

Finite State Machines

יהודה אפק, נתן אינטרטור
אוניברסיטת תל אביב

Based on lectures from
George Mason and CMU

Resources

- Sundar Rajan, *Essential VHDL: RTL Synthesis Done Right*
Chapter 6, Finite State Machines
Chapter 10, Getting the Most from Your State Machine
- *Introduction to VHDL*
http://www-ee.uta.edu/Online/Zhu/Fall_2004/
- *VHDL CHIP* <http://altera.com/>

Definition of a State Machine

- All programmable logic designs can be specified in Boolean form. However some designs are easier to conceptualize and implement using non-Boolean models. The State Machine model is one such model.

Definition of a State Machine

- A state machine represents a system as a set of states, the transitions between them, along with the associated inputs and outputs.
- So, a state machine is a particular conceptualization of a particular sequential circuit. State machines can be used for many other things beyond logic design and computer architecture.

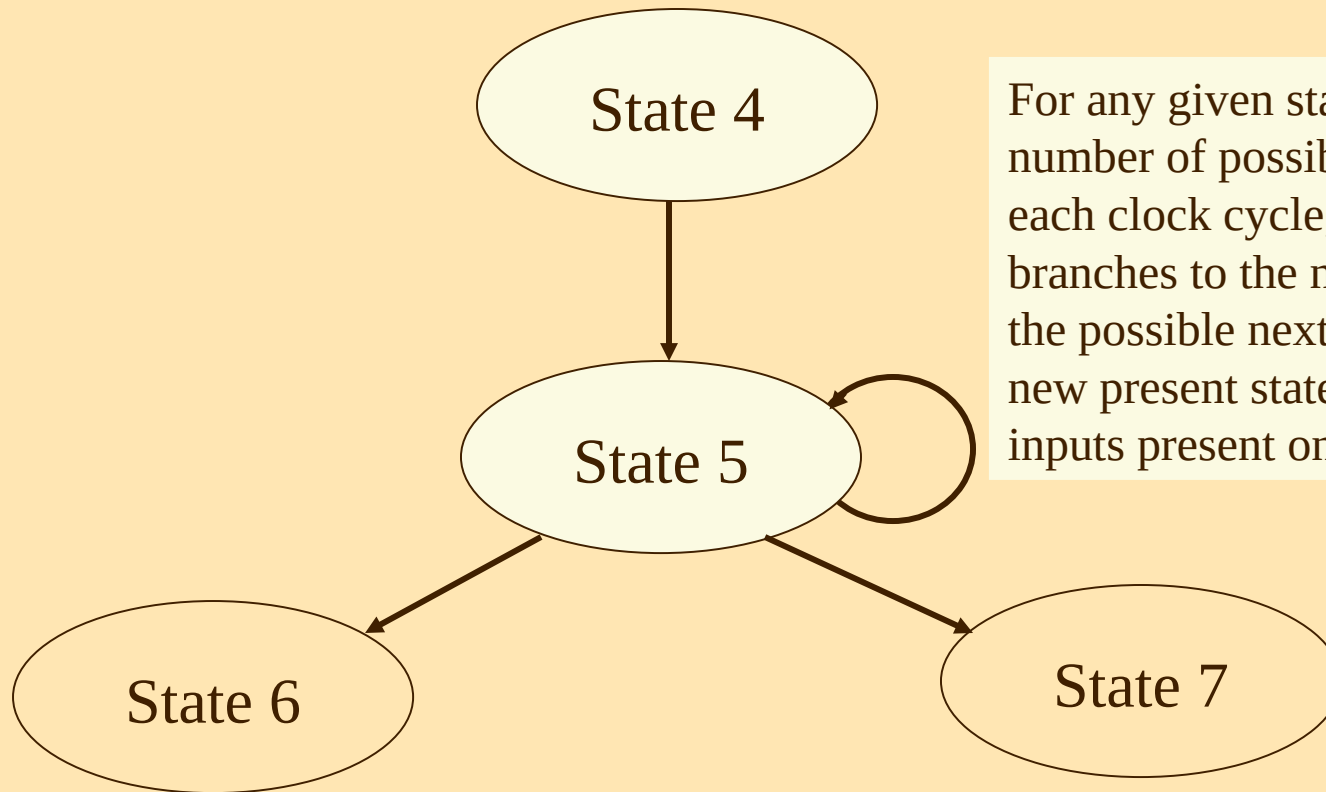
Finite State Machines

- Any Circuit with Memory Is a Finite State Machine
 - Even computers can be viewed as huge FSMs
- Design of FSMs Involves
 - Defining states
 - Defining transitions between states
 - Optimization / minimization
- Above Approach Is Practical for Small FSMs Only

State Machines: Definition of Terms

- State Diagram
 - Illustrates the form and function of a state machine. Usually drawn as a bubble-and-arrow diagram.
- State
 - A uniquely identifiable set of values measured at various points in a digital system.
- Next State
 - The state to which the state machine makes the next transition, determined by the inputs present when the device is clocked.
- Branch
 - A change from present state to next state.
- Mealy Machine
 - A state machine that determines its outputs from the present state and from the inputs.
- Moore Machine
 - A state machine that determines its outputs from the present state only.

Present State and Next State



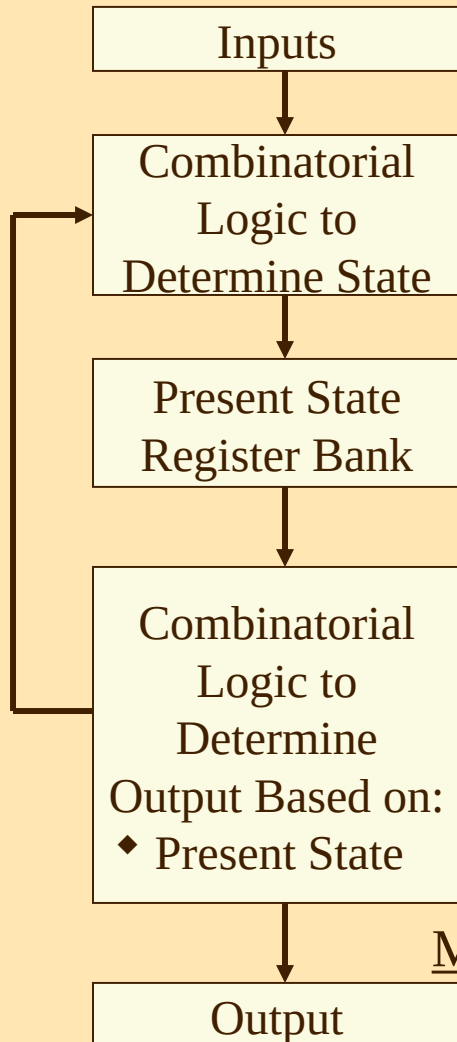
For any given state, there is a finite number of possible next states. On each clock cycle, the state machine branches to the next state. One of the possible next states becomes the new present state, depending on the inputs present on the clock cycle.

- On a well-drawn state diagram, all possible transitions will be visible, including loops back to the same state. From this diagram it can be deduced that if the present state is State 5, then the previous state was either State 4 or 5 and the next state must be either 5, 6, or 7.

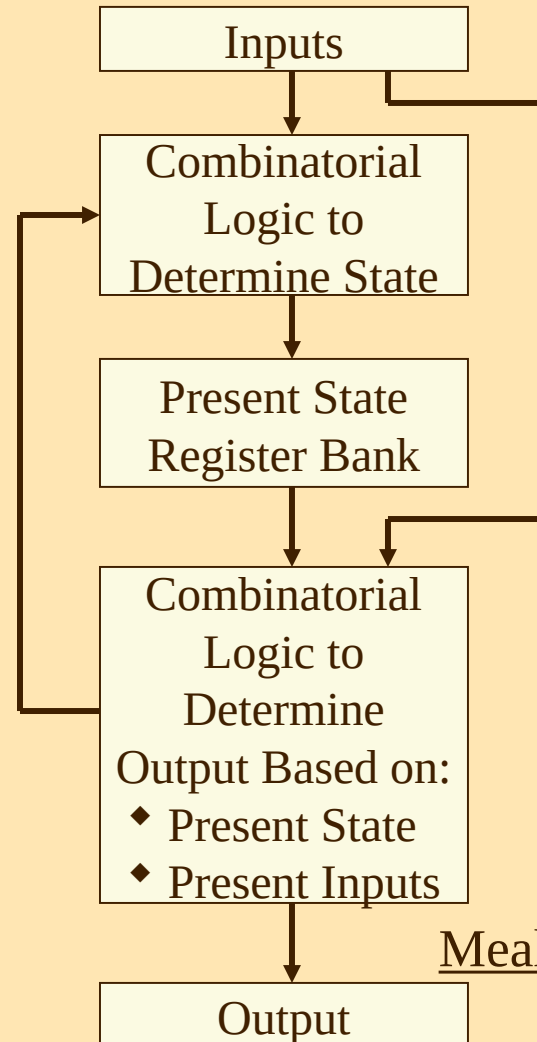
Moore and Mealy Machines

- Both these machine types follow the basic characteristics of state machines, but differ in the way that outputs are produced.
- **Moore Machine:**
 - Outputs are independent of the inputs, ie outputs are effectively produced from within the state of the state machine.
- **Mealy Machine:**
 - Outputs can be determined by the present state alone, or by the present state and the present inputs, ie outputs are produced as the machine makes a transition from one state to another.

Machine Models



Moore Machine

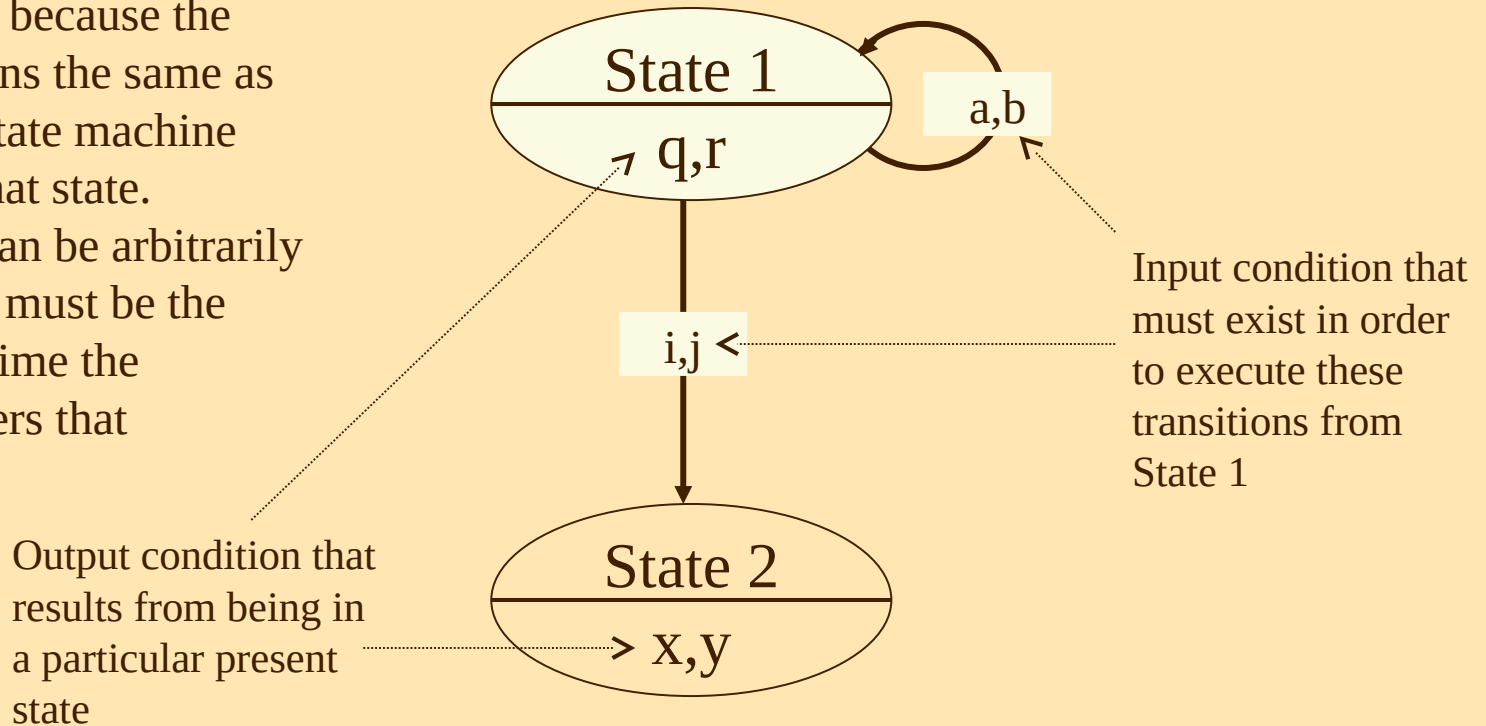


Mealy Machine

Moore Machine Diagrams

The Moore State Machine output is shown inside the state bubble, because the output remains the same as long as the state machine remains in that state.

The output can be arbitrarily complex but must be the same every time the machine enters that state.

