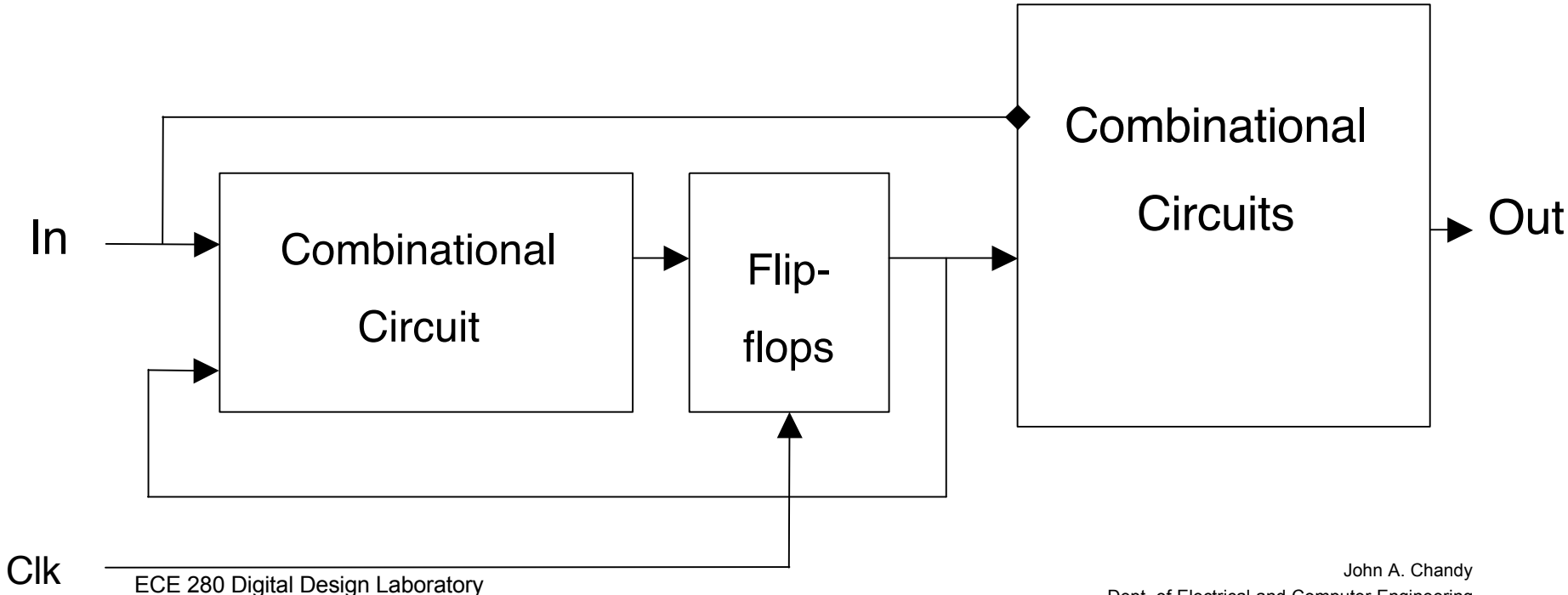
ECE 280 / CSE 280
Digital Design Laboratory
Lecture 3

State Machines

Synchronous Sequential Circuits

- Synchronous sequential logic circuits are realized using combinational logic and flip-flops.
- Block diagram of generic sequential circuit



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Sequential Circuits

- A set of inputs {in} and produces a set of outputs {out}.
- The values of the outputs {Q} of the flip-flops are referred to as states of the circuit.
- Under control of the clock signal *clk*, the flip-flop outputs change their state as determined by the combinational logic that feeds the inputs on these flip-flops. Thus the circuit moves from one state to another.
- To ensure that only one transition from one state to another takes place during one clock cycle, the flip-flops have to be edge- triggered type. They can be triggered by a low-to-high or high-to-low clock transitions.
- An active clock edge refers to the clock edge that causes transitions.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Moore Versus Mealy Type Sequential Circuits

- **Moore** Type Sequential Circuits – are sequential circuits whose outputs depend only on the state of the circuit (*Edward Moore, 1950)*
- **Mealy** Type Sequential Circuits – are sequential circuits whose output depend on both the states and the primary inputs (*George Mealy, 1950).*
- *Sequential Circuits are also called finite state machines (FSMs). The name derives from the fact that the functional behavior of these circuits can be represented using a finite number of states.*

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Design Steps

(1) Obtain the specification of the desired circuit.

