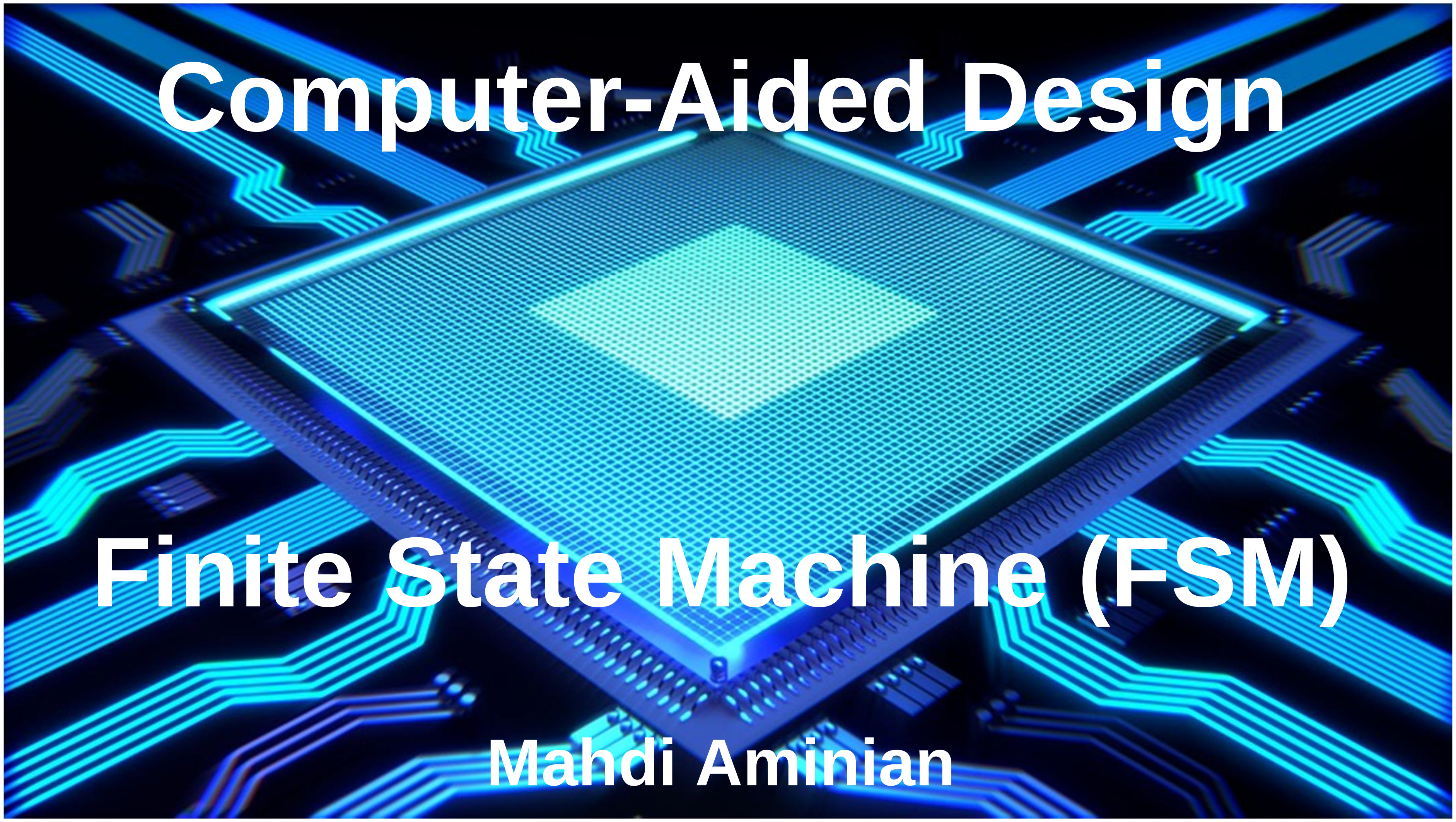


Computer-Aided Design

Finite State Machine (FSM)

Mahdi Aminian

The University Of Guilan

Rasht - Iran

Some slides courtesy of:

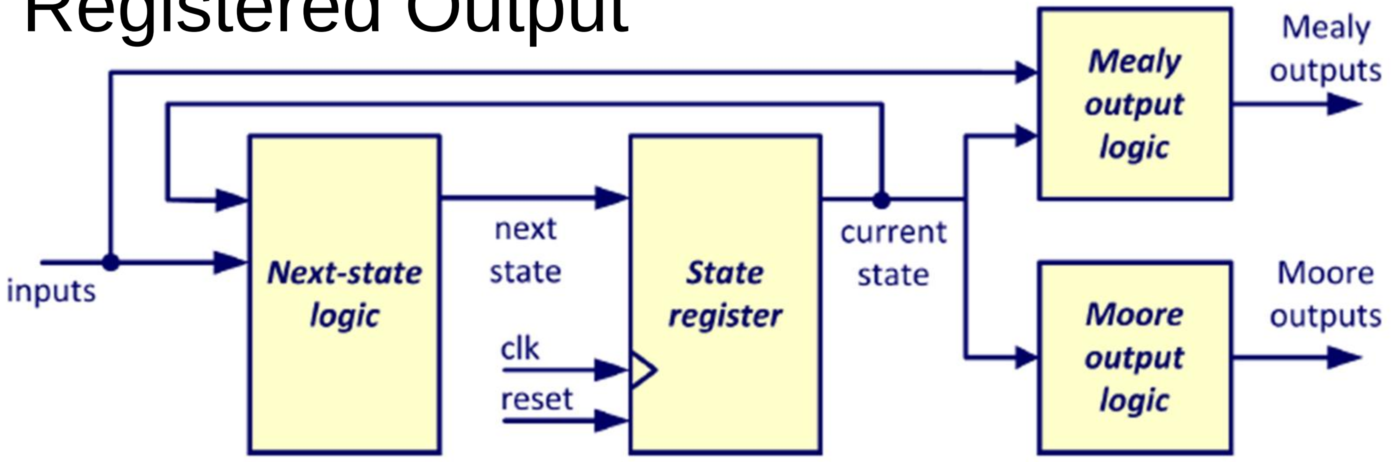
- “Hardware Systems Modeling”, A. Vachoux, EPFL
- CAD slides from Dr. Saheb Zamani

FSM

- FSM ?
- FSM Design ?
- FSM Implementation ?

