Process “style 4”

- Processes with a sensitivity list including a clock signal, and eventually an asynchronous reset signal. An if statement constitutes the process, sensitive to the clock (and reset) events.

```
Process (ck_name [, reset_name])
begin
      [if (reset_name = value) then
            … reset behavior
      els]if (ck_name = value and ck_name'event) then
            … clocked behavior
      end if;
end process;
```


Specifying FSMs

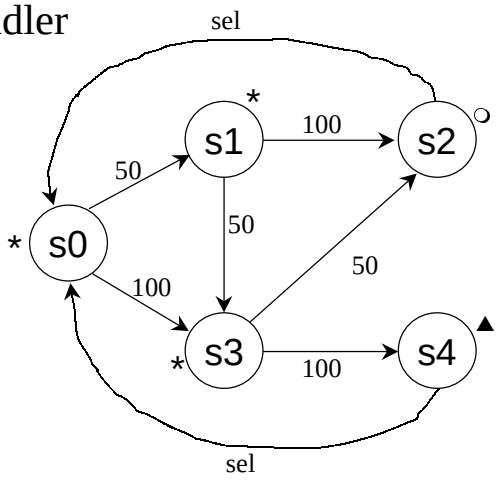
- Selection of:
  - Moore or Mealy style
  - Reset or not
  - Synchronous/asynchronous reset
  - Buffers position
  - *Data type definition for State elements*

FSMs in VHDL - Examples

- Two problems:
  - Coin handler for can disposal
    - Accepts coins of 50, 100, Selection button
    - When at 150 allows selection and no change
    - When at 200 allows selection and change
  - Bit sequence recognizer
    - Recognizes concatenated sequences
      - …0110…
      - …1110…
    - When recognized, output at 1.

FSMs in VHDL - Coin Handler - 1

- Coin handler



FSMs in VHDL - Coin Handler - 2

```
PACKAGE CH_pack IS
  type CHstate is (s0, s1, s2, s3, s4)
END CH_pack;

ENTITY CoinHandler is
  PORT(clk, res: in bit;
       Pin: in bit_vector(1 downto 0);
       --01: 50, 10: 100, 11:sel
       Enab, Change: out bit)
END CoinHandler;
```


FSMs in VHDL - Coin Handler - 3

```
ARCHITECTURE fsm OF CoinHandler IS
  signal pstate: CHstate := s0;
  signal nstate: CHstate := s0;
BEGIN
  StateEvol: process(clk, res)
  BEGIN
  IF (clk'EVENT and clk = '1') THEN
      IF (res = '0') THEN
         pstate <= s0;
      ELSE
         pstate <= nstate;
      END IF;
  END IF;
  END PROCESS StateEvol;
```

Synchronous reset signal

Reset active low

FSMs in VHDL - Coin Handler - 4

```
NSL: process(pstate, Pin)
BEGIN
  CASE pstate IS
    WHEN ‘s0’ => IF Pin = “01” THEN
                    nstate <= s1;
                 ELSIF Pin = “10” THEN
                    nstate <= s3;
                 ELSE
                    nstate <= pstate;
                 END IF;
    WHEN ‘s1’ => IF Pin = “01” THEN
                    nstate <= s2;
                 …
```