(2) Derive the states for the machine by first selecting a starting state. Then, given the specification of the circuit, consider all valuations of the inputs to the circuit and create new states as needed for the machine to respond to inputs.

(3) Create a state diagram to keep track of the states as they are visited. When completed, the state diagram shows all states in the machine and gives the conditions under which the circuits move from one state to another.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Design Steps – Cont.

(4) Create a state table from the state diagram.

(5) Minimize the number of states using appropriate minimization procedure.

(6) Decide the number of state variables needed to represent all states and perform the state assignment.

Note: There are many different state assignment possible for a given sequential circuit. Some assignments may be better than others.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Design Steps – Cont.

(7) Choose the type of flip-flops to be used in the circuit. Derive the next-state logic expressions to control the inputs of the flip-flops and then derive logic expressions for the outputs of the circuit.

(8) Implement the circuit as indicated by the logic expressions.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

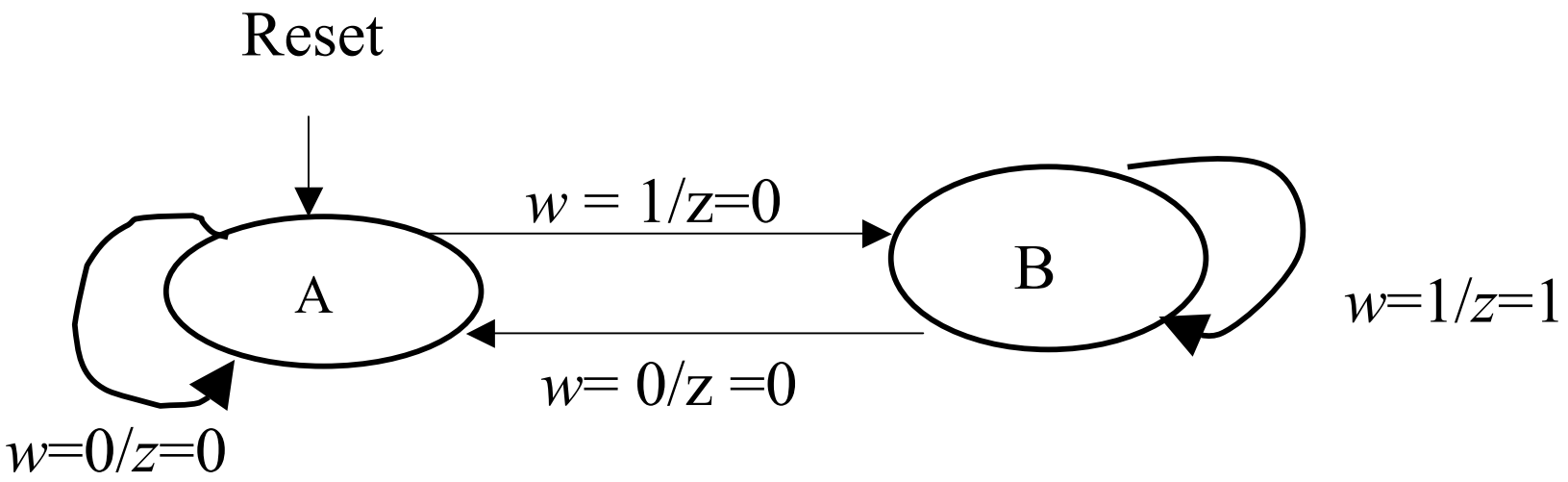
Mealy-type FSM

- Design a circuit that meets the following specification:
  a) The circuit has one input, *w*, and one output *z*.
  b) All state changes in the circuit occur on the positive edge of a clock signal.
  c) If during two immediately preceding clock cycles the input *w* was equal to 1. The output *z* is equal to 1, and occurs in the same clock cycle as the second occurrence of *w* = 1. Otherwise, the value of *z* is equal to zero as shown in the table below

Clock cycle:	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
w :	0	1	0	1	1	0	1	1	1	0	1
z :	0	0	0	0	1	0	0	1	1	0	0

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

State Diagram



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

State Diagram

- State A
  - As long as the input *w*=0, the machine should remain in state A, producing an output *z*=0. Shown by the arc that loops back into A.
- State B
  - When w=1, the machine has to move to a new state, B, to record the fact that an input of 1 has occurred. The output stays at 0 on the move to B, as shown by the arc *w=1/z=0* from A to B.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

