Design a Moore sequential circuit with one input *X* and one output *Z*. The output *Z* is to be 1 if the total number of 1's received is odd and at least two consecutive 0's have been received. A typical input and output sequence is:

$$\begin{array}{lcccccccccc}
X = & & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
 & & \uparrow & & & \uparrow & & \uparrow & \uparrow & \uparrow \\
 & & a & & & b & & c & d & e \\
Z = & (0) & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1
\end{array}$$

Example: Another FSM (Moore)

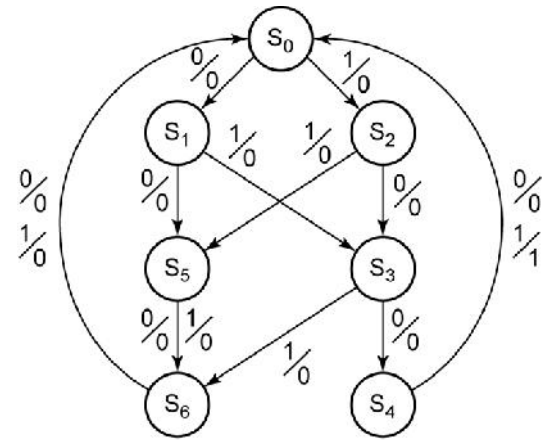


Example: Another FSM (Mealy)

A sequential circuit has one input (*X*) and one output (*Z*). The circuit examines groups of four consecutive inputs and produces an output *Z* = 1 if the input sequence 0101 or 1001 occurs. The circuit resets after every four inputs. Find a Mealy state graph. A typical input and output sequence is:

$$X = 0101 \mid 0010 \mid 1001 \mid 0100$$
$$Z = 0001 \mid 0000 \mid 0001 \mid 0000$$

Example: Another FSM (Mealy)



Exercise

Specifications for a deposit return machine

The machine returns the 5¢ deposit on a container to the user.

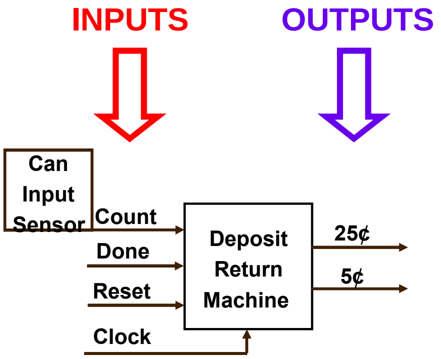
For every five containers returned, the machine dispenses a quarter.

If the user has less than five containers credit when the user presses the DONE button, the machine dispenses the correct number of nickels.

If the DONE button is pressed with no containers in the machine, no coins are

Deposit Machine - 1

Step 1. Understand the problem



Block Diagram

Deposit Machine - 2a

Step 2. Map into more suitable abstract representation

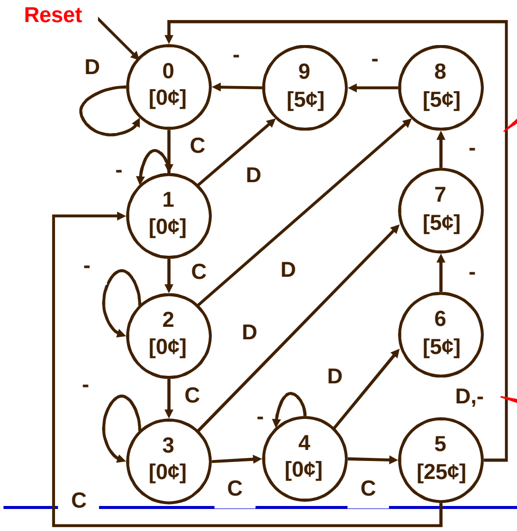
Typical input sequences

0 cans @ 0¢, Done
1 can @ 5¢, Done
2 cans @ 10¢, Done
3 cans @ 15¢, Done
4 cans @ 20¢, Done
5 cans @ 25¢, Done
5 cans @ 25¢ ì dispense quarter, 0 cans @ 0¢

Assumption:
Can Sensor Input and Done Button cannot be asserted at the same time

Deposit Machine - 2b

Step 2. Map into more suitable abstract representation



i.e., C' D'

i.e., D + C' D'

State diagram notation

Inputs: C(an), D(one)
Note: Reset (is used only to start the FSM)

Outputs: [25¢], [5¢]