Finite State Machines & VHDL

- **One- , two- or three-processes**
- **State Encoding**
- **FSM Types**
  - Medvedev
  - Moore
  - Mealy
  - Registered Output



Enumeration Types

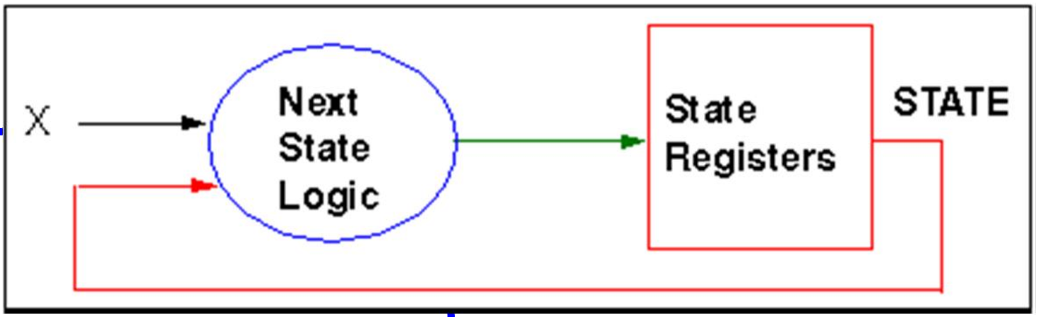
```
architecture EXAMPLE of ENUMERATION is

   type T_STATE is (RESET, START, EXECUTE, FINISH);

  signal CURRENT_STATE, NEXT_STATE : T_STATE ;
  signal TWO_BIT_VEC : bit_vector(1 downto 0);

begin

  -- valid signal assignments
  NEXT_STATE       <= CURRENT_STATE;
  CURRENT_STATE <= RESET;

  -- invalid signal assignments
  CURRENT_STATE <= "00";
  CURRENT_STATE <= TWO_BIT_VEC;

end EXAMPLE;
```

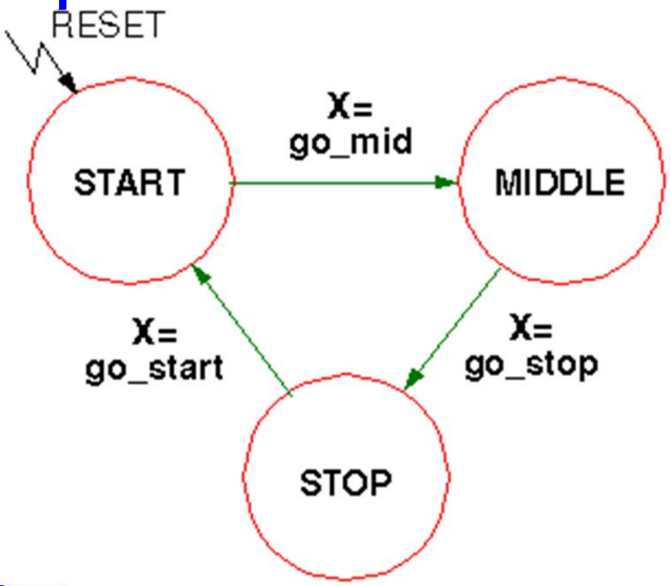
- **Designers may define their own types**
  - enhanced readability (commonly used to describe the states of a state machine)
  - limited legal values

Synthesis tools map enumerations onto a suitable bit pattern automatically.

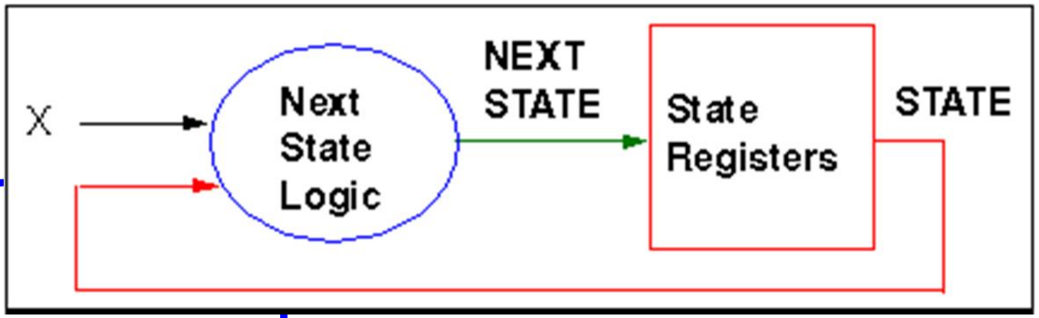
One-Process FSM

```
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 ;
```

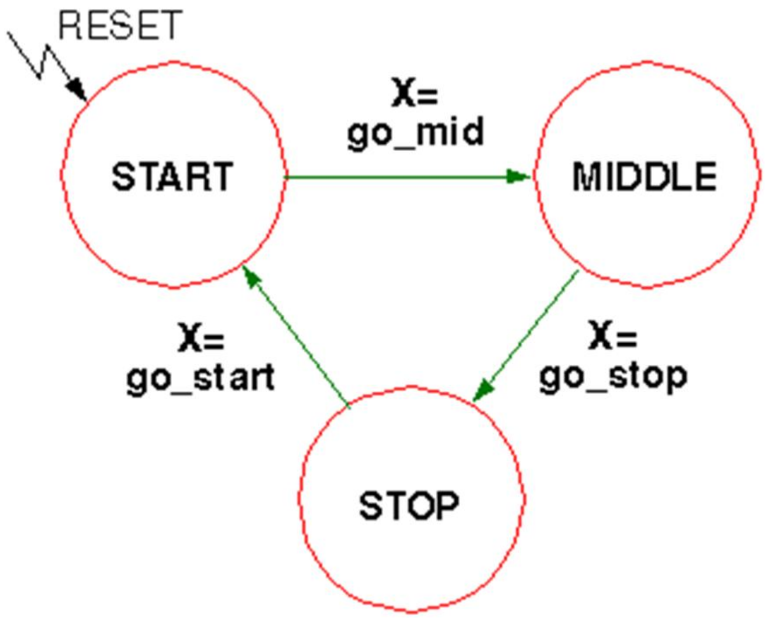




Two-Process FSM



```
FSM_LOGIC: process ( STATE , X)
begin
    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 ;

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 ;
```



How Many Processes?

- **Structure and Readability**
  - Asynchronous combinatory, synchronous storing elements
    => 2 processes
  - Graphical FSM (without output equations) resembles one state process
    => 1 process
- **Simulation**
  - Error detection easier with two state processes due to access to intermediate signals.
    => 2 processes
- **Synthesis**
  - 2 state processes can lead to smaller generic net list and therefore to better synthesis results
    (depends on synthesizer but in general, it is closer to hardware)
    => 2 processes

State Encoding

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

- **State encoding responsible for safety of FSM**

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

- **Default encoding: binary**

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

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

Note: unsafe FSM!