State Diagram

- State B
  - If *w*=1 when the machine is in state B, the machine should still stay in state B and produce an output *z*=1 as indicated by the arc *w=1/z=1.*
- State B to State A
  - If *w*=0, *z* should be set back asynchronously to 0, and the machine should move back to state A at the next active edge of the clock.
- A Moore machine implementation would require a third state to indicate the state when the output is 1

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

State Table for FSM

Present State	*w*	Next State	*z*
A	0	A	0
A	1	B	0
B	0	A	0
B	1	B	1

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

State Assigned Table for FSM

Present State	*w*	Next State	*z*
0	0	0	0
0	1	1	0
1	0	0	0
1	1	1	1

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

VHDL Implementation for Finite State Machine

- State Processes
- State Coding
- FSM Types
  - Moore
  - Mealy
  - Registered Output

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

VHDL model for Finite State Machine

```
ENTITY simple_FSM  IS
   PORT ( clk, reset    :  IN STD_LOGIC;
               w        :  IN STD_LOGIC;
               z        :  OUT STD_LOGIC);
 END simple_FSM;

ARCHITECTURE RTL of simple_FSM IS
    TYPE state_type IS ( state_A, state_B, state_C);
    SIGNAL state : state_type;

BEGIN
    PROCESS ( reset, clk )
    BEGIN                          -- Define reset state and next
    -- combinational process       -- transitions using a case
    END PROCESS;                   -- statement based on current state.
    -- output process              -- Define state machine outputs
END;
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

One-Process FSM

```
FSM_FF: process (CLK, RESET)
begin
if RESET='1' then                    -- reset state
       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 ;
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Two-Process FSM

```
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 ;
```

The VHDL source code contains two processes. The logic for the NEXT_STATE calculation is described in a separate process.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

How many processes?

- One process:
  - FSM states change with special input changes
    => One-process more comprehensible
- Two-process:
  - Error detection easier with two-process.
  - Two-process can lead to smaller generic netlist and therefore to better synthesis result.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

State Encoding

- The synthesis tool itself will transform the state names into a binary description on his own.

```
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**

Most synthesis tools select a binary code by default

Speed optimized default encoding: **one hot**

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Hand Coding

```
subtype STATE_TYPE is
std_ulogic_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 ;
```

- The best method of state encoding is hand coding, i.e. the designer decides by himself which code will be use.
- Defining constants
- Control of encoding
- Safe FSM
- Simulatable
- Portable design among different synthesis tool
- More effort

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Moore-type FSM

- 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 ;
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

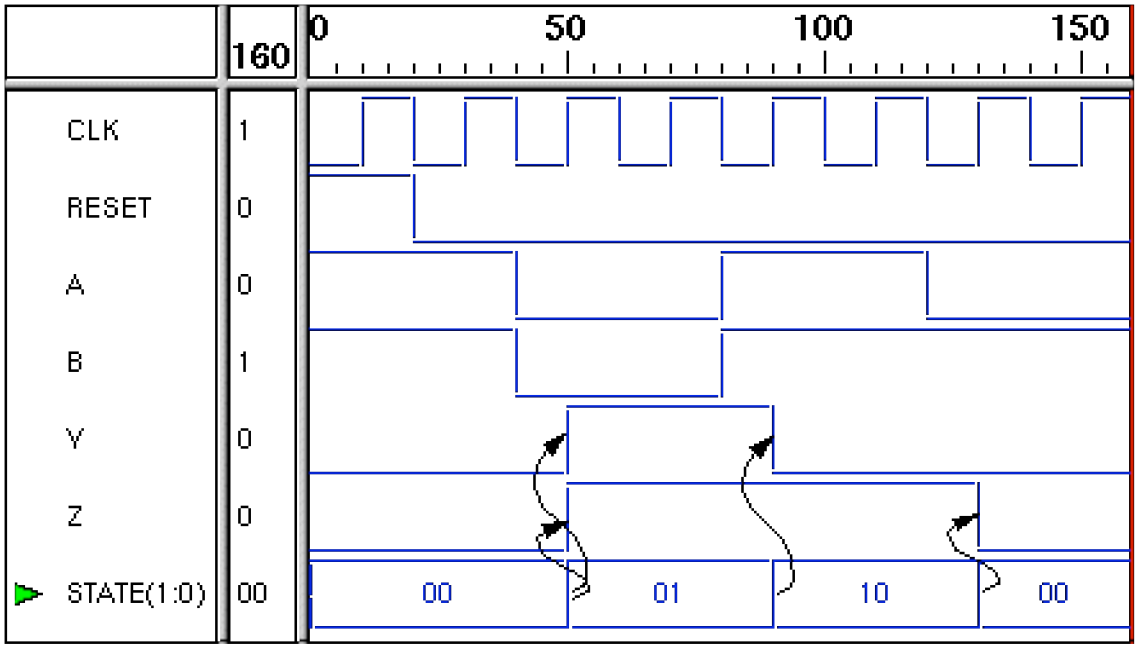
Moore example