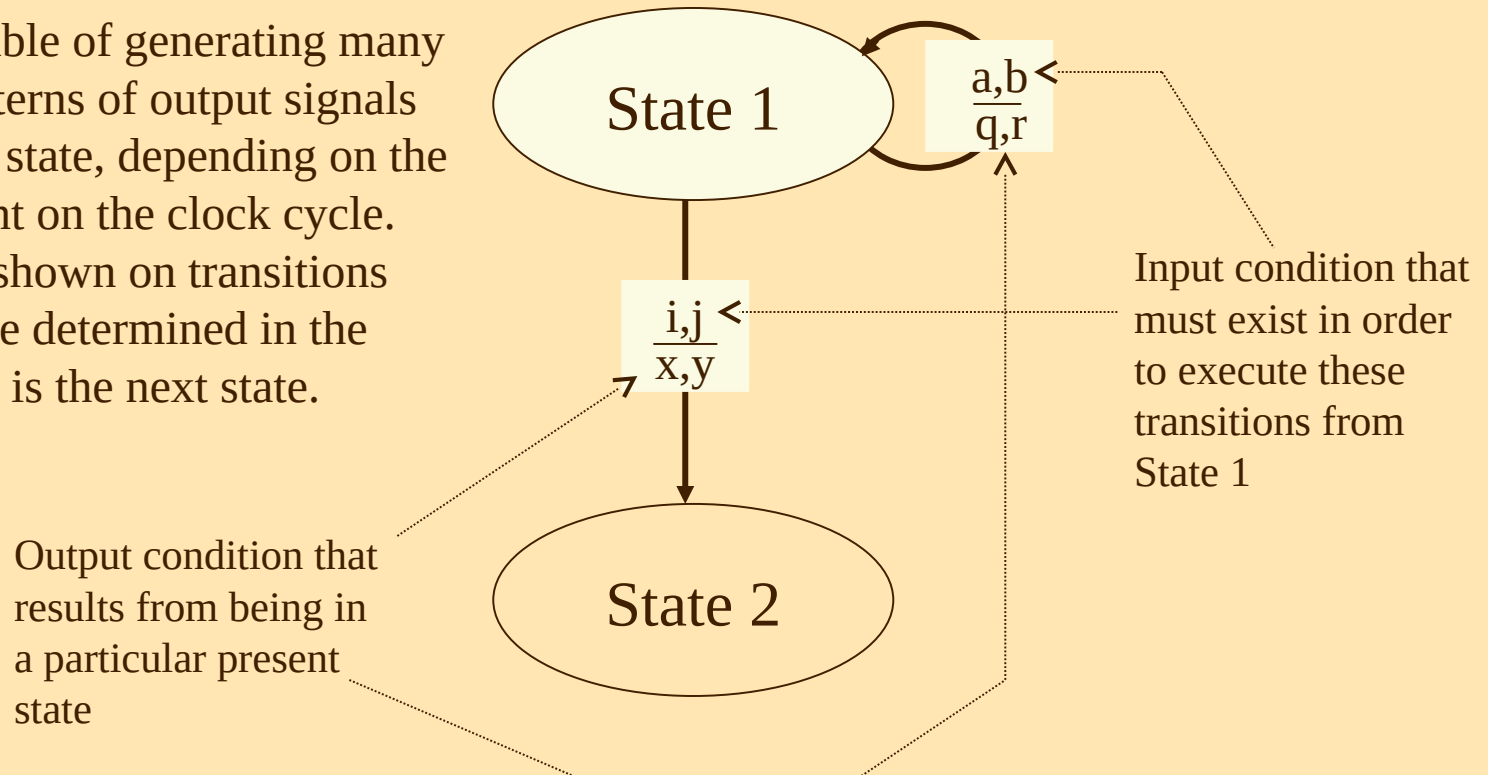


Mealy Machine Diagrams

The Mealy State Machine generates outputs based on:

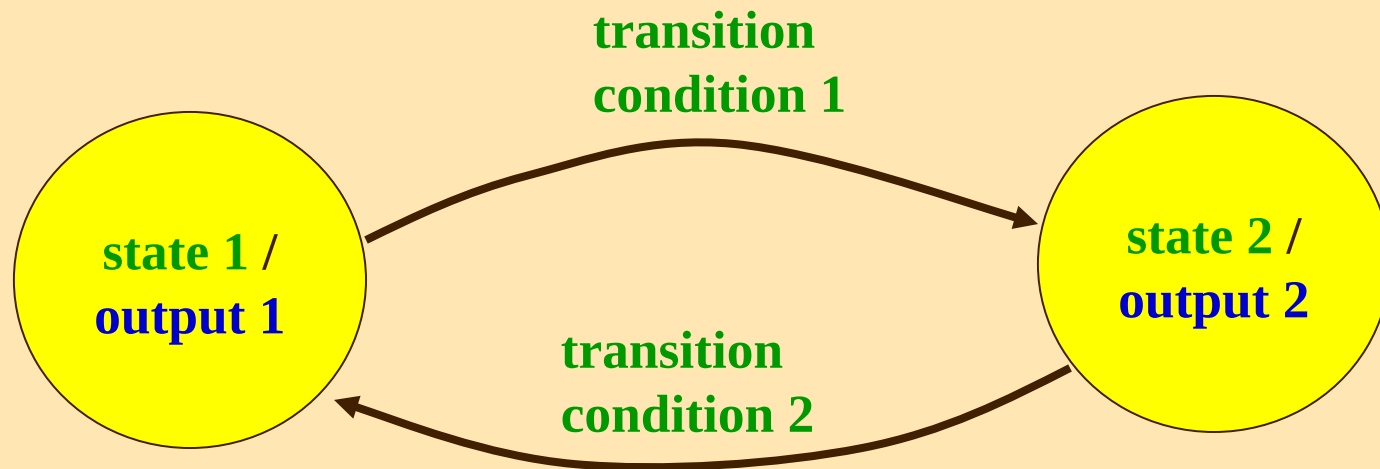
- ♦ The Present State, and
- ♦ The Inputs to the M/c.

So, it is capable of generating many different patterns of output signals for the same state, depending on the inputs present on the clock cycle. Outputs are shown on transitions since they are determined in the same way as is the next state.



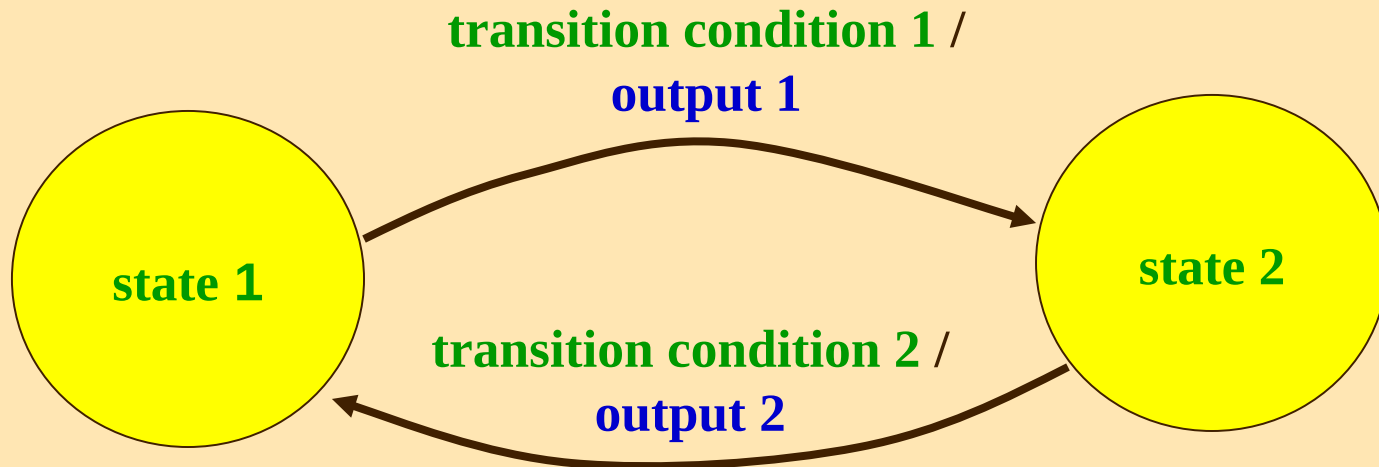
Moore Machine

- Describe Outputs as Concurrent Statements Depending on State Only



Mealy Machine

- Describe Outputs as Concurrent Statements Depending on State and Inputs



Moore vs. Mealy FSM (1)

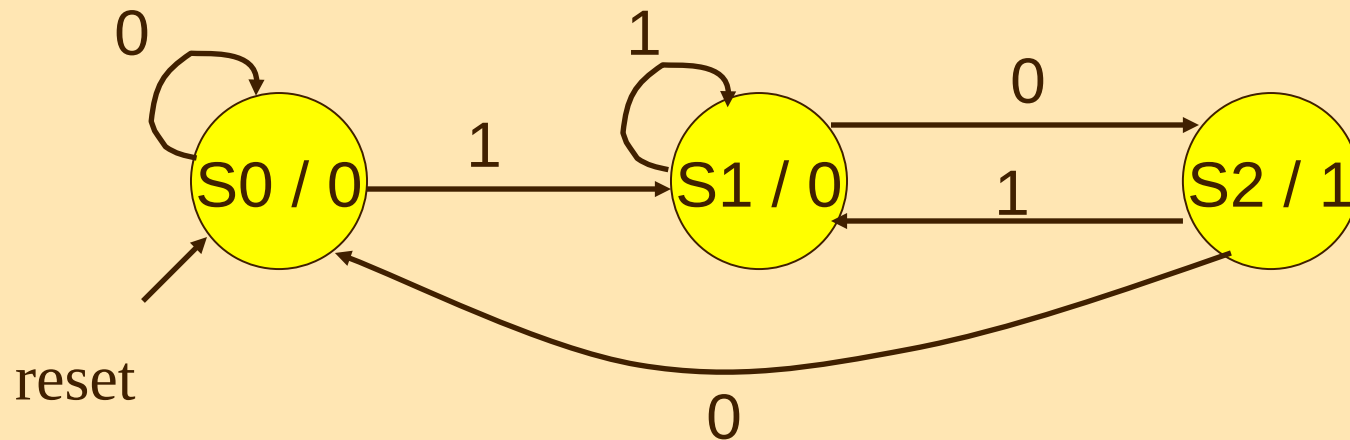
- Moore and Mealy FSMs Can Be Functionally Equivalent
- Mealy FSM Has Richer Description and Usually Requires Smaller Number of States
 - Smaller circuit area

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 have a shorter critical path

Moore FSM - Example 1

- Moore FSM that Recognizes Sequence 10



Meaning
of states:

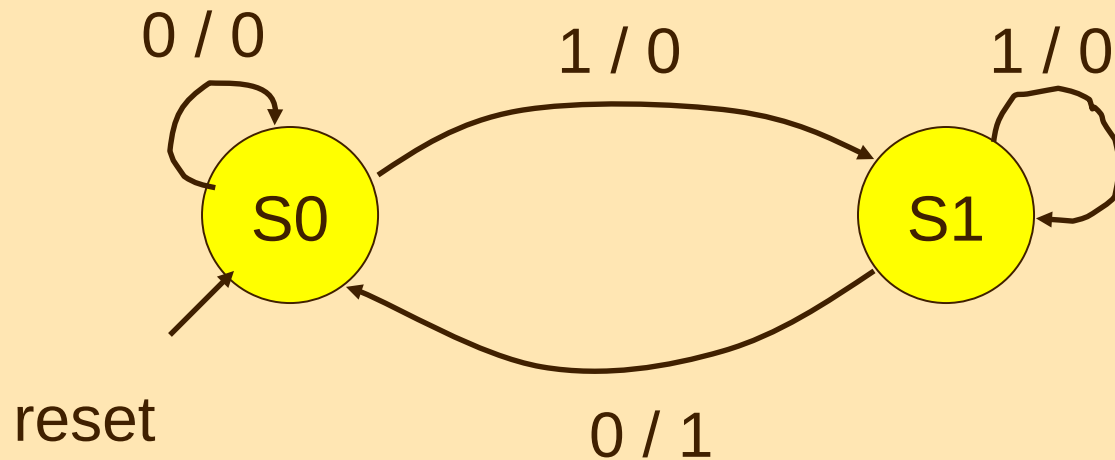
S0: No
elements
of the
sequence
observed