Encoding 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**

Not necessarily safe;
some synthesis tools will ignore "when others" choice

Extension of Type Declaration

```
type STATE_TYPE is (START, MIDDLE, STOP, DUMMY) ;
signal STATE : STATE_TYPE ;
…
   case STATE is
        when START    => …
        when MIDDLE  => …
        when STOP      => …

         when DUMMY     => …     -- or when others

   end case ;
```

- **Adding dummy values**
- **Only for binary encoding**
- **Advantages:**
  - Safe FSM after synthesis

$\{2^{\lceil \log 2(\#\text{ of states})\rceil} - n\}$ dummy states
(n=20 => 12 dummy states)

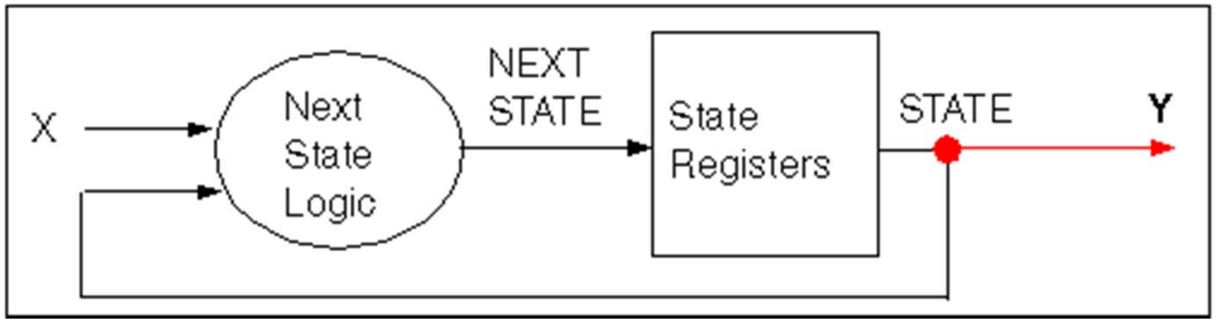
Changing to one hot coding => unnecessary hardware
(n=20 => 12 unnecessary Flip Flops)

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 ;
```

- **Defining constants**
- **Control of encoding**
- **Safe FSM**
- **Portable design**
- **Disadvantage:**
- More effort (especially when design changes)

FSM: Medvedev



Two Processes

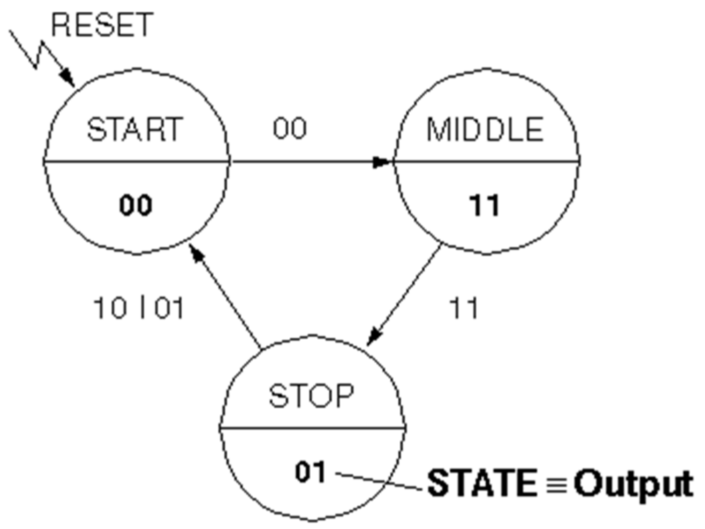
```
architecture RTL of MEDVEDEV is
   ...
begin
  REG: process (CLK, RESET)
  begin
     -- State Registers Inference
  end process REG ;
  CMB: process (X, STATE)
  begin
    -- Next State Logic
  end process CMB ;
  Y <= S ;
end RTL ;
```

- The output vector resembles the state vector: **Y = S**

One Process

```
architecture RTL of MEDVEDEV is
   ...
begin

   REG: process (CLK, RESET)
   begin
      --
State Registers Inference with Logic Block
   end process REG ;
   Y <= S ;
end RTL ;
```

Medvedev Example (2-Process)

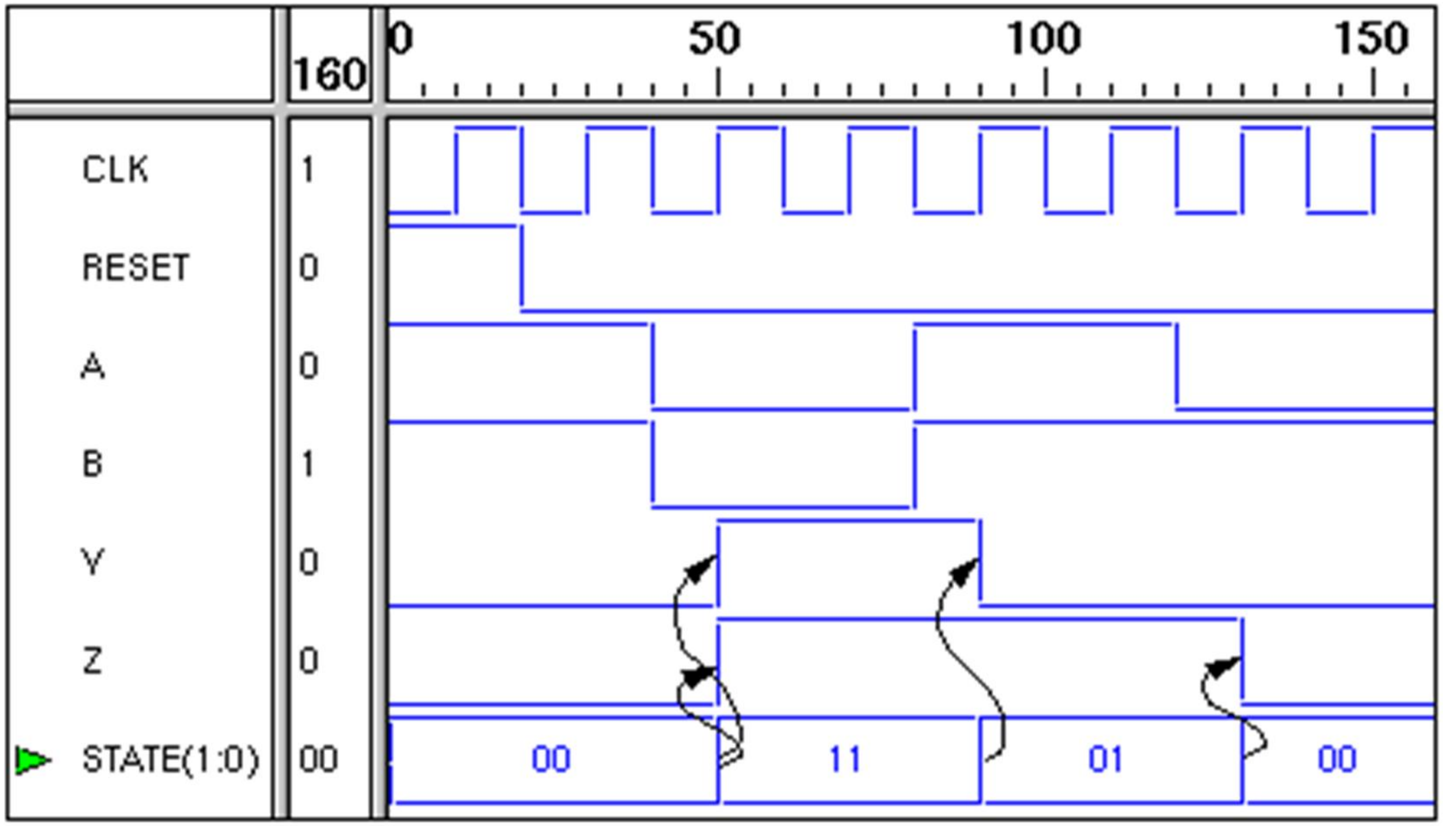