```
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 ;
   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 ;
```

The output logic is not contained in a combinational process because of space limitation. Instead, it is implemented via separate concurrent signal assignments.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Waveform Moore Example



The values of output vector (Y,Z) change simultaneously with the values of the STATE vector => Moore machine

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Mealy-type FSM

- 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 ;
```

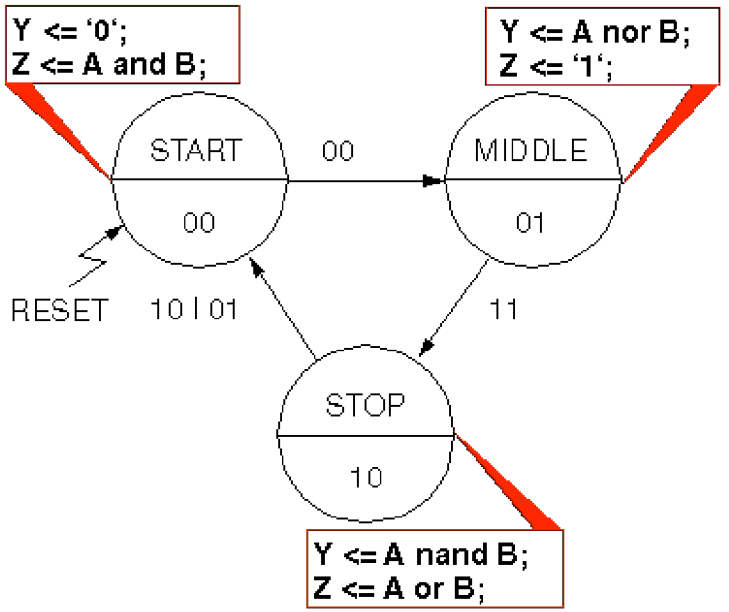
Two Processes

```
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 ;
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Mealy Example

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

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 ;
```

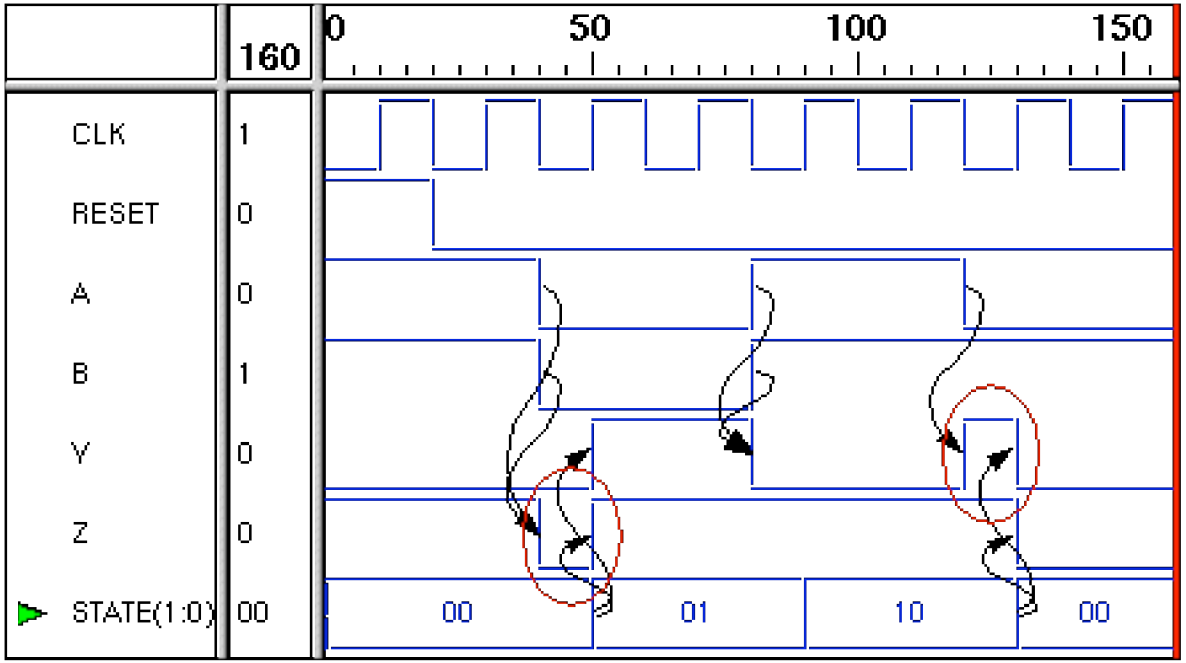



```
subtype STATE_TYPE is std_ulogic_vector(1 downto 0);
constant START    : STATE_TYPE := "00";
constant MIDDLE   : STATE_TYPE := "01";
constant STOP     : STATE_TYPE := "10";
```