S1: "1"
observed

S2: "10"
observed

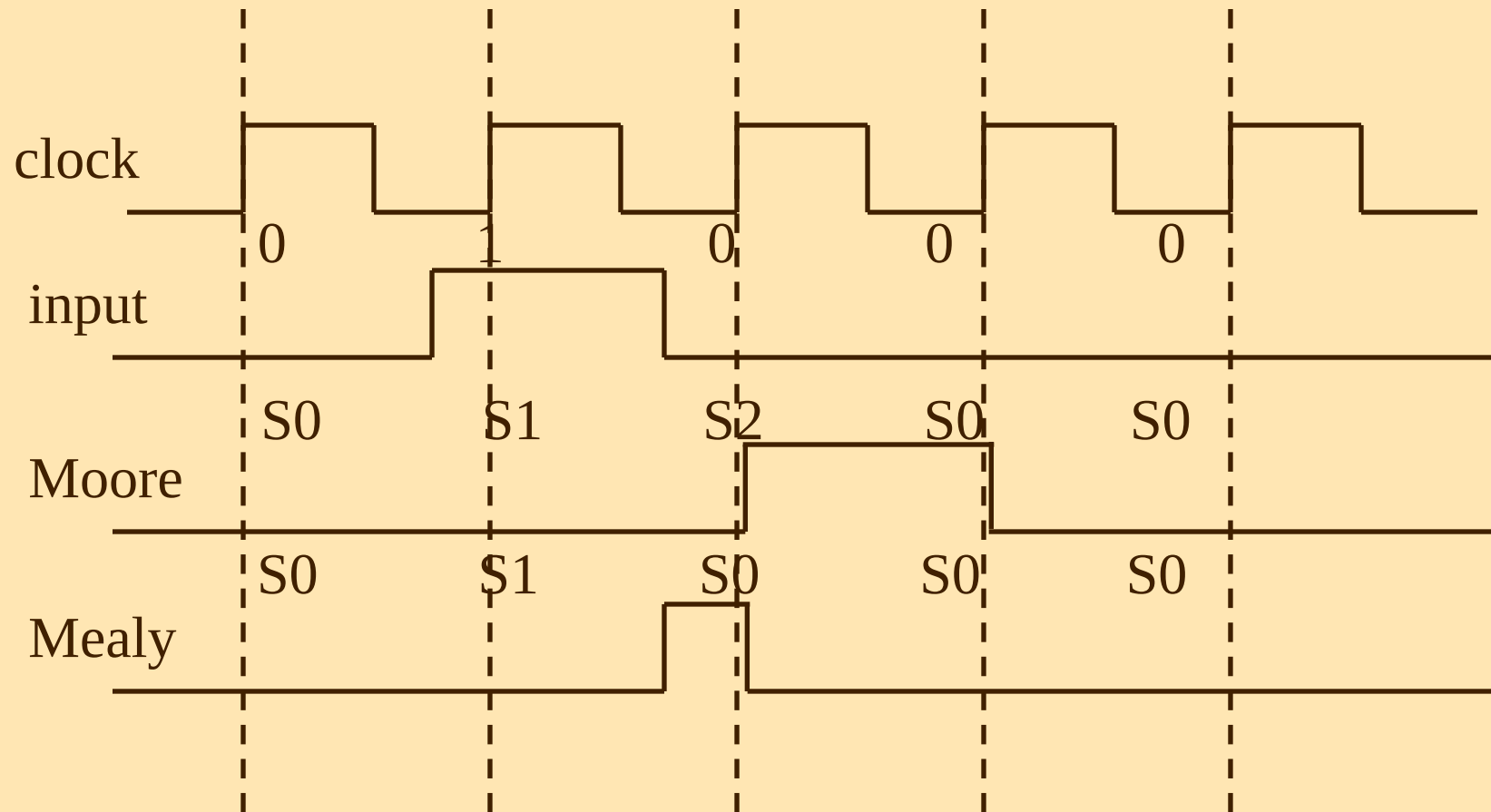
Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence 10

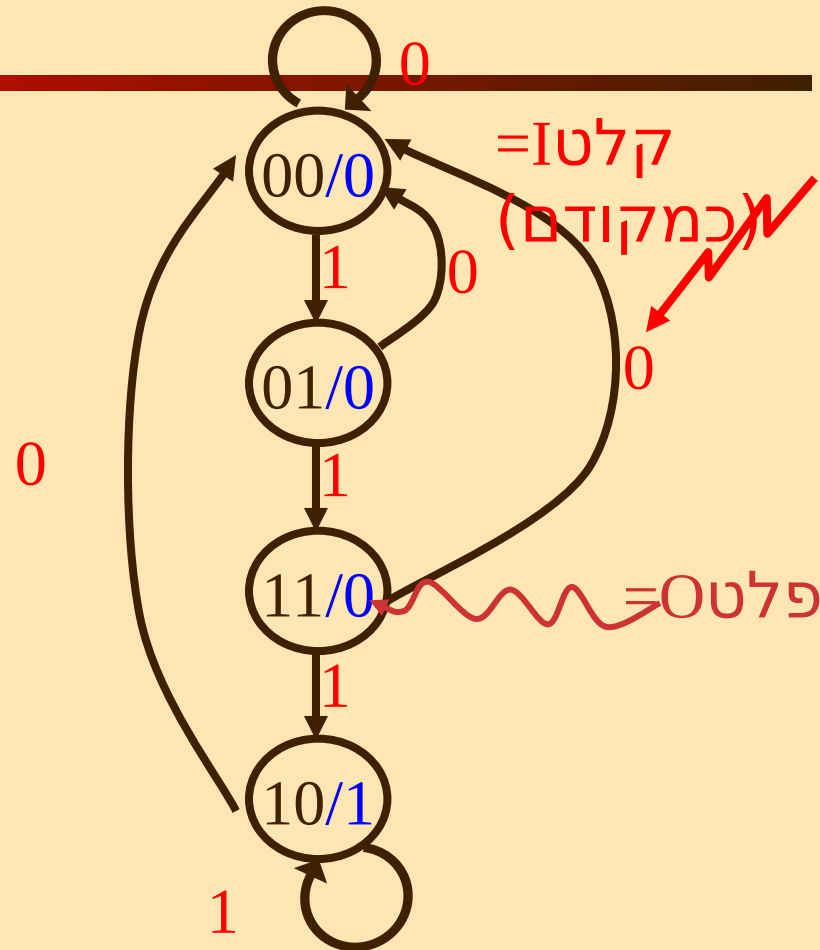


Meaning of states:	S0: No elements of the sequence observed	S1: "1" observed
-----------------------	--	---------------------

Moore & Mealy FSMs – Example 1

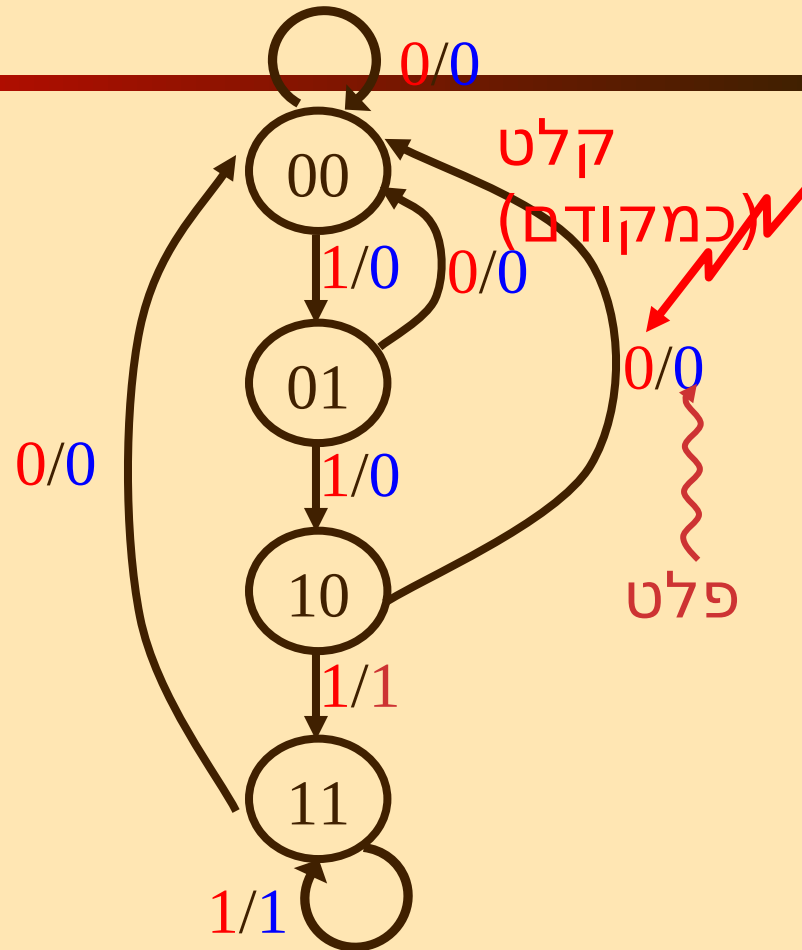


Finite State Machine (FSM)



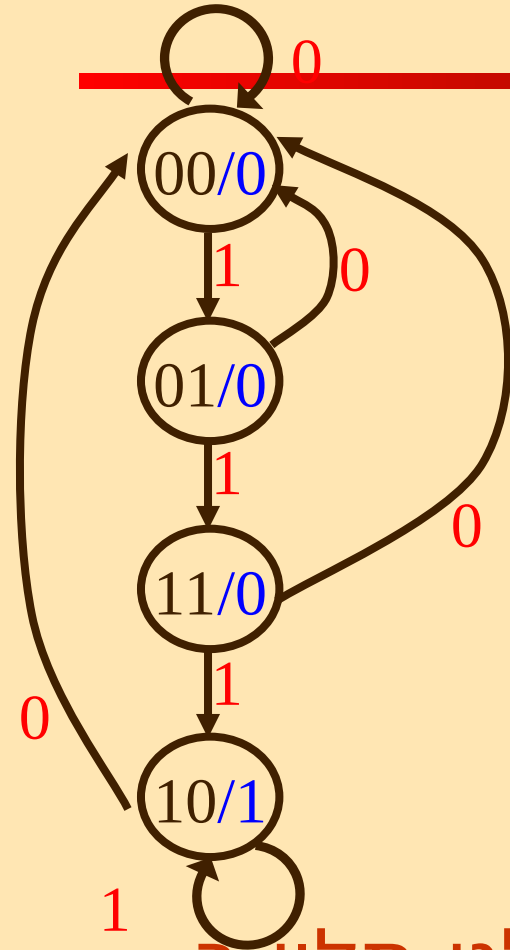
- האוטומט פולט 1 אחרי ש"ראה" לפחות 3 1-ים מאז ה-0 האחרון.
- Moore FSM פלט מצוייר במצבים

Finite State Machine (FSM)



- האוטומט פולט 1 אחרי ש"ראה" לפחות 3 1-ים מאז ה-0 האחרון.
- Mealy FSM פלט מצוייר על הקשתות.

Moore - טבלת המצבים



		I=0		I=1		
		A_{t+1}	B_{t+1}	A_{t+1}	B_{t+1}	
A_t	B_t	A_{t+1}	B_{t+1}	A_{t+1}	B_{t+1}	O_t
0	0	0	0	0	1	0
0	1	0	0	1	1	0
1	1	0	0	1	0	0
1	0	0	0	1	0	1

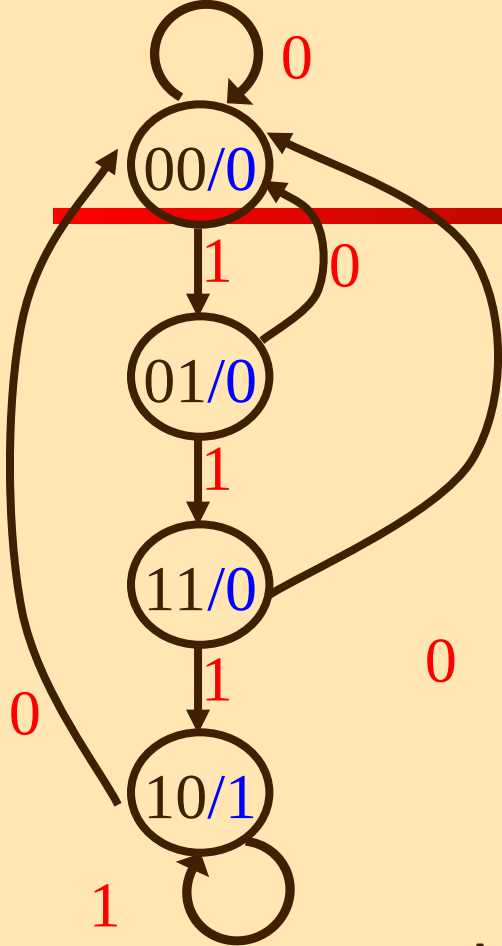
A & B - הפלט תלוי ב

Mealy - טבלת המצבים

A_t	B_t	$X=0$		$X=1$		$X=0$	$X=1$
		A_{t+1}	B_{t+1}	A_{t+1}	B_{t+1}	O_t	O_t
0	0	0	0	0	1	0	0
0	1	0	0	1	0	0	0
1	0	0	0	1	1	0	1
1	1	0	0	1	1	0	1