```
subtype STATE_TYPE is std_ulogic_vector(1 downto 0);
constant START   : STATE_TYPE := "00";
constant MIDDLE  : STATE_TYPE := "11";
constant STOP    : STATE_TYPE := "01";
```

```
architecture RTL of MEDVEDEV_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;
```

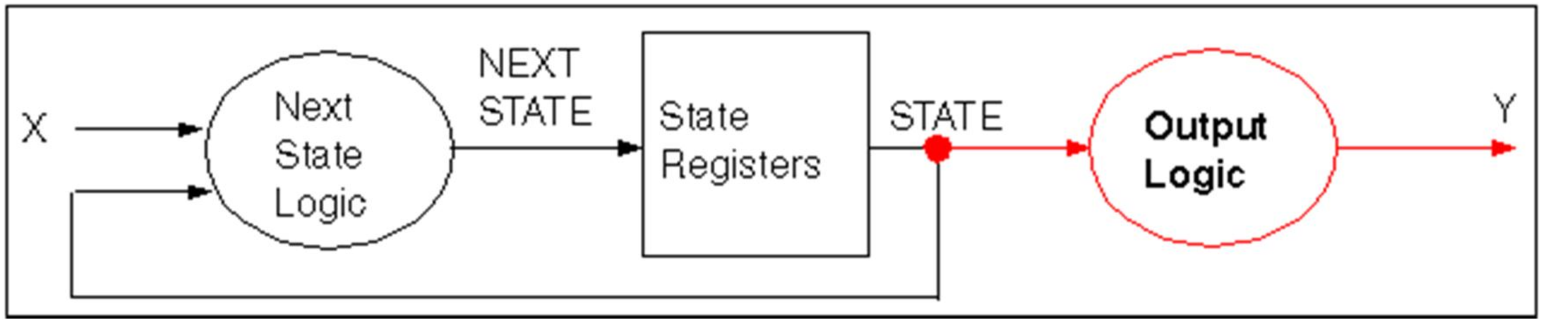
```
    CMB: process (A,B,STATE)  begin
       case STATE is
         when START  => if (A or B)='0' then
                          NEXTSTATE <= MIDDLE ;
                        end if ;
         when MIDDLE => if (A and B)='1' then
                          NEXTSTATE <= STOP ;
                     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

Medvedev Example Waveform



- **(Y,Z) = STATE => Medvedev machine**

FSM: Moore



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

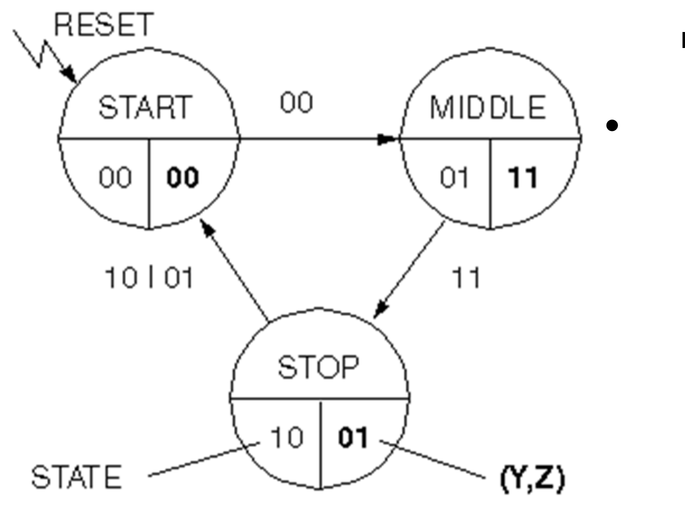
Two Processes

```
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 ;
```

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 ;
```

Moore Example



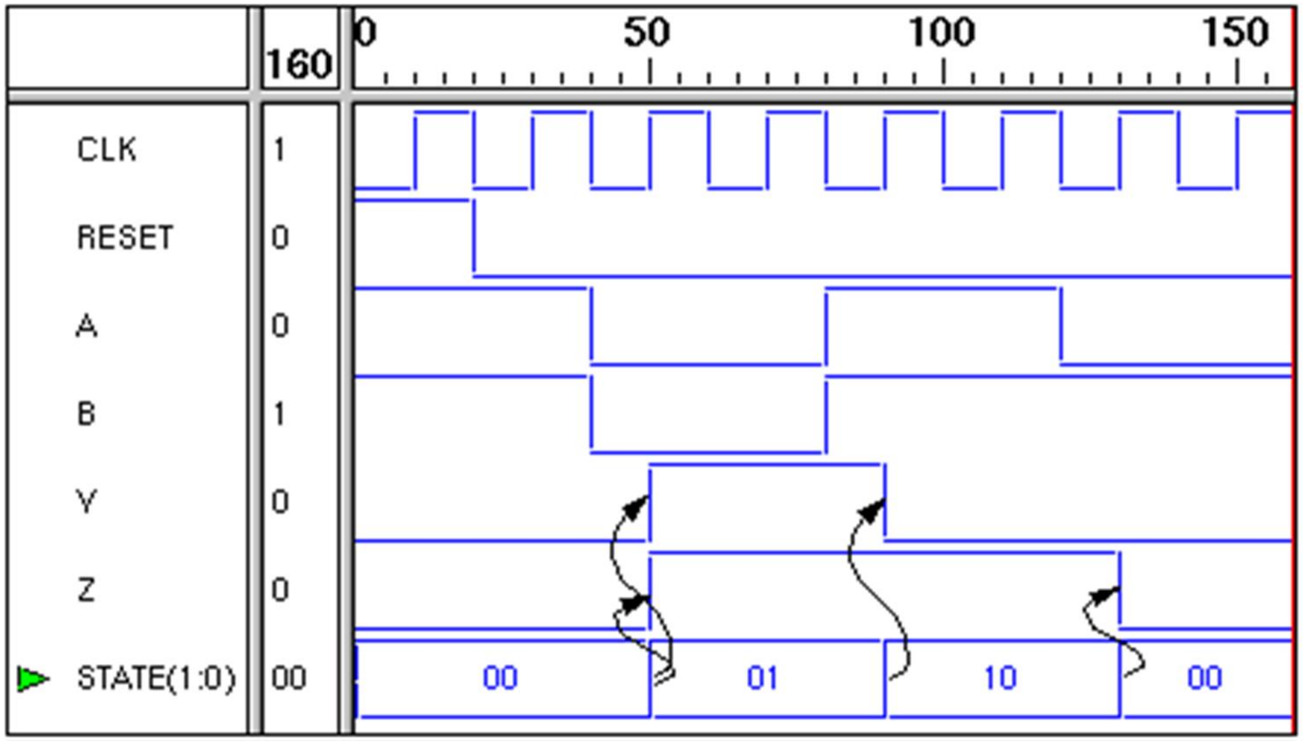
```
subtype STATE_TYPE is std_ulogic_vector(1 downto 0);
constant START    : STATE_TYPE := "00";
constant MIDDLE   : STATE_TYPE := "01";
constant STOP     : STATE_TYPE := "10";
```

- **Since outputs depend only on the current state, no signals other than STATE appears in the sensitivity list.**