The input signals appear on the right side of the assignments

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Waveform Mealy Example

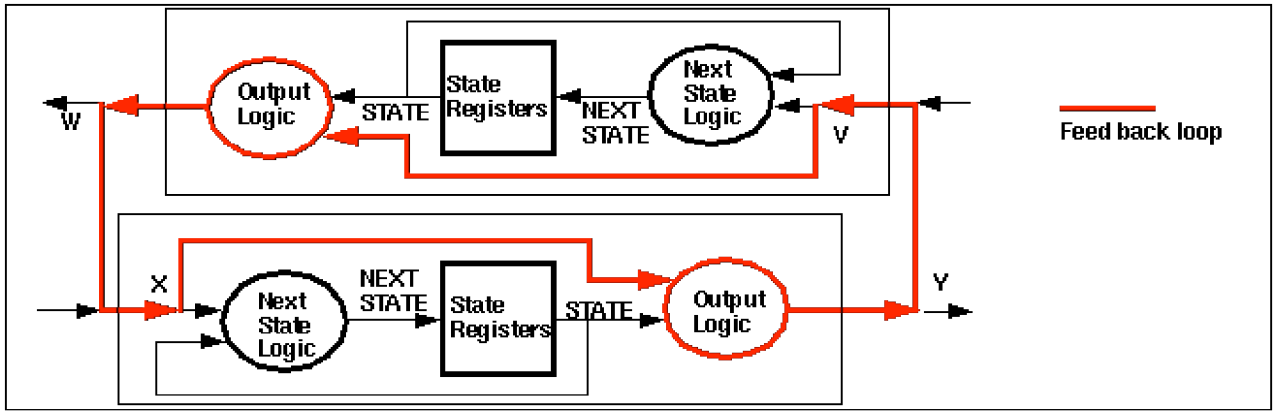


- Output (Y,Z) changes with input => Mealy machine
- Note the "spikes" of Y and Z in the waveform
  - This can lead to unexpected triggering of subsequent blocks
  - FSM has to be modeled carefully in order to avoid spikes in normal operation

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Modeling Aspects

- **Moore** is preferred because of safe operation
  - The new values are stable long before the next active clock edge occurs and spikes are avoided.
- **Mealy** is more flexible, but danger of
  - Spikes
  - Unnecessary long paths (maximum clock period)
  - If two Mealy machines are connected in a row then there is the danger of combinational feedback loops.



John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Registered Output

- As shown before, combinational feedback loops can occur if two Mealy machines are connected in a row. These loops can be avoided if the outputs are "clocked", i.e. if the outputs are connected to Flip Flops. With this, the feedback loop is broken up.
  - Avoiding long paths and uncertain timing
- Two versions are known for clocking the output.
  - With one additional clock period
  - Without additional clock period (Mealy)

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

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: . . . -- combinational process