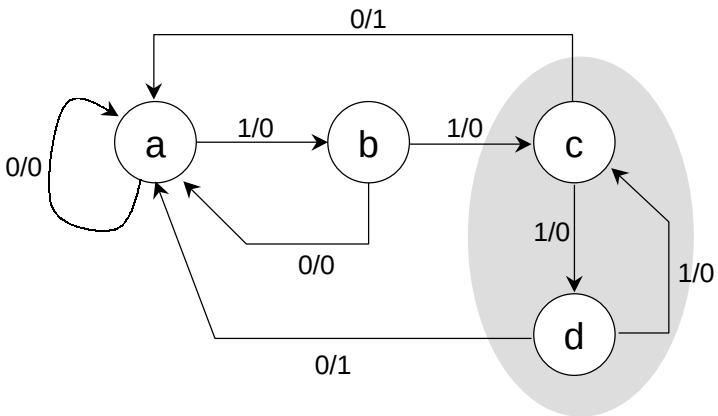

FSMs in VHDL - Coin Handler - 5

```
    WHEN ‘s4’ => IF Pin = “11” THEN
                    nstate <= s0;
                 ELSE
                    nstate <= pstate;
                 END IF;
    WHEN OTHERS => nstate <= s0;
  END CASE;
END PROCESS NSL;
```

FSMs in VHDL - Coin Handler - 6

```
OL: Process(pstate)
BEGIN
  CASE pstate IS
    WHEN s0|s1|s3 =>  Enab = '0';
                      Change = '0';
    WHEN s2 => Enab = '1';
               Change = '0';
    WHEN s4 => Enab = '1';
               Change = '1';
    WHEN OTHERS => Enab = '0';
                   Change = '0';
  END CASE;
  END Process OL;
END fsm;
```

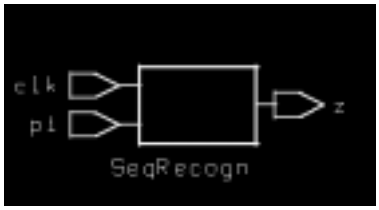

FSMs in VHDL - Recognizer - 1



FSMs in VHDL - Recognizer - 2

```
PACKAGE RecognPack IS
      type Rstate is (a, b, c, d);
END RecognPack;

ENTITY Recognizer is
      PORT(clk: in bit;
           PI: in bit;
           Z: out bit)
END Recognizer ;
```



FSMs in VHDL - Recognizer - 3

```
ARCHITECTURE ThreeProc OF Recognizer IS
      signal pstate, nstate: Rstate := a;
BEGIN
      NextState: PROCESS
      BEGIN
        WAIT UNTIL clk = '1' AND clk'EVENT
        CASE pstate IS
            WHEN a => IF PI = '1' THEN
                            nstate <= b;
                      ELSE
                            nstate <= a;
                      END IF;
            WHEN b => IF PI = '1' THEN
                            nstate <= c;
                      ELSE
                            nstate <= a;
                      END IF;
```

FSMs in VHDL - Recognizer - 4

```
    WHEN c => IF PI = '1' THEN
                      nstate <= d;
              ELSE
                      nstate <= a;
              END IF;
    WHEN d => IF PI = '1' THEN
                      nstate <= c;
              ELSE
                      nstate <= a;
              END IF;
     WHEN OTHERS => nstate <= a;
   END CASE;
END PROCESS NextState;
```

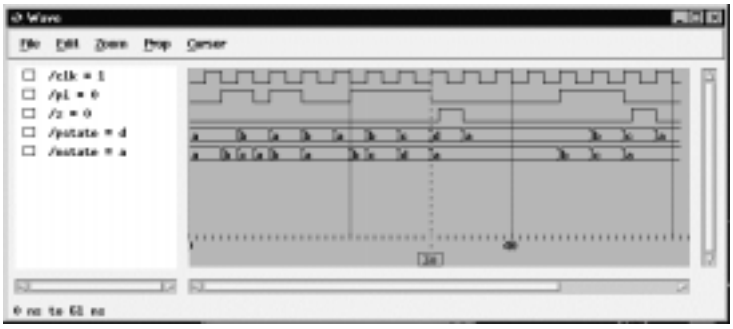

FSMs in VHDL - Recognizer - 5

```
Output: PROCESS(pstate, PI)
BEGIN
  CASE pstate IS
    WHEN a => Z <= '0';
    WHEN b => Z <= '0';
    WHEN c => IF PI = '1' THEN
                      Z <= '0';
              ELSE
                      Z <= '1';
              END IF;
    WHEN d => IF PI = '1' THEN
                      Z <= '0';
              ELSE
                      Z <= '1';
              END IF;
    WHEN OTHERS => Z <= '0';
  END CASE;
END PROCESS;
```