```
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 ;
```

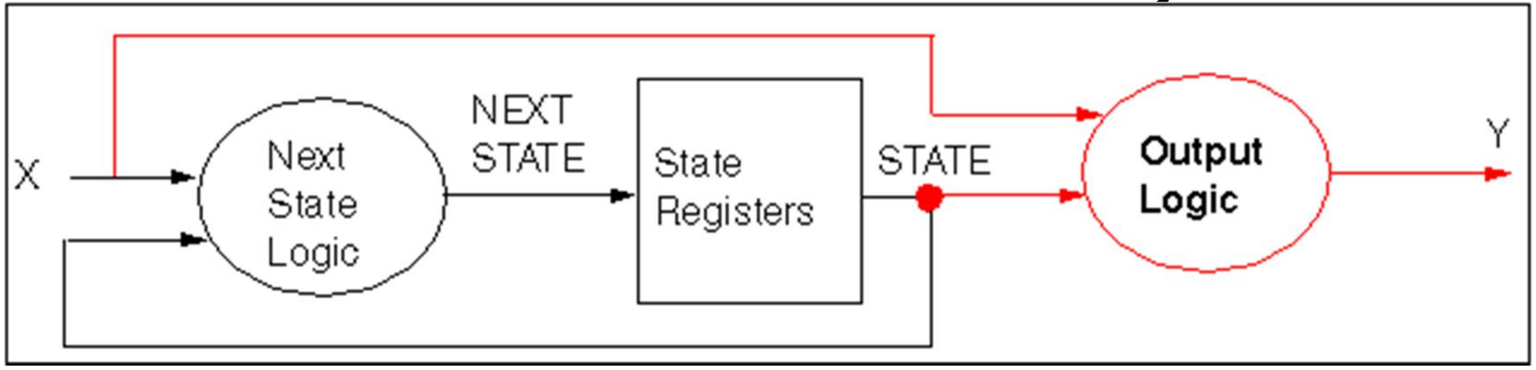
```
CMB: process (A,B,STATE) begin

   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 ;
```

Moore Example Waveform



- **(Y,Z) changes simultaneously with STATE à Moore machine**

FSM: Mealy



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

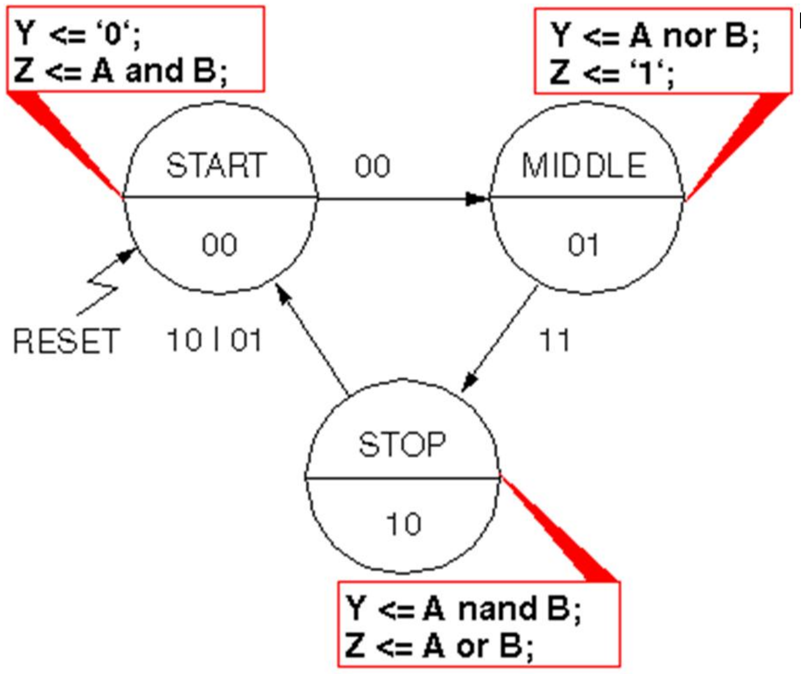
Two Processes

```
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 ;
```

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 ;
```

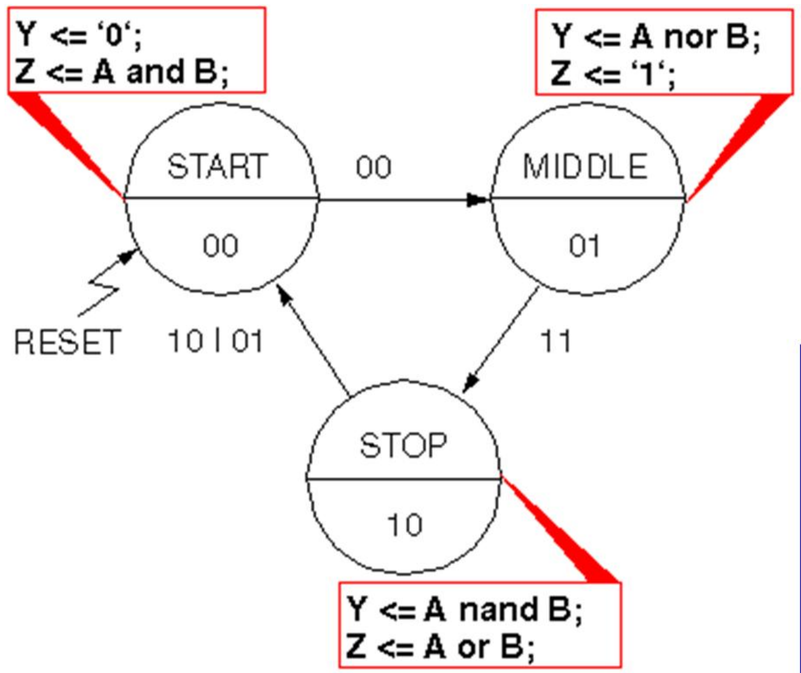
Mealy Example



```
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";
```