X - הפלט תלוי ב

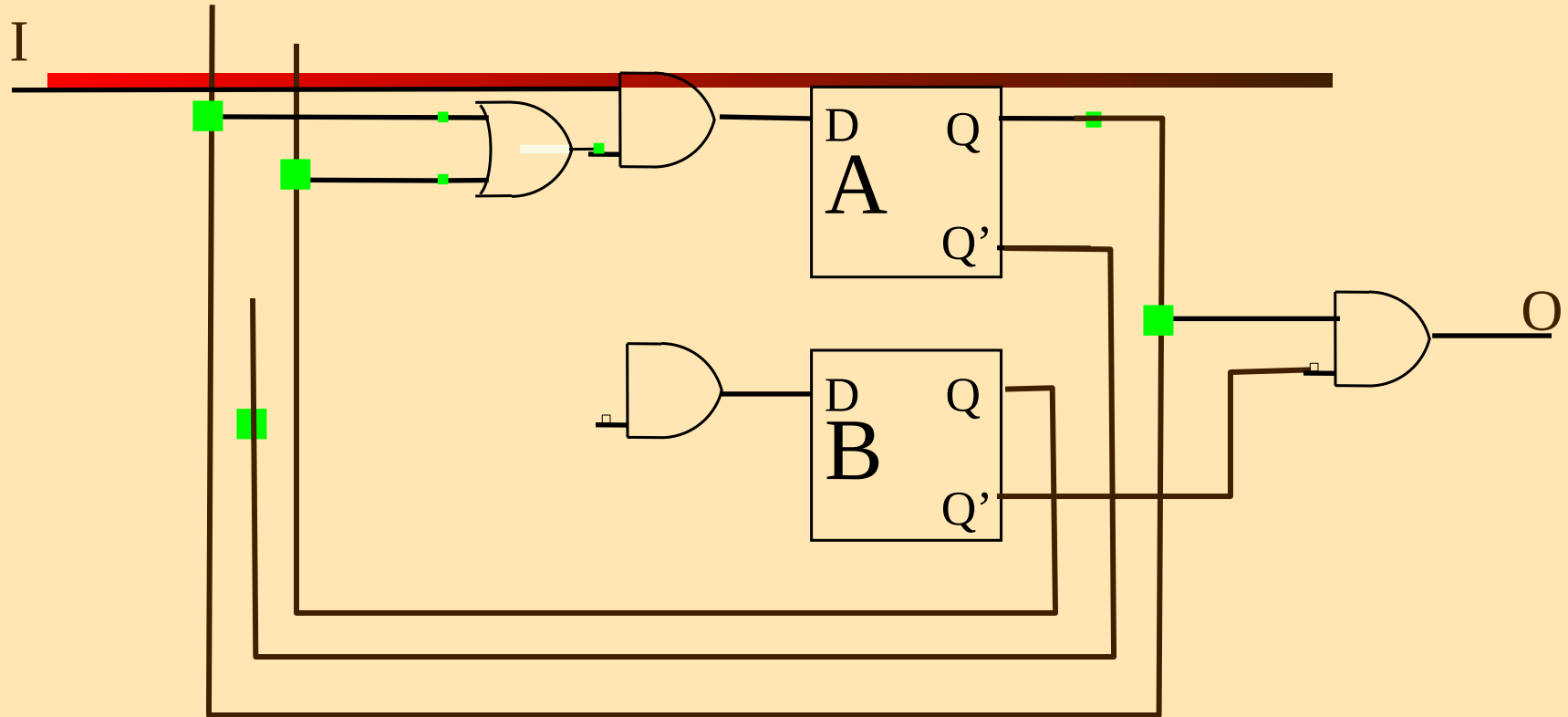
Moore - טבלת המצבים



		I=0		I=1		
		A_{t+1}	B_{t+1}	A_{t+1}	B_{t+1}	
A_t	B_t	A_{t+1}	B_{t+1}	A_{t+1}	B_{t+1}	O_t
0	0	0	0	0	1	0
0	1	0	0	1	1	0
1	1	0	0	1	0	0
1	0	0	0	1	0	1

AB						AB				
B_{t+1}	00	01	11	10		A_{t+1}	00	01	11	10
0					I {	0				
1						1				

דוגמא - Moore

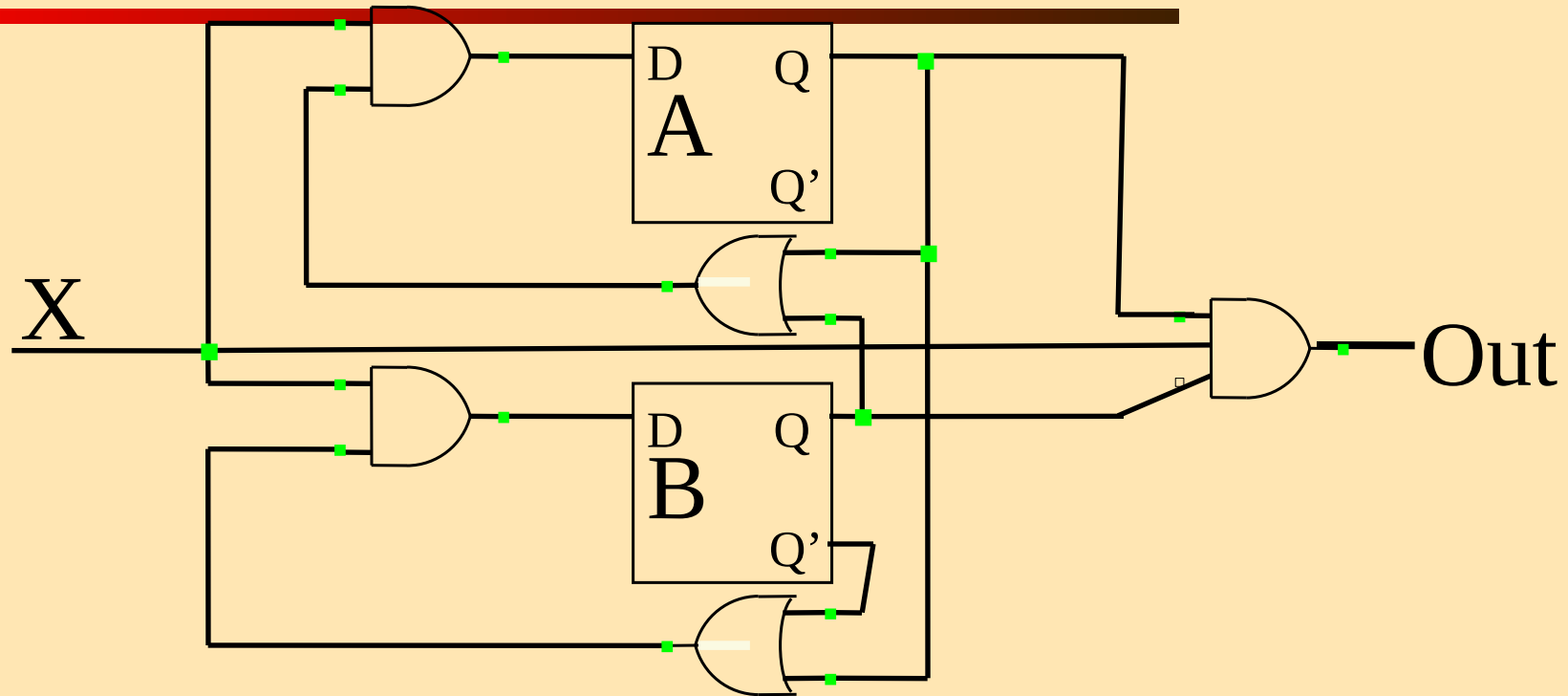


$$A_{t+1} = A * I + B * I = I(A + B)$$

$$B_{t+1} = \bar{A} * I$$

- כניסה אחת ויצאה אחת $O = A * \bar{B}$
- מצבים $4 \leftarrow \text{Data מסוג 2FF}$

דוגמא – Mealy



- כניסה אחת ויציאה אחת
- X , Q_A , Q_B - היציאה תלויה ב.
- מצבים $4 \Leftarrow \text{Data}$ מסוג 2FF.

Moore Vs. Mealy

פלט:

Moore – פונקציה של המצב לבד

Mealy – פונקציה של המצב והקלט

אוטומט:

Moore – הפלט "רשום" על המצב

Mealy – הפלט "רשום" על הקשת (מעבר)

שיקוליב:

Moore – $T_s + T_h$ מספיק שיהיה קבוע) לא תלוי ב"יציבות" הקלט

נוספים אם דרושה תלות היציאה בקלט FFs אך ידרש

Mealy – פשוט לממוש אם יש תלות של היציאה בקלט אך נדרשת יציבות.

(ולהפך) Mealy – שיקול ל Moore

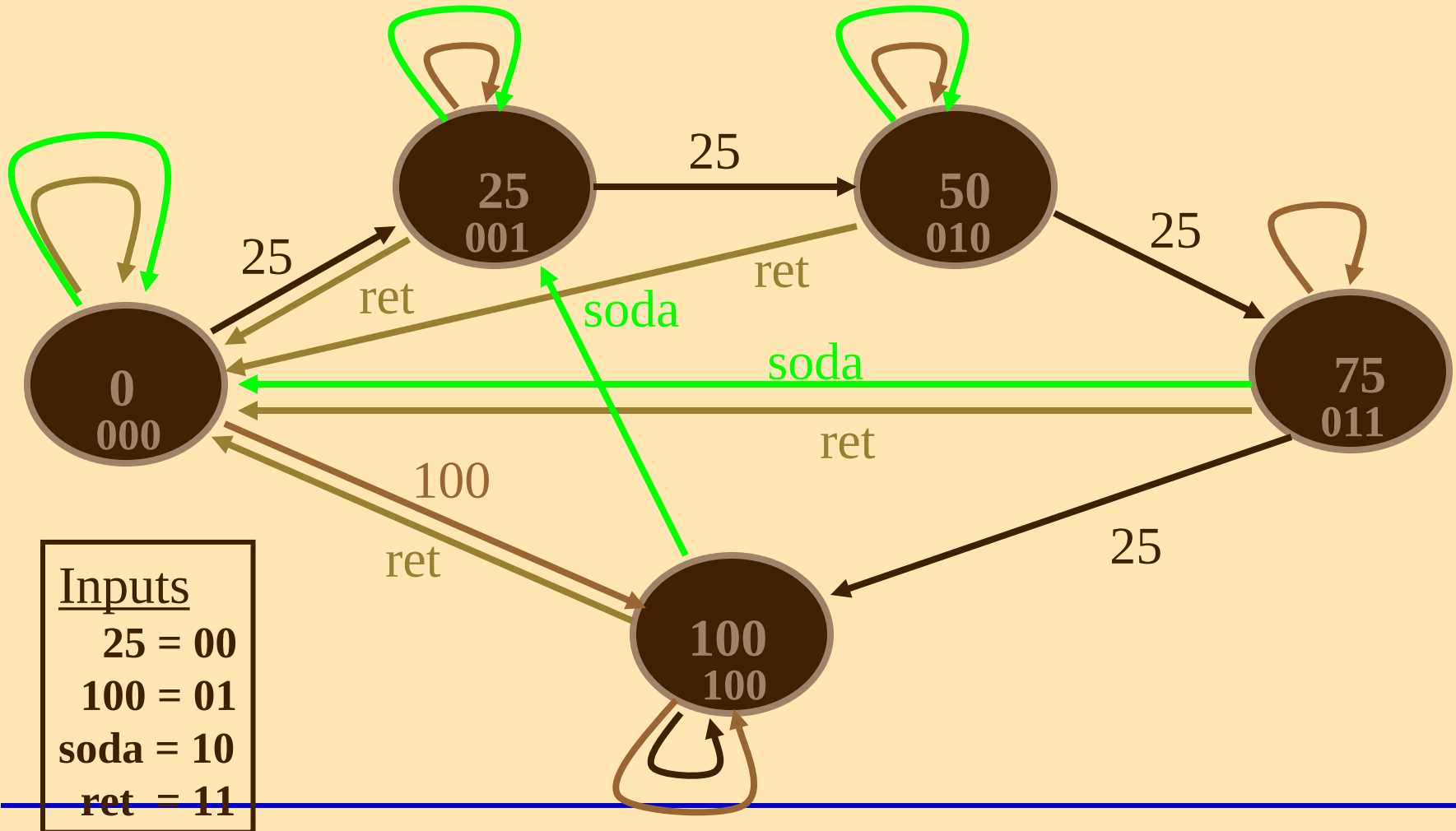
Example: Vending Machine

- Takes only quarters and dollar bills
- Won't hold more than \$1.00
- Sodas cost \$.75
- Possible actions (inputs)
 - deposit \$.25 (25)
 - deposit \$1.00 (\$)
 - push button to get soda (soda)
 - push button to get money returned (ret)

Example: Vending Machine

- State: description of the internal settings of the machine, e.g. how much money has been deposited and not spent
- Finite states: 0, 25, 50, 75, 100,
- Rules: determine how inputs can change state

Example: Vending Machine



Example: Vending Machine

state			input		new state		
S2	S1	S0	I0	I1	S2	S1	S0

0	0	0	0	0	0	0	1
0	0	1	0	0	0	1	0
0	1	0	0	0	0	1	1
0	1	1	0	0	1	0	0
1	0	0	0	0	1	0	0

0	0	0	0	1	1	0	0
0	0	1	0	1	0	0	1
0	1	0	0	1	0	1	0
0	1	1	0	1	0	1	1
1	0	0	0	1	1	0	0

state			input		new state		
S2	S1	S0	I0	I1	S2	S1	S0

0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	1	0	0	0	0
1	0	0	1	0	0	0	1

0	0	0	1	1	0	0	0
0	0	1	1	1	0	0	0
0	1	0	1	1	0	0	0
0	1	1	1	1	0	0	0
1	0	0	1	1	0	0	0

Example: Vending Machine

state			input		new state		
S2	S1	S0	I0	I1	S2	S1	S0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	1	0
0	1	0	0	0	0	1	1
0	1	1	0	0	1	0	0
1	0	0	0	X	1	0	0
0	0	0	0	1	1	0	0
0	0	1	0	1	0	0	1
0	1	0	0	1	0	1	0
0	1	1	0	1	0	1	1

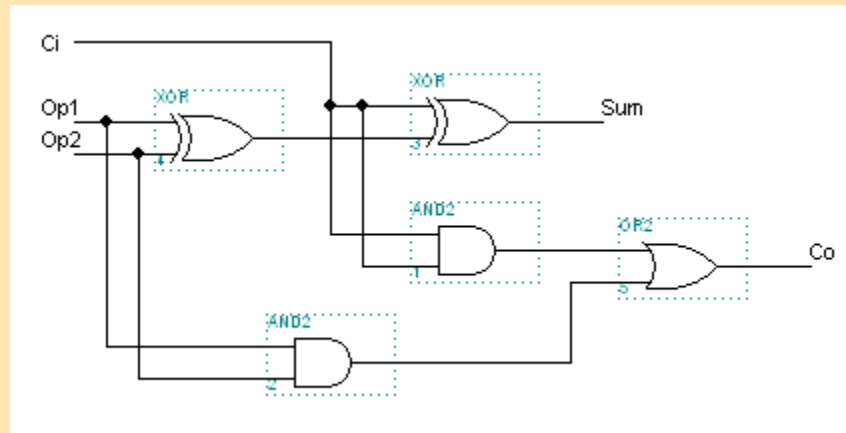
state			input		new state		
S2	S1	S0	I0	I1	S2	S1	S0
0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	1	0	0	0	0
1	0	0	1	0	0	0	1
X	X	X	1	1	0	0	0

What is VHDL?

- **V**ery High Speed Integrated Circuit
Hardware **D**escription **L**anguage
- Used to describe a desired logic circuit
- Compiled, Synthesized and Burned onto a working chip
- Simplifies hardware for large projects
- **Examples:** Combinatorial Logic, Finite State Machines

Let's Start Simple

- Combinatorial/Arithmetic Logic
- 1-bit full-adder



■ Three Approaches to VHDL
Programming: Structural, Arithmetic, and
Behavioral

Structural (I)

```
-- Defining one-bit fulladd component;  
-- taken from Brown/Uranesic, Fig. A4
```

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

```
ENTITY fulladd IS  
    PORT ( Cin, x, y  :IN  STD_LOGIC;  
          S, Cout    :OUT STD_LOGIC);  
END fulladd;
```

```
ARCHITECTURE LogicFunc OF fulladd IS  
BEGIN  
    S      <= x xor y xor Cin;  
    Cout   <= (x and y) or (x and Cin) or (y and Cin);  
END LogicFunc;
```

Included Libraries: Used in compiling and synthesis. The same for each project.

Entity Declaration: Indicates what comes in and what goes out.

Architecture Declaration: Defines the entity on a functional level.

Structural (II)

- Structurally defined code assigns a logical function of the inputs to each output
- This is most useful for simple combinatorial logic

```
ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
    S    <= x xor y xor Cin;
    Cout <= (x and y) or (x and Cin) or (y and Cin);
END LogicFunc;
```

Arithmetic

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

ENTITY adder IS
    PORT (op1, op2      : IN  INTEGER;
          result        : OUT INTEGER);
END adder;

ARCHITECTURE maxpld OF adder IS
BEGIN
    result <= CONV_INTEGER(op1 + op2);
END maxpld;
```

- Arithmetic Operation allows for simpler code, but possibly at the expense of chip real estate.
- What is wrong with this code? Think about how the integers are implemented by the synthesizer.

Arithmetic (II)

```
ENTITY adder IS
    PORT (op1, op2      : IN  INTEGER RANGE 0 to 7;
          result        : OUT INTEGER RANGE 0 to 15);
END adder;
```

- If you choose to code on a higher level, be sure to specify ranges for your variables, otherwise Altera will assume 32-bit unsigned values.
- There is not enough room on the whole chip to store one 32-bit value.

Behavioral

- Describe how the circuit works is meant to work and let the synthesizer work out the details.
- This is most useful for Finite State Machines and programs involving sequential statements and processes. We'll see some examples shortly.

Bringing Components Together

- You can design several different “circuits” in Altera and then bring them together to form a larger design on a single chip.
- Two methods:
 - Code Directly via the Netlist
 - Altera Tools Graphical Editor

Structural Netlist

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

ENTITY adder4struct IS
    PORT ( Cin      :IN  STD_LOGIC;
          x3,x2,x1,x0 :IN  STD_LOGIC;
          y3,y2,y1,y0 :IN  STD_LOGIC;
          s3,s2,s1,s0 :OUT STD_LOGIC;
          Cout       :OUT STD_LOGIC);
END adder4struct;

ARCHITECTURE Structure OF adder4struct IS
    SIGNAL c1,c2,c3 : STD_LOGIC;
    COMPONENT fulladd
        PORT (Cin, x, y :IN  STD_LOGIC;
              s, Cout  :OUT STD_LOGIC);
    END COMPONENT;
BEGIN
    stage0: fulladd PORT MAP(Cin,x0,y0,s0,c1);
    stage1: fulladd PORT MAP(c1,x1,y1,s1,c2);
    stage2: fulladd PORT MAP(c2,x2,y2,s2,c3);
    stage3: fulladd PORT MAP(
        Cin => c3, Cout=> Cout, x => x3, y => y3, s => s3 );
END Structure;
```

Using our Full Adder code from earlier. . .

- Each stage is made up of a full adder component.
- The fulladd code from earlier is also part of this vhdl file, it is not shown here.
- The carry out from each stage is assigned as carry in to the next stage.
- Notice that c1, c2, c3 are internal signals written in to allow transfer of data between the stages.
- This is important because you cannot specify an output pin of a component as an input pin in the same entity. c1, c2, and c3 are like buffers.

Syntax Notes and Helpful Hints

- Don't forget semi-colons where necessary
- Top level entity and filename must be the same
- If you design a smaller “circuit” to be part of a larger project, it is worthwhile for you to test that small piece to ensure that it functions as you intend it to.
- More is often less. Be specific about your code and the synthesizer will reward you with ample chip space.

Finite State Machines (FSMs)

- What is an FSM?

- Two types:
 - Moore
 - Mealy

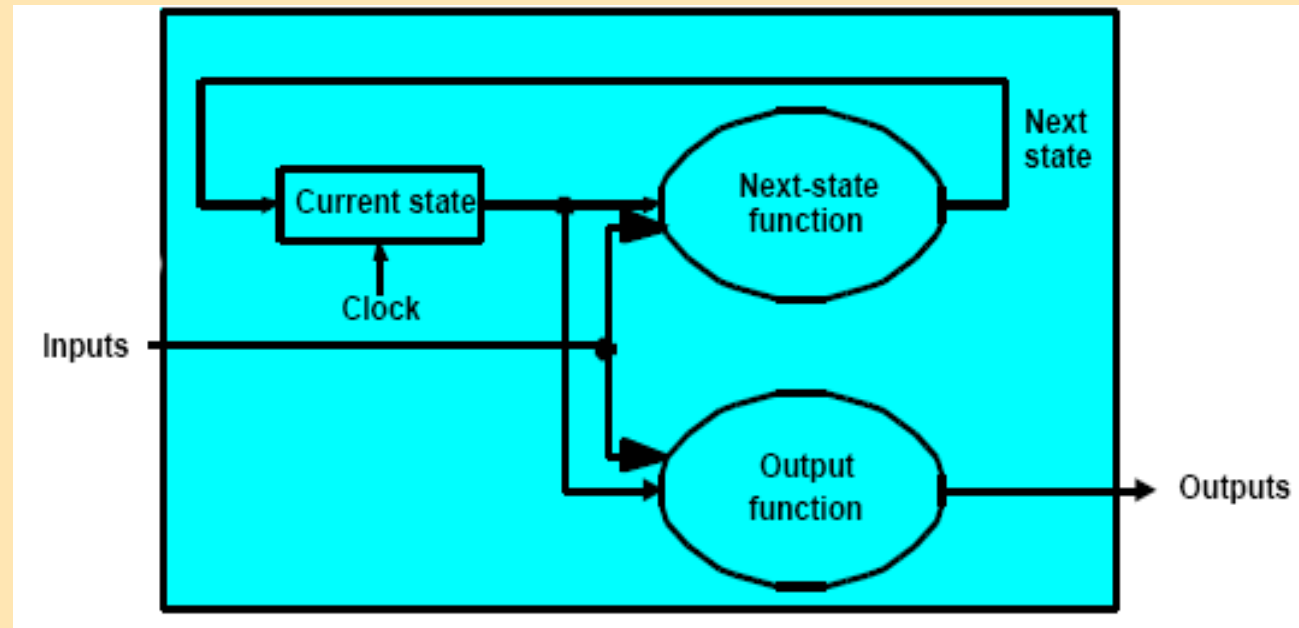
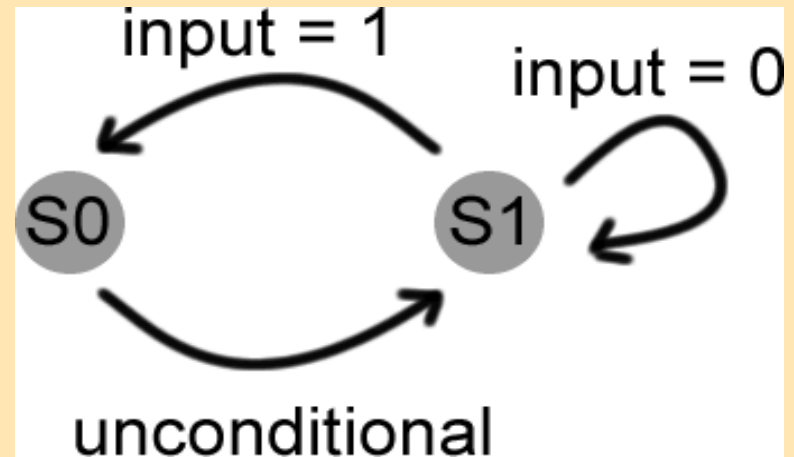


Figure B.27

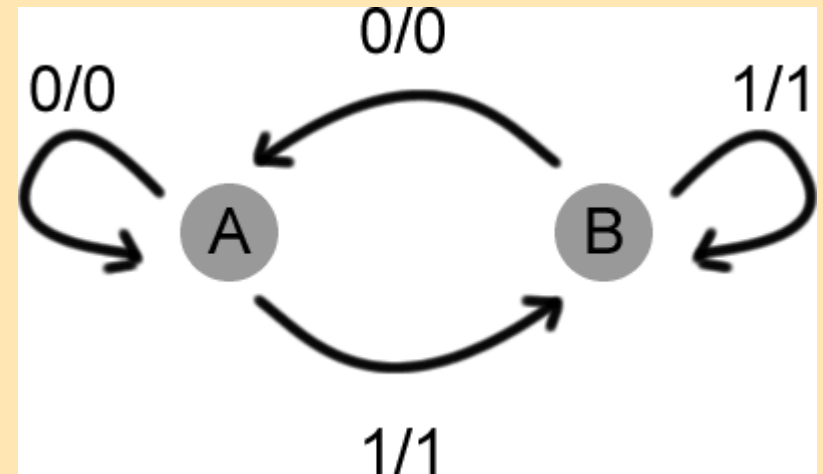
Moore FSM

- Output depends **ONLY** on current state
- Outputs associated with each state are set at clock transition



Mealy FSM

- Output depends on inputs AND current state
- Outputs are set during transitions



Coding FSMs in Altera

```
-- Mealy FSM, single-process description
```

```
ENTITY fsm_mealy1 IS
```

```
PORT (Clock, Resetn  :IN  STD_LOGIC;
```

```
      w              :IN  STD_LOGIC;
```

```
      z              :OUT STD_LOGIC);
```

```
END fsm_mealy1;
```

```
ARCHITECTURE Behavior OF fsm_mealy1 IS
```

```
    TYPE State_type IS (A,B);
```

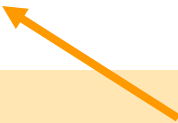
```
    SIGNAL y : State_type;
```

```
BEGIN
```

Process Statement

- Process computes outputs of sequential statements on each clock tick with respect to the sensitive signals.

```
-- Single Process: sensitive to Resetn, Clock  
-- Affects states and output (change during transition)  
PROCESS (Resetn, Clock)  
  BEGIN
```



Sensitivity list

'EVENT

- 'EVENT is an Altera construct that represents when the signal is transitioning

```
IF (Clock'EVENT AND Clock = '1') THEN
  CASE y IS
    WHEN A =>
      IF w = '0' THEN
        y <= A;
        z <= '0';
```

IF statement reads:

If Clock is making a positive transition THEN

VHDL codes for FSM

- Mealy FSM – see *mealy1.vhd* on the web
- Moore FSM - see *moore.vhd* on the web
- Now let's take a look how to edit, compile, simulate and synthesize your design using Altera software
- (proceed with hands on tutorial)

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

FSM States (1)

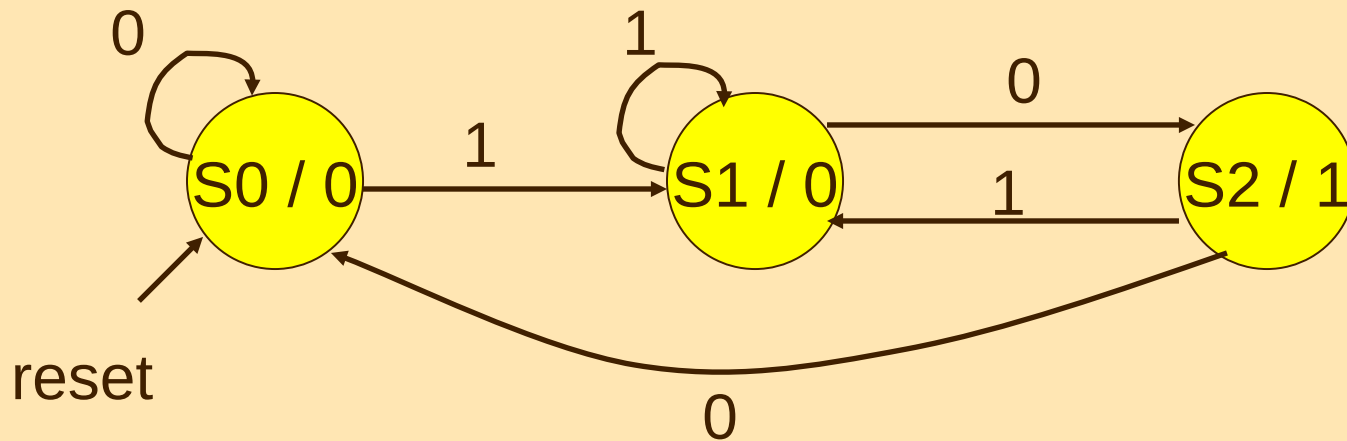
```
architecture behavior of FSM is
    type state is (list of states);
    signal FSM_state: state;
begin
    process(clk, reset)
    begin
        if reset = '1' then
            FSM_state <= initial state;
        else
            case FSM_state is
                •
                •
                •
```

FSM States (2)

```
case FSM_state is
  when state_1 =>
    if transition condition 1 then
      FSM_state <= state_1;
    end if;
  when state_2 =>
    if transition condition 2 then
      FSM_state <= state_2;
    end if;
    .
    .
    .
end case;
end if; end process;
```

Moore FSM - Example 1

- Moore FSM that Recognizes Sequence 10



Moore FSM in VHDL

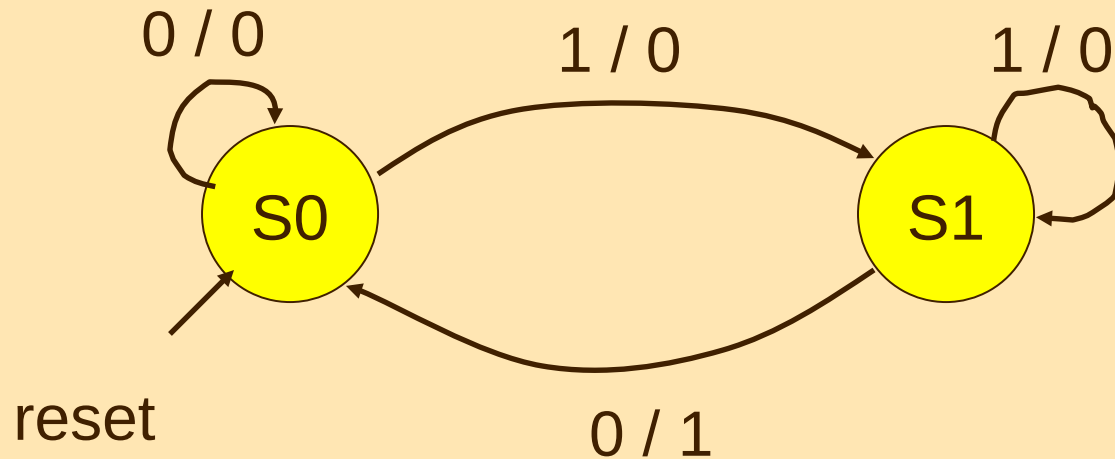
```
type state is (S0, S1, S2);
signal Moore_state: state;

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; end if;
            when S1 =>
                if input = '0' then Moore_state <= S2; 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



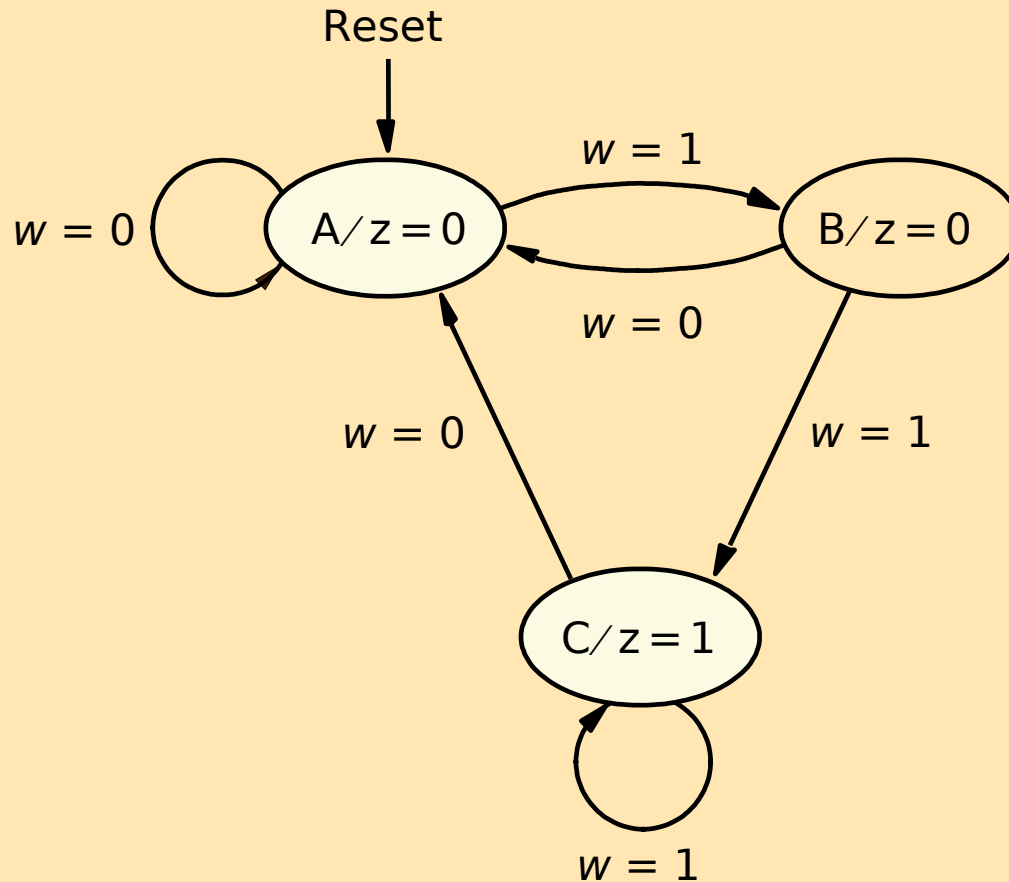
Mealy FSM in VHDL

```
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; end if;
            when S1 =>
                if input = '0' then Mealy_state <= S0; end if;
        end case;
    end if;
End process;

Output <= '1' when (Mealy_state = S1 and input = '0') else '0';
```

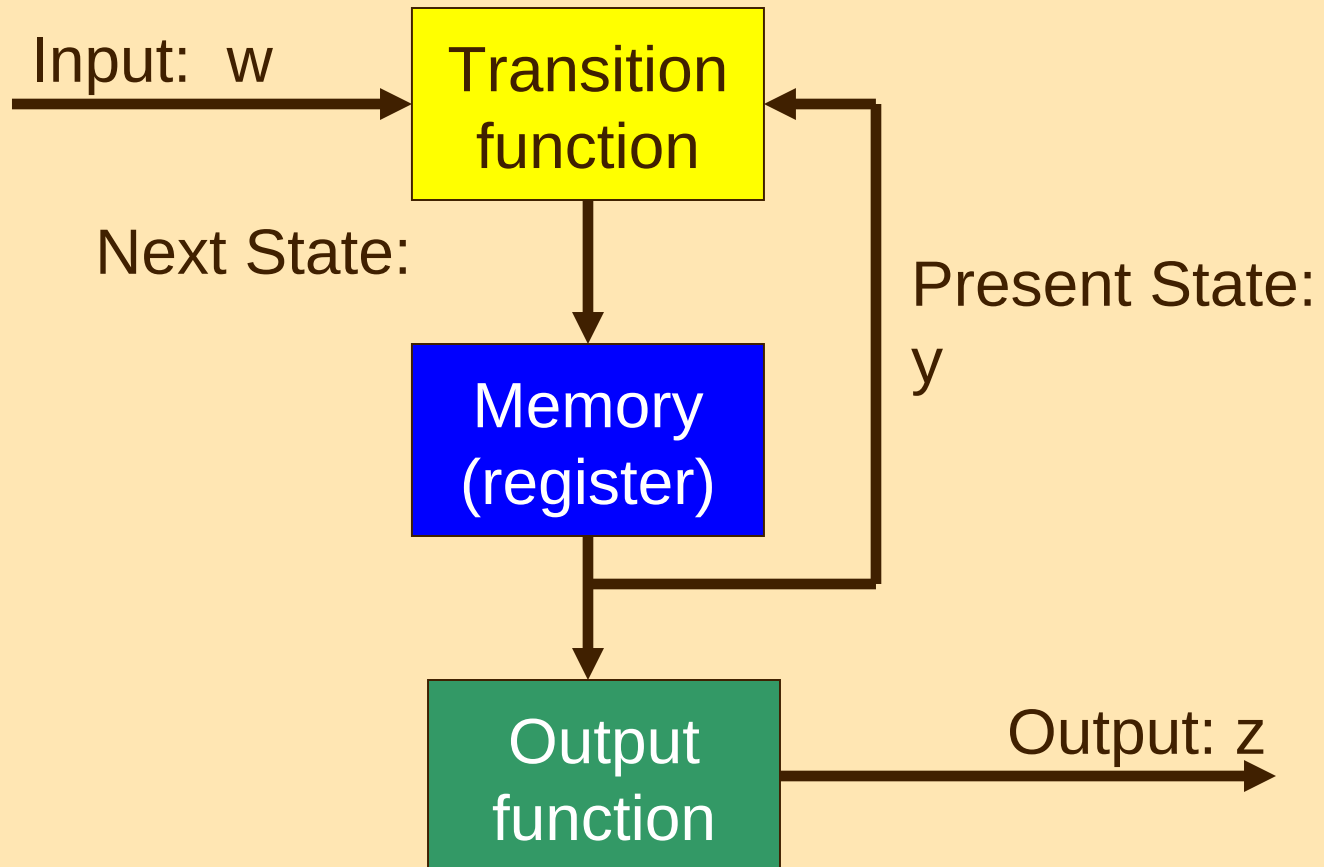
Moore FSM – Example 2: State diagram



Moore FSM – Example 2: State table

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

Moore FSM



Moore FSM – Example 2: VHDL code (1)

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

```
ENTITY simple IS
```

```
    PORT ( Clock, Resetn, 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 ;
```

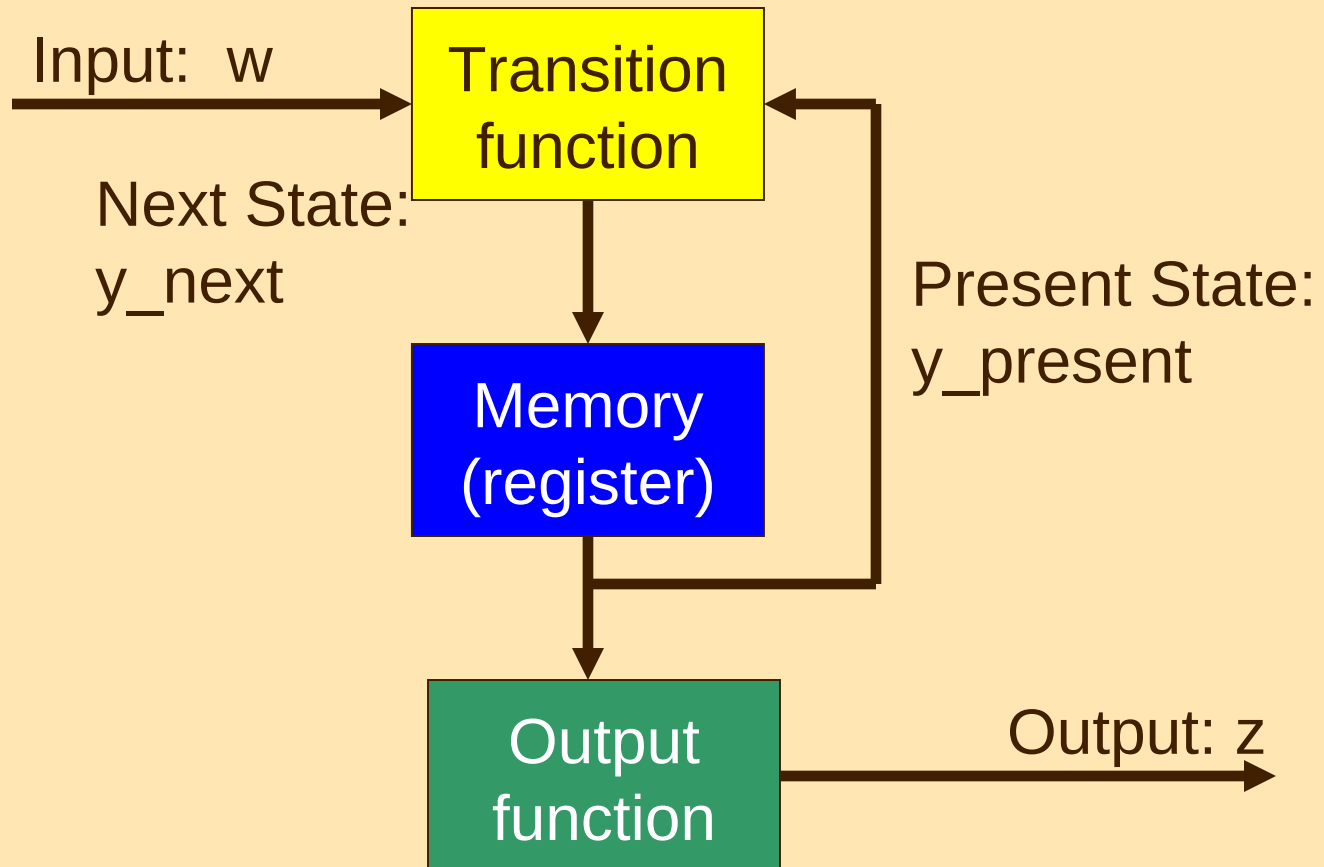
```
        ELSIF (Clock'EVENT AND Clock = '1') THEN
```

```
con't ...
```

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 ;
END IF ;
END PROCESS ;
z <= '1' WHEN y = C ELSE '0' ;
END Behavior ;
```

Moore FSM



Alternative VHDL code (1)

```
ARCHITECTURE Behavior OF simple IS
    TYPE State_type IS (A, B, C) ;
    SIGNAL y_present, y_next : State_type ;
BEGIN
    PROCESS ( w, y_present )
    BEGIN
        CASE y_present IS
            WHEN A =>
                IF w = '0' THEN
                    y_next <= A ;
                ELSE
                    y_next <= B ;
                END IF ;
            WHEN B =>
                IF w = '0' THEN
                    y_next <= A ;
                ELSE
                    y_next <= C ;
                END IF ;
```

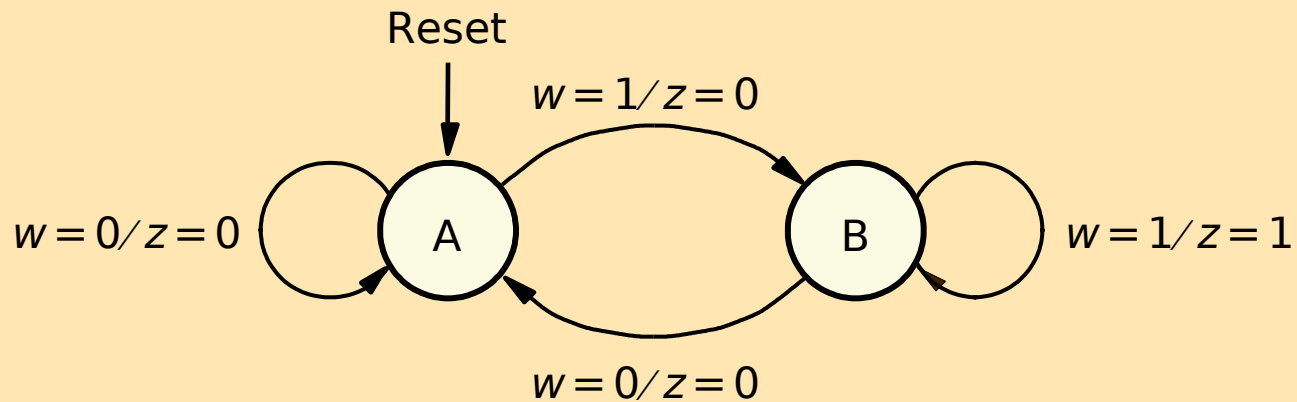
Alternative VHDL code (2)

```
        WHEN C =>
            IF w = '0' THEN
                y_next <= A ;
            ELSE
                y_next <= C ;
            END IF ;
        END CASE ;
    END PROCESS ;
```

```
PROCESS (Clock, Resetn)
BEGIN
    IF Resetn = '0' THEN
        y_present <= A ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
        y_present <= y_next ;
    END IF ;
END PROCESS ;
```

```
z <= '1' WHEN y_present = C ELSE '0' ;
END Behavior ;
```

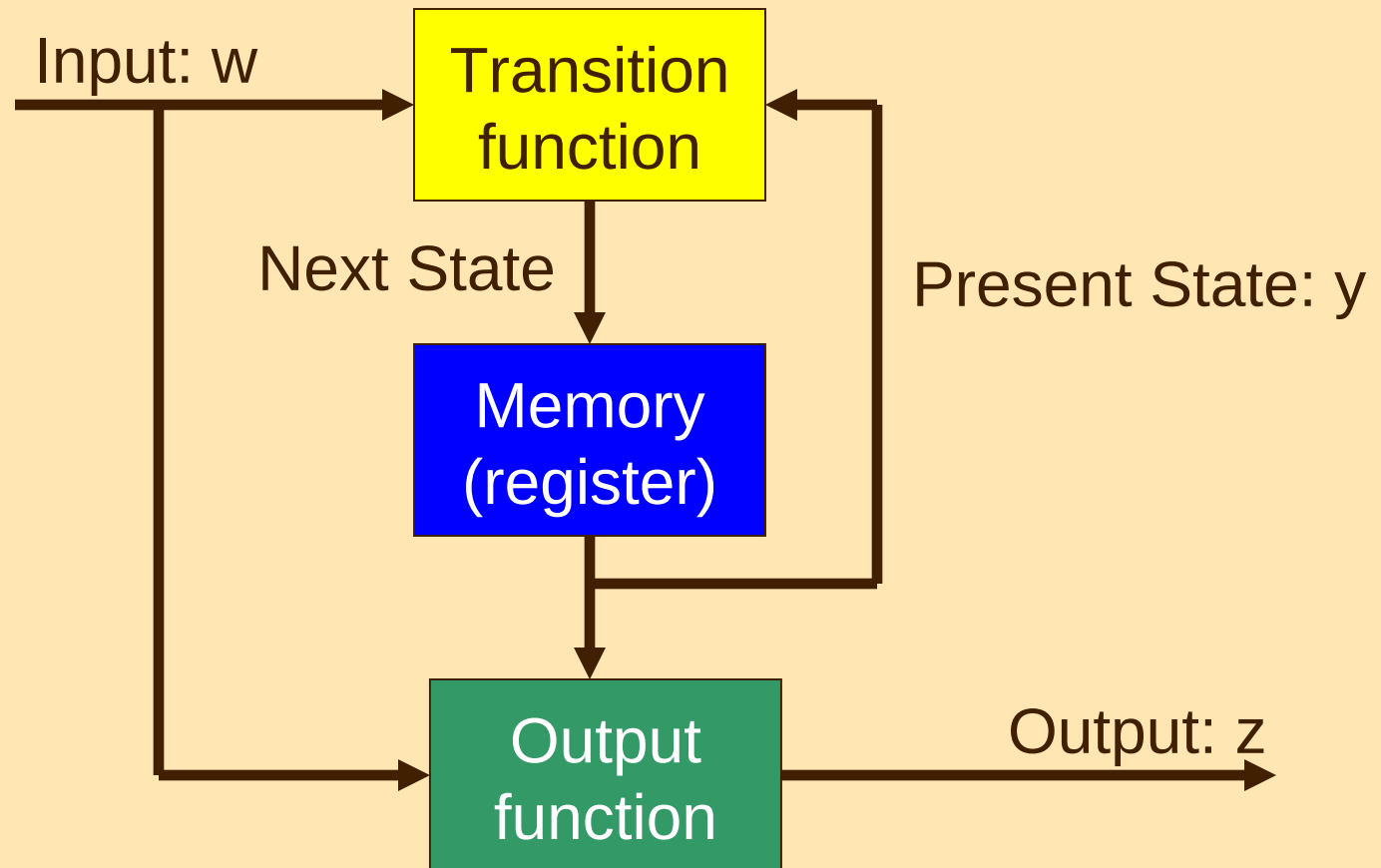
Mealy FSM – Example 2: State diagram



Mealy FSM – Example 2: State table

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

Mealy FSM



Mealy FSM – Example 2: VHDL code (1)

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

ENTITY mealy IS
    PORT ( Clock, Resetn, 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
            CASE y IS
                WHEN A =>
                    IF w = '0' THEN y <= A ;
                    ELSE y <= B ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Mealy FSM – Example 2: VHDL code (2)

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

with y select
    z <= w when B,
    z <= '0' when others;

END Behavior ;
```

State Encoding Problem

- State Encoding Can Have a Big Influence on Optimality of the FSM Implementation
 - No methods other than checking all possible encodings are known to produce optimal circuit
 - Feasible for small circuits only
- Using Enumerated Types for States in VHDL Leaves Encoding Problem for Synthesis Tool

Types of State Encodings (1)

- Binary (Sequential) – States Encoded as Consecutive Binary Numbers
 - Small number of used flip-flops
 - Potentially complex transition functions leading to slow implementations
- One-Hot – Only One Bit Is Active
 - Number of used flip-flops as big as number of states
 - Simple and fast transition functions
 - Preferable coding technique in FPGAs

Types of State Encodings (2)

State	Binary Code	One-Hot Code
S0	000	10000000
S1	001	01000000
S2	010	00100000
S3	011	00010000
S4	100	00001000
S5	101	00000100
S6	110	00000010
S7	111	00000001

A user-defined attribute for manual state assignment

(ENTITY declaration not shown)

ARCHITECTURE Behavior OF simple IS

TYPE State_type IS (A, B, C) ;

ATTRIBUTE ENUM_ENCODING : STRING ;

ATTRIBUTE ENUM_ENCODING OF State_type : TYPE IS "00 01 11"

SIGNAL y_present, y_next : State_type ;

BEGIN

con't ...

Figure 8.34

Using constants for manual state assignment (1)

ARCHITECTURE Behavior OF simple IS

SUBTYPE ABC_STATE is STD_LOGIC_VECTOR(1 DOWNT0 0);

CONSTANT A : ABC_STATE := "00" ;

CONSTANT B : ABC_STATE := "01" ;

CONSTANT C : ABC_STATE := "11" ;

SIGNAL y_present, y_next : ABC_STATE;

BEGIN

PROCESS (w, y_present)

BEGIN

CASE y_present IS

WHEN A =>

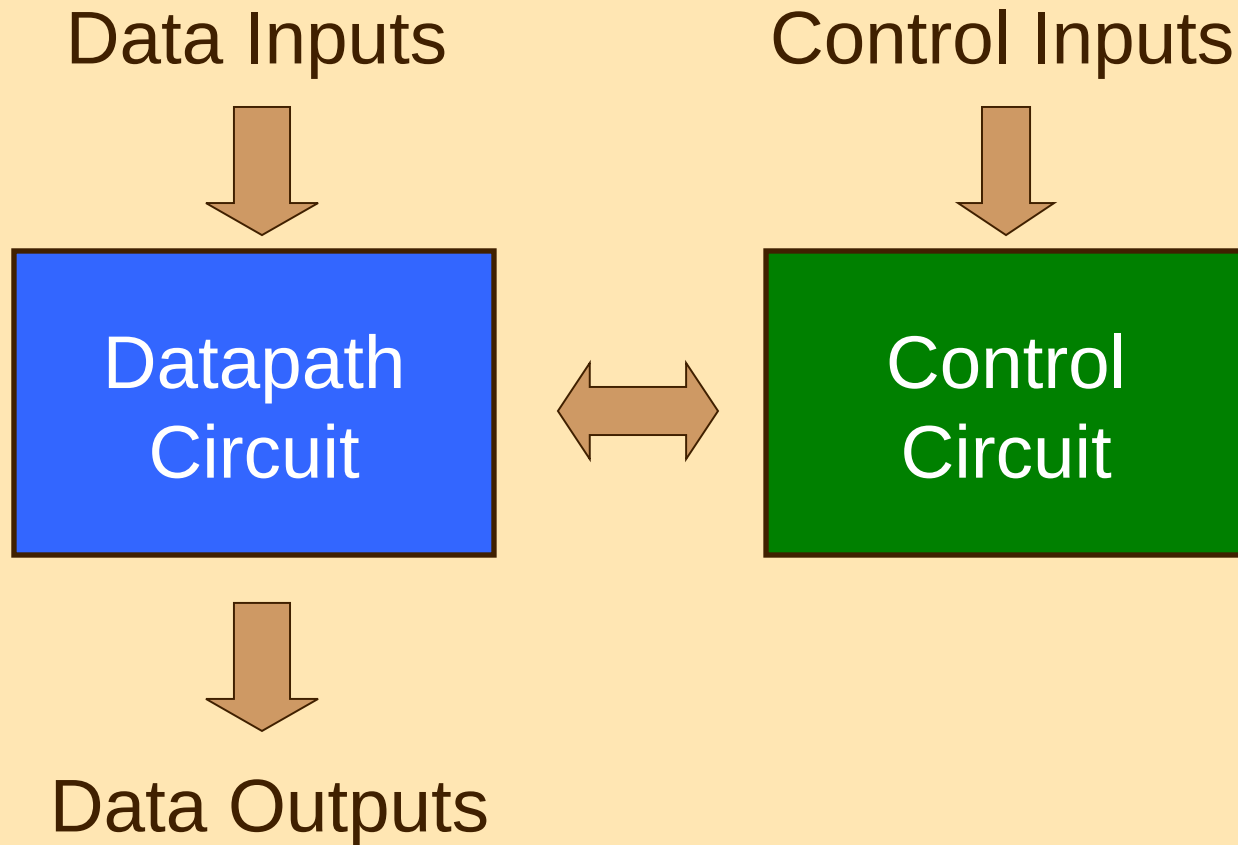
IF w = '0' THEN y_next <= A ;

ELSE y_next <= B ;

END IF ;

... con't

RTL Design Components



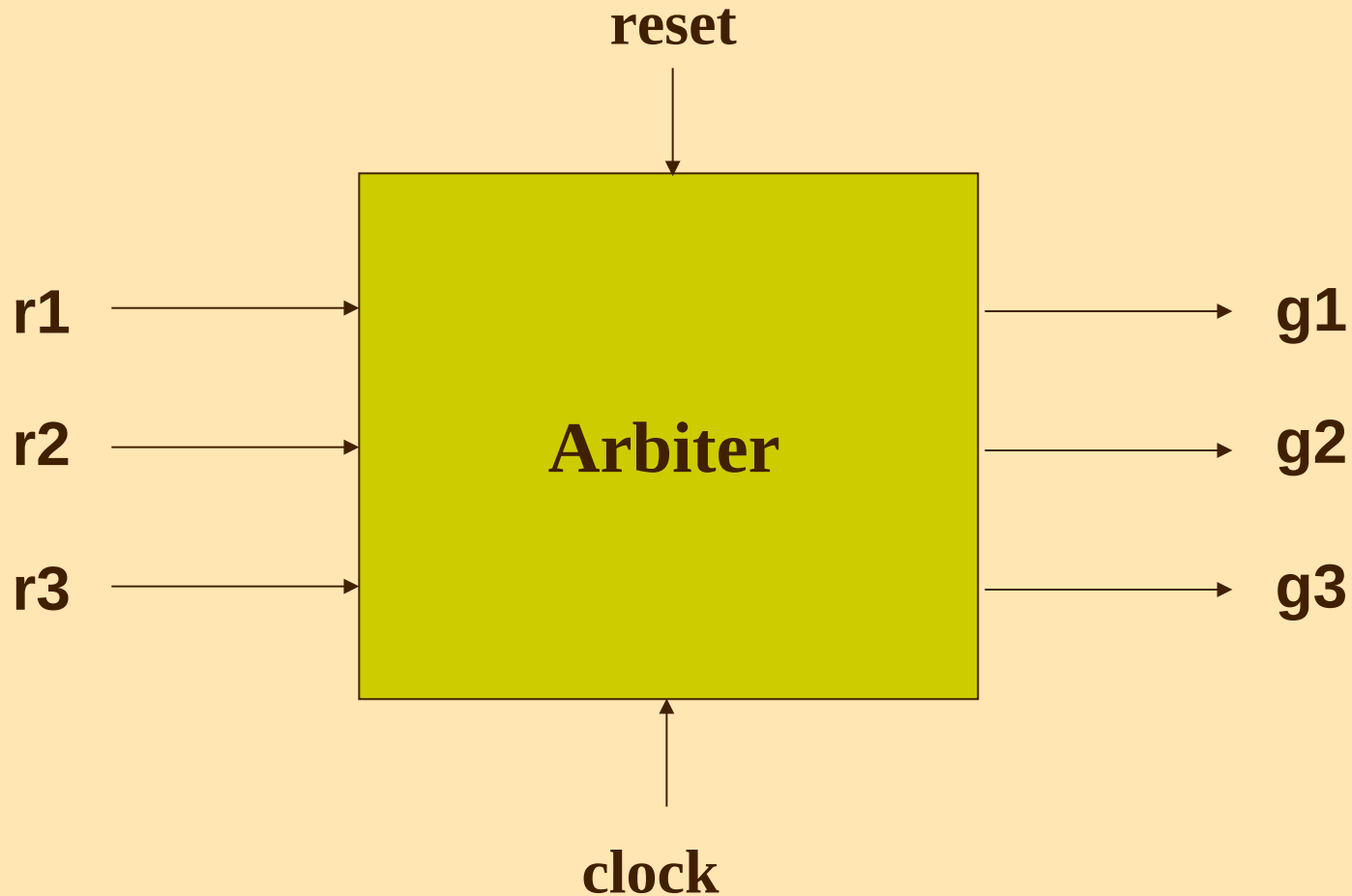
Datapath Circuit

- Provides All Necessary Resources and Interconnects Among Them to Perform Specified Task
- Examples of Resources
 - Adders, Multipliers, Registers, Memories, etc.

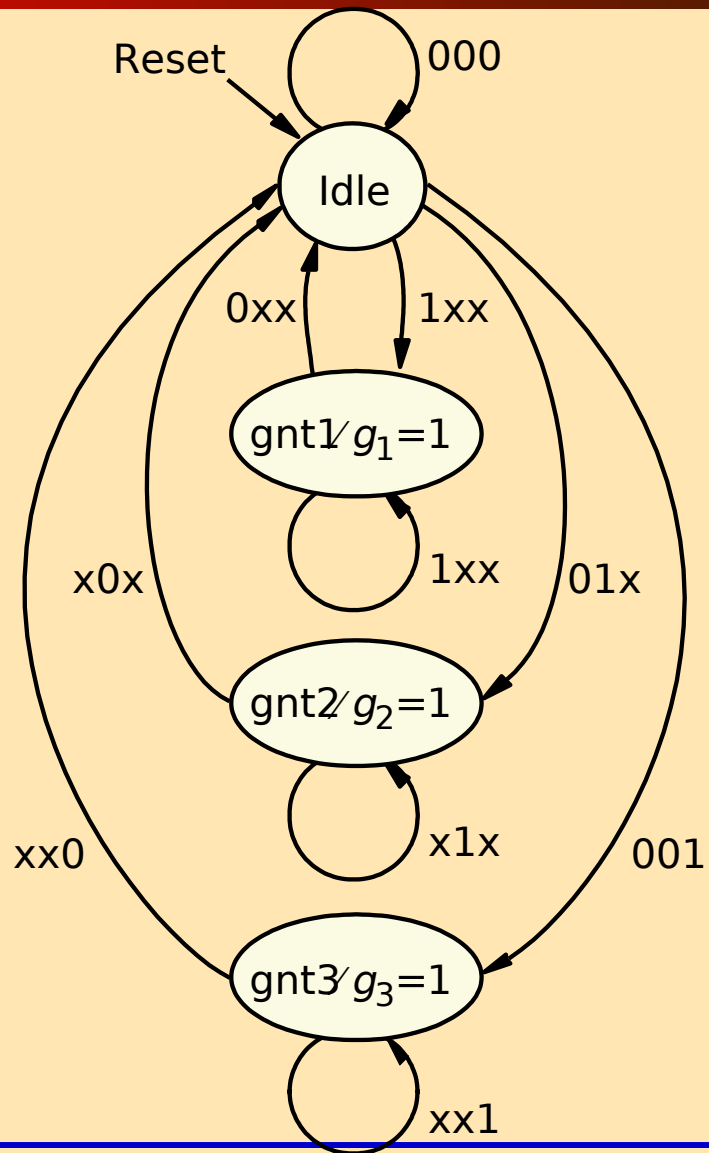
Control Circuit

- Controls Data Movements in Operational Circuit by Switching Multiplexers and Enabling or Disabling Resources
- Follows Some 'Program' or Schedule
- Usually Implemented as FSM

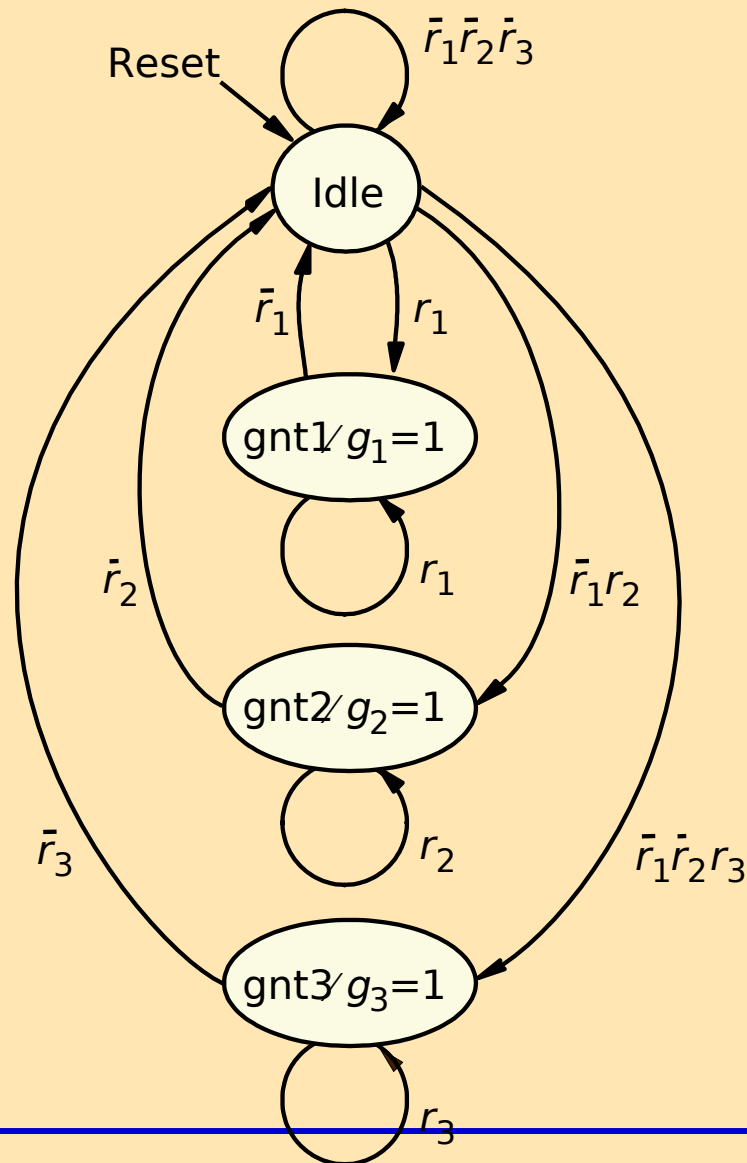
Control Unit Example: Arbiter (1)



Control Unit Example: Arbiter (2)



Control Unit Example: Arbiter (3)



Arbiter – 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 ;

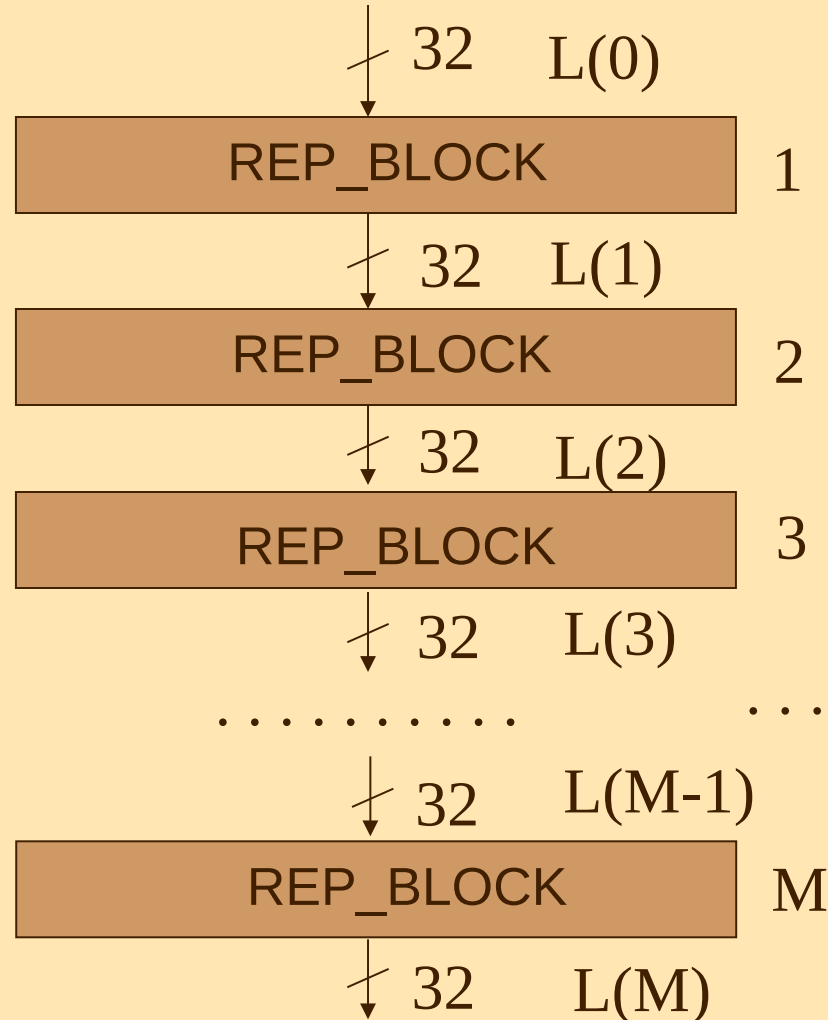
ARCHITECTURE Behavior OF arbiter IS
    TYPE State_type IS (Idle, gnt1, gnt2, gnt3) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN y <= Idle ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            CASE y IS
                WHEN Idle =>
                    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 ;
            END CASE ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Arbiter – VHDL code (2)

```
        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 ;
        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 ;
```

Questions?

Arrays of std_logic_vectors



Arrays of std_logic_vectors

```
type sig_array is array(0 to M) of std_logic_vector(31 downto 0);  
...  
signal L: sig_array;  
...  
begin  
  L(0) <= A;  
  CASCADE: for I in 1 to M generate  
    C: REP_BLOCK  
      port map(REP_IN => L(I-1),  
              REP_OUT=>L(I));  
  end generate;  
  Z <= L(M);  
end;
```