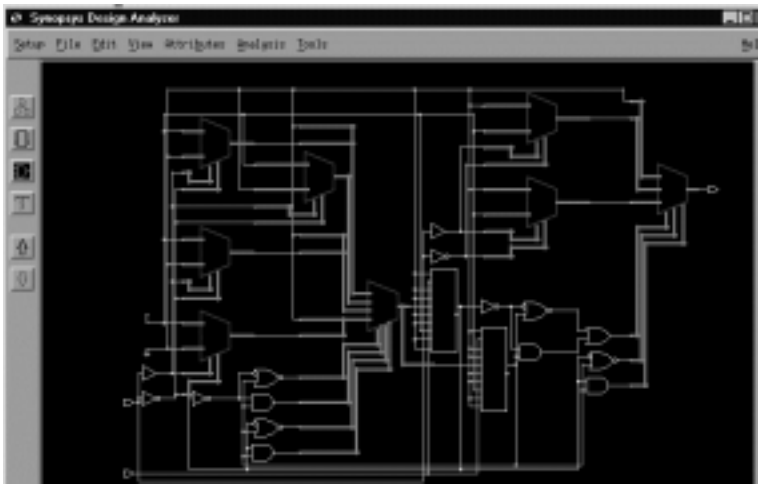
FSMs in VHDL - Recognizer - 6

```
  StateEvol: Process
  begin
        WAIT UNTIL clk'EVENT and clk = '1';
        pstate <= nstate;
  end process;
end ThreeProc;
```

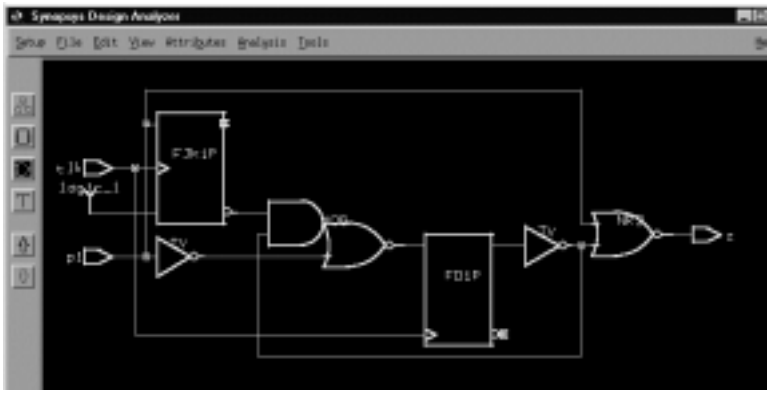

FSMs in VHDL - Recognizer - 7



FSMs in VHDL - Recognizer - 8



FSMs in VHDL - Recognizer - 9



Sequential elements - summary

- Latch inference
  - No specified assignment for a conditional branch
  - It is not possible to model latches using variables
- Clock inference

```
CLK = '1' AND clk'EVENT
```

  - type: Bit, Boolean, Std_ulogic, Std_logic
  - No ELSE clause in the IF statement checking the clock edge

Signals: wires or memory?

- Latches are inferred when signals are not assigned in all conditional branch.
- Latches are inferred each time an assignment is made before a wait until statement
- Memory (not latches) is inferred on variables anytime it is necessary to store data for subsequent simulation steps.

Synthesis guidelines

- Data Flow statements translate into combinational logic
- Sequential logic is specified in a PROCESS statement that includes a Clock signal.
- Explicit State Machines contain one WAIT UNTIL CLOCK'EDGE in a process, with a CASE statement describing the FSM.
- Implicit State Machines contain multiple WAIT UNTIL CLOCK'EDGE in a process without a sensitivity list.