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

Mealy Example (Another Code)

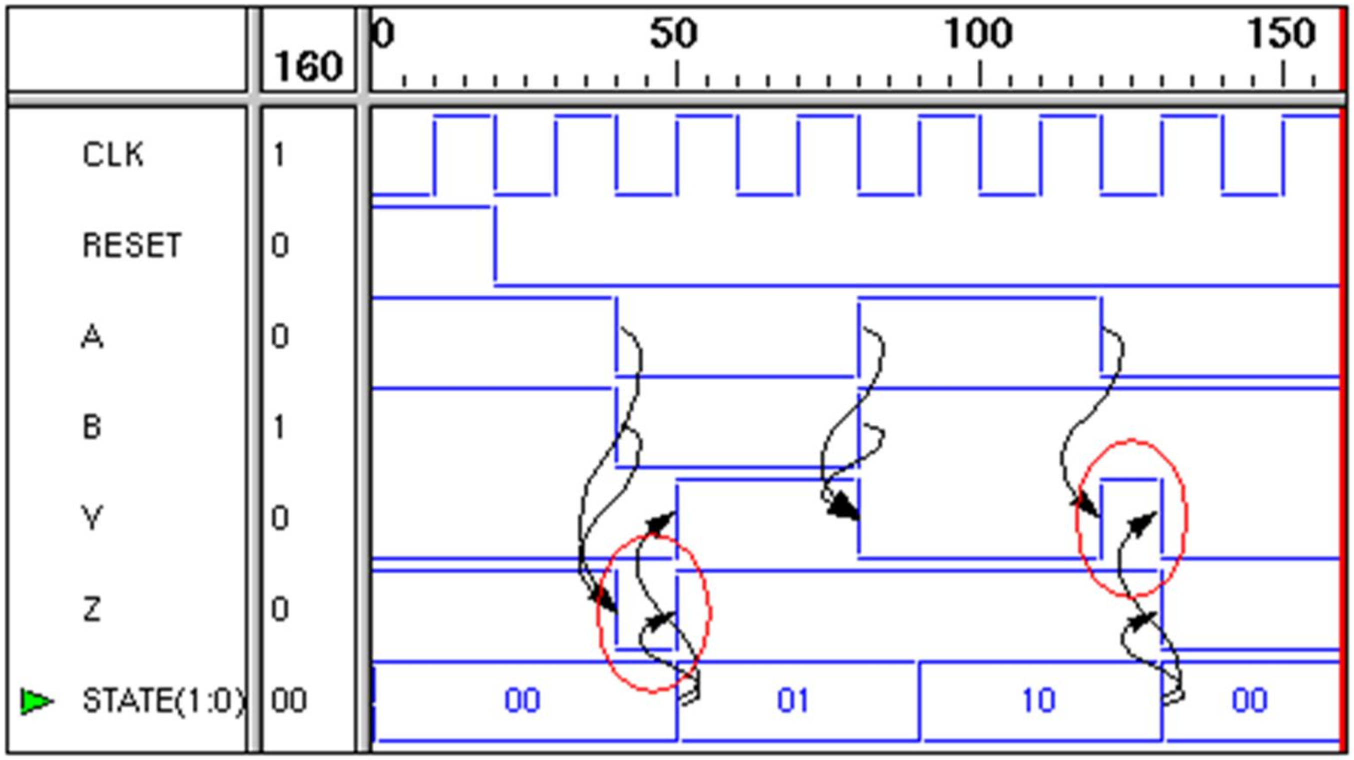


```
subtype STATE_TYPE is std_ulogic_vector(1 downto 0);
constant START      : STATE_TYPE := "00";
constant MIDDLE     : STATE_TYPE := "01";
constant STOP       : STATE_TYPE := "10";
```

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

```
REG: · · ·   -- clocked STATE process
CMB: · · ·   -- Like Medvedev and Moore Examples
-- Concurrent signal assignments for outputs
Y <= '1'
   when (STATE = MIDDLE and (A or B) = '0')
            or
            (STATE = STOP and (A and B) = '0')
   else '0';
Z <= '1'
   when (STATE = START and (A and B) = '1')
            or (STATE = MIDDLE) or
            (STATE = STOP and (A or B) = '1')
   else '0';
end RTL ;
```

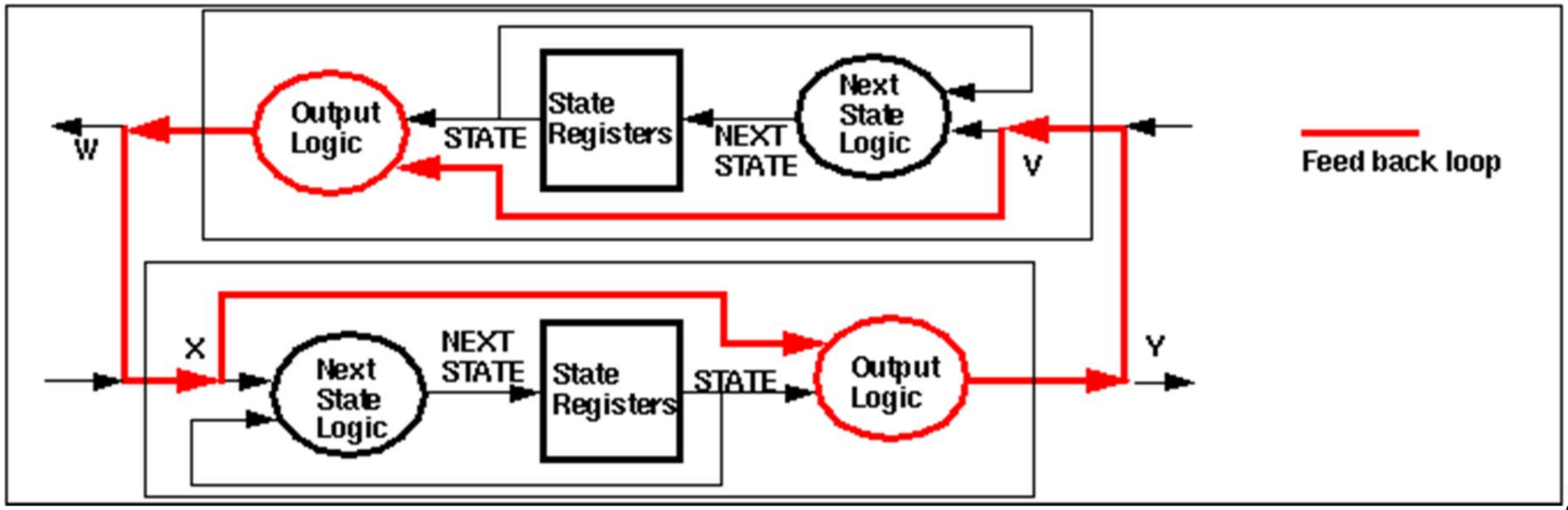
Mealy Example Waveform



- **(Y,Z) changes with input => Mealy machine**
- **Note the "spikes" of Y and Z in the waveform**

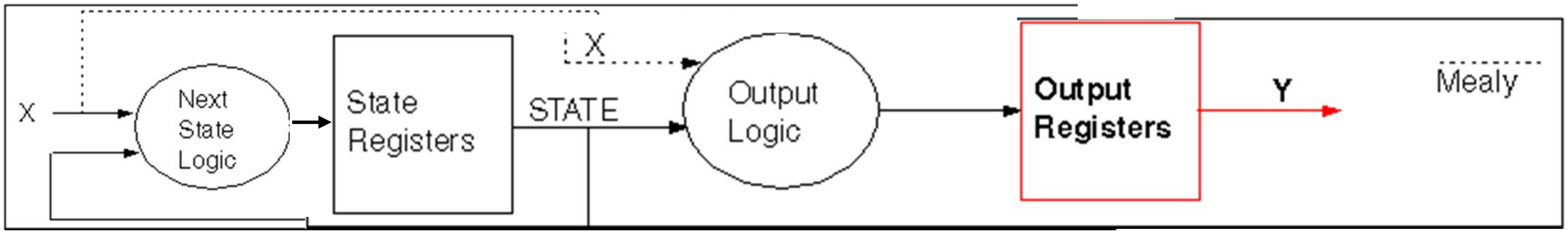
Modeling Aspects

- **Medvedev is too inflexible**
  - but less hardware (no combinational circuit for output)
  - More effort to calculate state vector.
- **Moore is preferred because of safe operation**
  - since o/p depends only on state vector.
  - **à** next output values are stable long before the next clock edge.
- **Mealy more flexible, but danger of**
  - Spikes
  - Unnecessary long paths (maximum clock period)
  - Combinational feed back loops

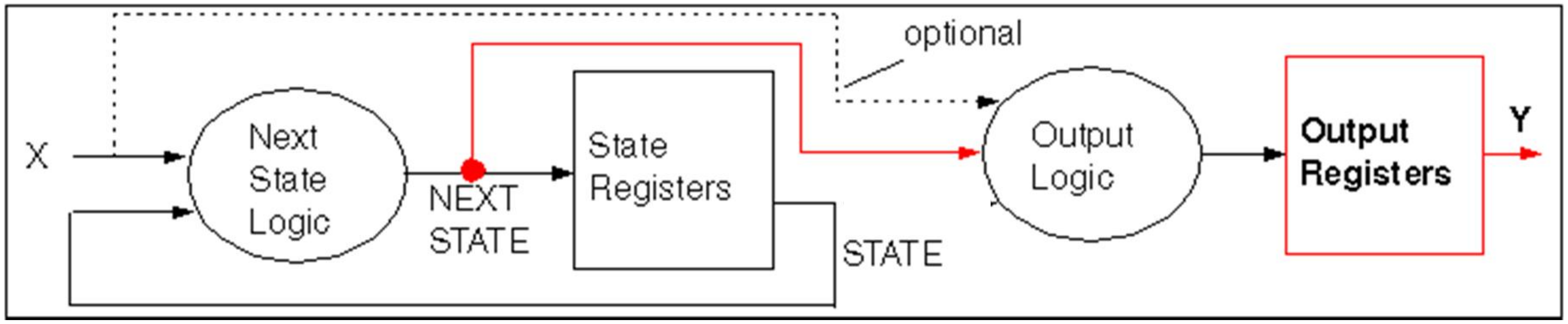


Registered Output

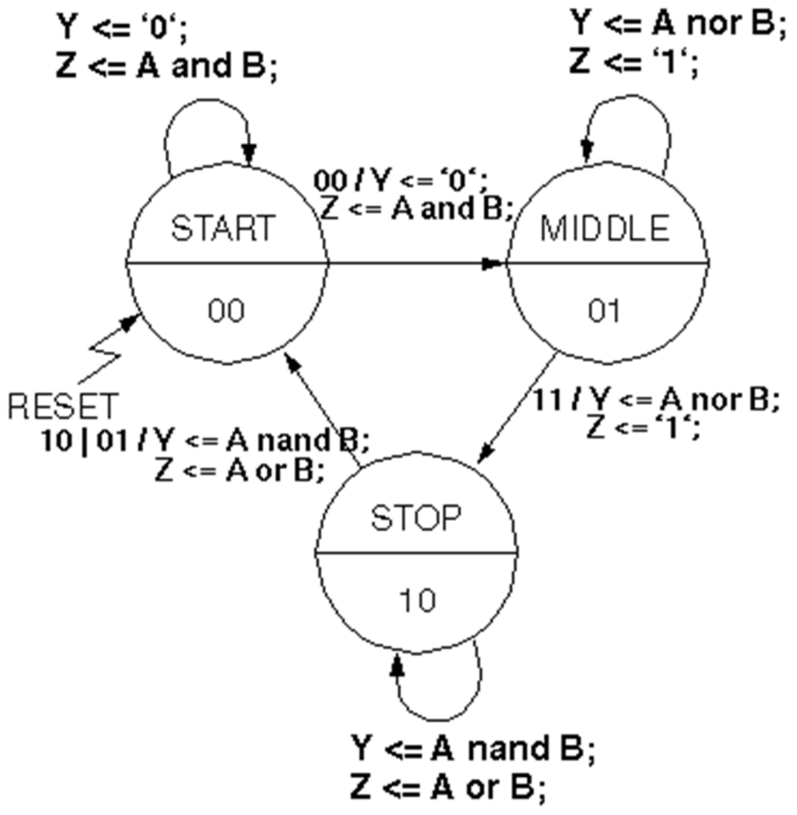
- **Avoiding long paths and combinational loops.**
- **With one additional clock period**



