Explicit and Implicit Models

- Explicit - declares a state register that stores the FSM state
- Implicit - describes state implicitly by using multiple event controls
- Mealy versus Moore types

Explicit FSM Building Blocks

- Block
  - Continuous
    - State register
    - Next state logic: X
    - Output logic: X
    - Output register: X
  - Procedural
    - X

- Combinations
  - Next state & output logic
  - Next state logic & state register
  - State register & output register

- Connection Type
  - Moore
  - Mealy
Types of Explicit Models

- State register - Combinational next state and output logic
- State register - Combinational next state logic - Combinational output logic
- State register - Combinational next state logic - Registered output logic

State register - Combinational next state and output logic

State register - Combinational next state logic - Combinational output logic
State register - Combinational next state logic - Registered output logic

FSM Example: State Register- Next State Logic – Output Logic - 1

```
module fsm_sr_ns_o (clk, reset, a, b, Y, Z);
    input clk, reset, a, b;
    output Y, Z;
    reg[1:0] state, next_state;
    reg Y, Z;
    parameter S0 = 2’b00, S1 = 2’b01, S2 = 2’b10;
    // state register
    always@(posedge clk or posedge reset)
        if (reset)
            state <= S0;
        else
            state <= next_state; // continued on next slide
    // next state logic
    always@(state or a or b)
        case (state)
            S0: if (a)
                next_state = S1;
            else
                next_state = S0;
            S1: if (b)
                next_state = S2;
            else
                next_state = S1;
            S2: next_state = S0;
            default: next_state = 2’bx;
        endcase
    // output logic
    always@(state or a or b)
        case (state)
            S0: if (a) Z = 1;
            S1: begin
                Y = 1;
                if (b) Z = 1;
            end
            default: begin
                Y = 1’bx; Z = 1’bx;
            end
        endcase
endmodule
```
FSM Example: State Register - Next State & Output Logic – 1

```verilog
module fsm_sr_nso (clk, reset, a, b, Y, Z);
input clk, reset, a, b;
output Y, Z;
reg[1:0] state, next_state;
reg Y, Z;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;
//state register
always@((posedge clk or posedge reset))
if (reset)
    state <= S0;
else
    state <= next_state; //continued on next slide
endmodule
```

FSM Example: State Register - Next State & Output Logic - 2

```verilog
//next state & output logic
if (b)
    begin
        next_state = S2;
        Z = 1;
    end
else
    begin
        next_state = S1;
        S2: next_state = S0;
        default: next_state = 2'bx;
    endcase
endmodule
```

Verilog - State Register - Next State and Output Logic - Output Register

- If delay of the output for one clock cycle acceptable
  - Output register can simply be added
  - Same output logic can feed output flip-flop inputs as fed combinational outputs
- If outputs are to obey specifications on a clock cycle specific basis, i.e., are not delayed
  - Then the output flip-flop D-input functions must be defined one cycle earlier than the normal combinational output.
  - Can be done for Moore-type FSM outputs
  - Mealy-type outputs can be handled only by modification of FSM.

FSM Example - State Register - Next State Logic & Output Logic - Output Register

- Original Combinational: \( Y = S1 \)

- New Sequential: \( Y(t+1) = S0 \cdot a + S1 \cdot \neg b \)
- Impossible to do Z!
FSM Example - State Register - Next State Logic and Output logic - Output Register - General Transformation Method

module fsm_sr_ns_o_or (clk, reset, a, b, Y, Z);
input clk, reset, a, b;
output Y, Z;
reg[1:0] state, next_state;
reg Y, Z;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;
//state register
always @(posedge clk or posedge reset)
if (reset)
begin
state <= S0; Y <= 0;
end
else begin
state <= next_state; Y <= next_Y;
//continued on next slide
end
//next state logic
always @(state or a or b)
case (state)
S0: if (a)
next_state <= S1;
else
next_state <= S0;
S1: if (b)
next_state = S2;
else
next_state = S1;
S2: next_state = S0;
default: next_state = 2'bx;
endcase
//output logic
always @(state or a or b) begin
Next_Y = 0; Z = 0; /* fixes default output value; avoids latch*/
case (state)
S0: if (a) begin
Next_Y = 1;
Z = 1; end /* Z cannot be a registered output */
S1: if (!b) begin
Next_Y = 1;
else
Z = 1; end
endcase
endmodule

Combined Datapath & Control Modeling

- For higher level modeling, easier to visualize function
FSM Example - Combined Datapath & Control Modeling

- Assume for the running FSM Example that:
  - The example circuit interacts with a datapath consisting of a 4-bit up counter \(\text{COUNT}[3:0]\)
  - \(a = 1\) starts the circuit activity
  - \(Y = 1\) enables the counter to count up.
  - \(b = 1\) indicates that the counter has reached value 1111.
  - \(Z = 1\) toggles a flip-flop that is also changed asynchronously to 0 by signal reset.

---

FSM Example: Combined - 1

```vhdl
module cntrl_dtpth (clk, reset, a, b, COUNT, R);
  input clk, reset, a, b;
  output[3:0] COUNT;
  output R;
  reg[1:0] state, next_state;
  reg R;
  reg [3:0] COUNT;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;
  // state register
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      state <= S0; COUNT <= 0; R <= 0;
    end
    else begin
      case (state or a or b)
        S0: if (a) begin
          state <= S1;
          R <= ~ R;
        end
        S1: begin
          COUNT <= COUNT + 1;
          state <= S1;
        end
        S2: state <= S0;
        default: state <= 2'bx;
      endcase
      R <= ~ R;
    end
  endmodule
```

---

FSM Example: Combined - 2

```vhdl
case (state or a or b)
  S0: if (a) begin
    state <= S1;
    R <= ~ R;
  end
  else
    state <= S0;
  S1: begin
    COUNT <= COUNT + 1;
    state <= S1;
  end
  S2: state <= S0;
  default: state <= 2'bx;
endcase
endmodule
```

---

Implicit Model

- More abstract representation
- Restricted to structures in which a given state can be entered from only one other state!
- Yields simpler code
- Description of reset behavior more complex
Implicit Model (continued)

- Implicit model typically advances state by use of series of @posedge clock conditions
- Example: Mod 3 counter
  always begin
  @posedge clock
  count = 0;
  @posedge clock
  count = count + 1;
  @posedge clock
  count = count + 1;
  end

FSM Example: Implicit Model - 1

```plaintext
\This code is not synthsizable with FPGA Express (see *).
module fsm_implicit (clk, reset, a, b, Y);
input clk, reset, a, b;
output Y; //Z as Mealy output is not implementable
reg Y;

always @(posedge reset)
if (reset)
  begin
    Y <= 0;
    disable main; /*
  end

always @(posedge clk)   //S0*
if (a)
  Y <= 1;
else while (!a)
  @(posedge clk)
    if (b)  
      Y <= 0;
  @(posedge clk)
    Y <= 0;
  @(posedge clk)   //S2*
    ;
end
endmodule
```

FSM Example: Implicit Model - 2 *

```plaintext
always begin : main
  @(posedge clk)   //S0*
    if (a)
      Y <= 1;
  else while (!a)
    @(posedge clk)
      if (a)  
        ;
    @(posedge clk)
      Y <= 1;
  @(posedge clk)   //S1*
if (b)
  Y <= 0;
else while (!b)
  @(posedge clk)
    if (b)  
      ;
  @(posedge clk)
    Y <= 0;
  @(posedge clk)   //S2*
    ;
end
endmodule
```