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 ;
```

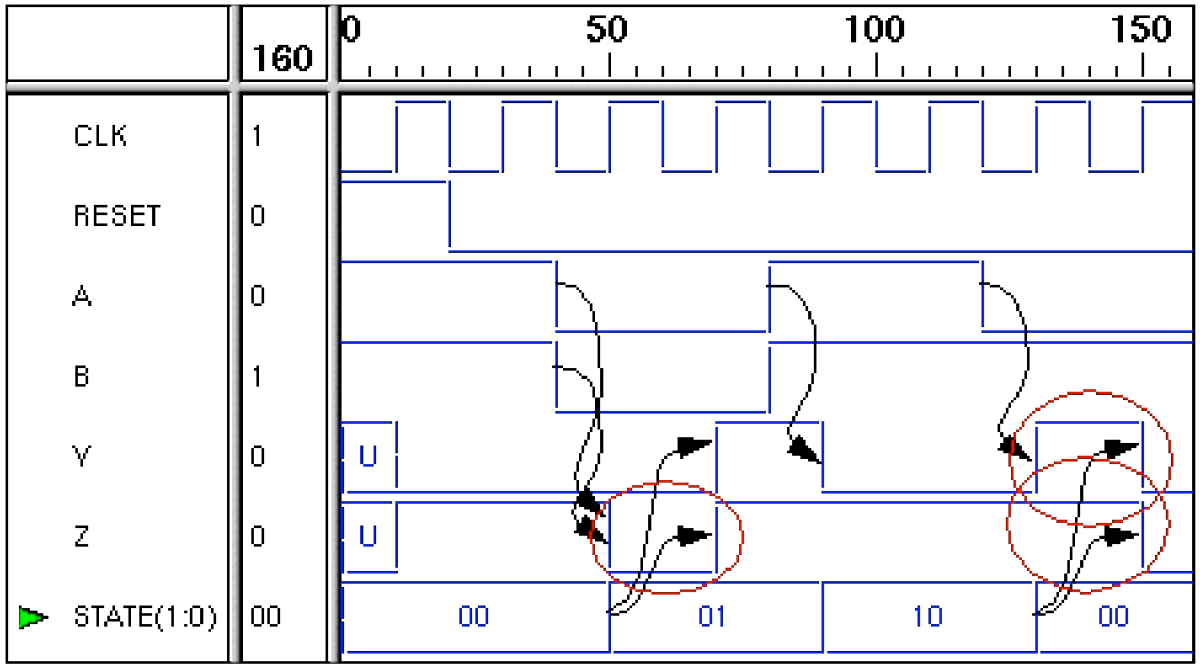

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Registered Output Example (1)

- As transitions take place only when an active clock edge occurs, it is clear that a value is assigned to the output signals with every active clock edge, i.e. Flip Flops have to be provided for the outputs.
- As the signal assignments of the old state (which will be exited) are connected to the transitions, the output values depend only on the current (old) state (STATE) but not on the new state (NEXT_STATE). Generally, the signal assignments can be hidden again behind the states in the graphical diagram. For this, so called in-state actions for the self loops and exit-actions for the transitions are used.
- In the VHDL source code, intermediate signals (*Y_I, Z_I*) are evaluated. The values of these signals depend on the input values and the current state. In the following clocked process, the intermediate signals are connected to Flip Flops.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

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 o)
- All value changes of intermediate signals occur only synchronously to the clock so that the state machine has a fixed and well known behavior.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

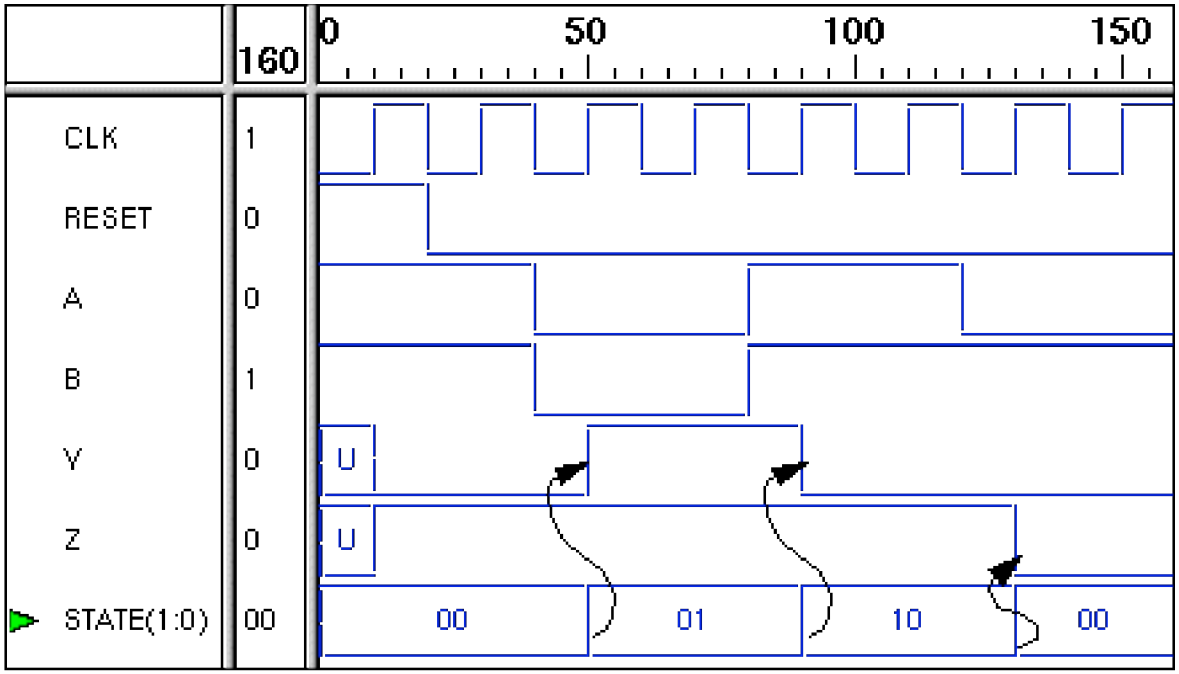
Registered Output Example (2)