- **Without additional clock period**



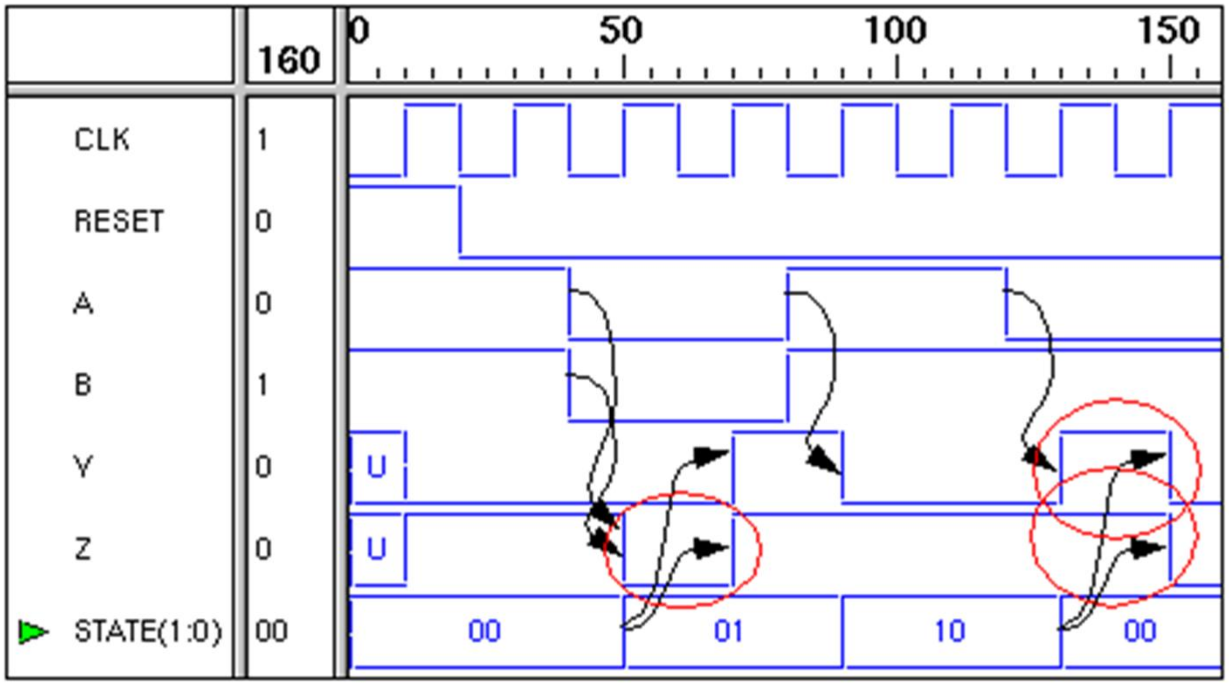
Registered Output Example (1)



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

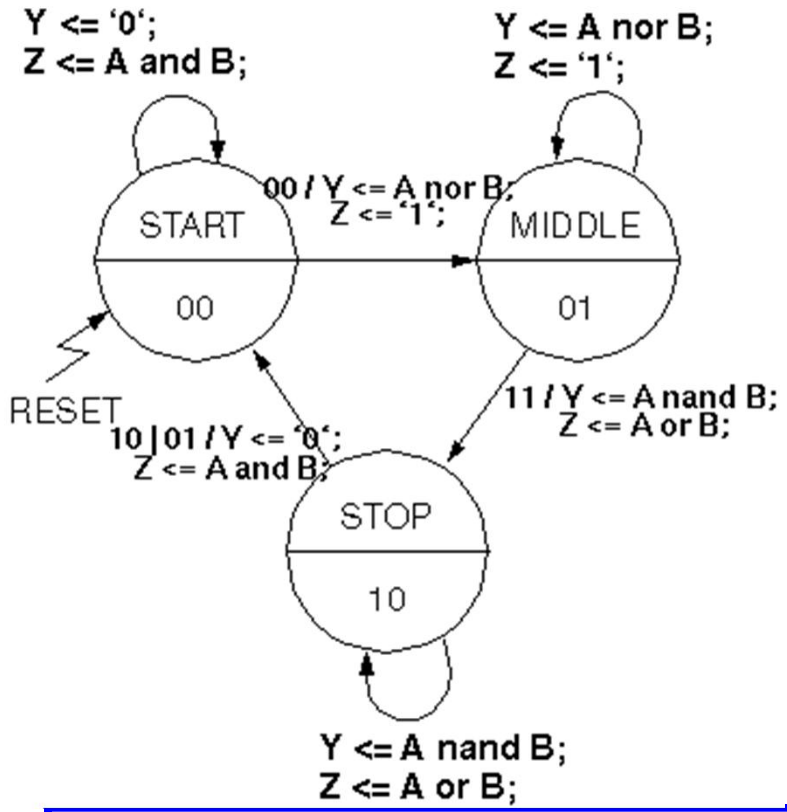
```
REG: · · ·   -- clocked STATE process

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 ;
```

Reg. Output Example Waveform



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

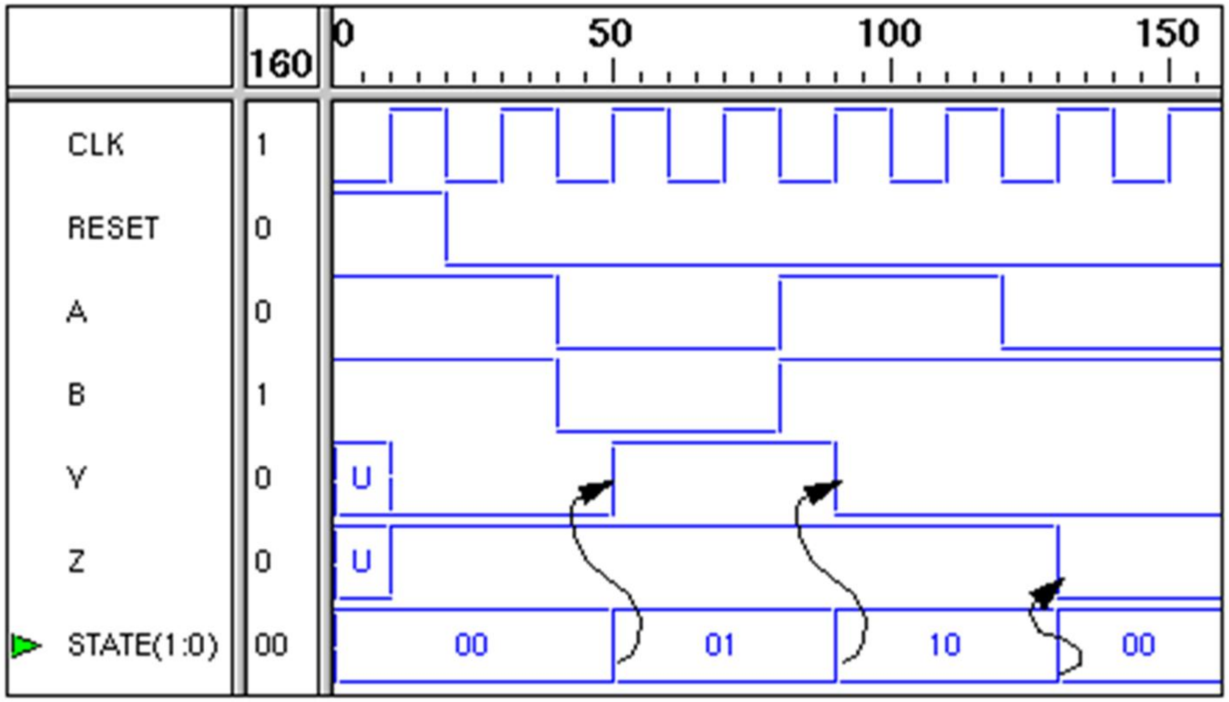
Registered Output Example (2)



```
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 ;
```

Reg. Output Example Waveform



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