```
architecture RTL of REG_TEST2 is
  signal Y_I , Z_I : std_ulogic ;
  signal STATE,NEXTSTATE : STATE_TYPE ;
begin
  REG: · · · -- clocked STATE process
  CMB: · · · -- combinational process

  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 ;
```

- The values of the intermediate signals are calculated from the values of the current inputs and the current successor state. Again, Flip Flops are inferred for the intermediate signals with another process.

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Waveform Registered Output Example (2)



- The output values change synchronously with the state changes now and undesired temporary values are eliminated.
- No delay between STATE and output changes.
- "Spikes" of original Mealy machine are gone!

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Testbench

- Simple testbench responses can be analyzed by waveform inspection
- Sophisticated testbenches may require more complicated verification techniques
  - Can take >50% of project resources

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Structure of a VHDL Testbench

```
entity TB_TEST is
end TB_TEST;

architecture BEH of TB_TEST is
  -- component declaration of UUT
  -- internal signal definition
begin
  -- component instantiation of UUT
  -- clock and stimuli generation
  wait for 100 ns;
  A <= 0;
  CLK <= 1;
  …
end BEH;

configuration CFG_TB_TEST of TB_TEST is
   for BEH;
       -- customized configuration
   end for;
end CFG_TB_TEST;
```

- Declaration of the UUT
- Connection of the UUT with testbench signals
- Stimuli and clock generation (behavioral modeling)
- Response analysis
- A configuration is used to pick the desired components for simulation
  - May be a customized configuration for testbench simulation

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Simple Testbench Example

```
entity ADDER is
   port (A,B : in bit;
      CARRY,SUM : out bit);
end ADDER;


architecture RTL of ADDER is
begin
  ADD: process (A,B)
  begin
     SUM <= A xor B;
     CARRY <= A and B;
  end process ADD;
end RTL;
```

```
entity TB_ADDER IS -- empty entity is defined
end TB_ADDER;      -- No need for interface

architecture TEST of TB_ADDER is
   component ADDER
     port (A, B: in bit;
           CARRY, SUM: out bit);
   end component;
   signal A_I, B_I, CARRY_I, SUM_I : bit;

begin
  UUT: ADDER port map(A_I, B_I, CARRY_I, SUM_I);

  STIMULUS: process
    begin
      A_I <= '1'; B_I <= '0'; wait for 10 ns;
      A_I <= '1'; B_I <= '1'; wait for 10 ns;
      -- and so on ...
    end process STIMULUS;
end TEST;

configuration CFG_TB_ADDER of TB_ADDER is
      for TEST
      end for;
end CFG_TB_ADDER;
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Testbench Clock

```
process
begin
   wait for 20 ns;
   CLOCK_LOOP : loop
      clk <= '0';
      wait for 10 ns;
      clk <= '1';
      wait for 10 ns;
   end loop CLOCK_LOOP;
end process
```


John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut

Lab 3

- Design a simple state machine to handle a one-item vending machine
- Things to remember:
  - **Buttons can cause multiple pulses, so they must be debounced**
  - **The button/coin pulse can be arbitrarily long**

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut