

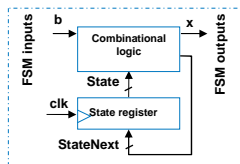
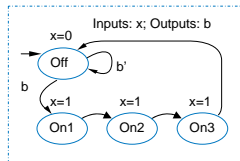
Lecture 12B

Sequential Logic Design Considerations

Finite-State Machines (FSMs)—Sequential Behavior

Modules with Multiple Procedures and Shared Variables

- Previously considered FSM implementation architecture and Verilog modeling



```

`timescale 1 ns/1 ns
module LaserTimer(B, X, Clk, Rst);
  input B;
  output reg X;
  input Clk, Rst;

  parameter S_Off = 0, S_On1 = 1,
            S_On2 = 2, S_On3 = 3;

  reg [1:0] State, StateNext;

  // CombLogic
  always @(State, B) begin
    case (State)
      S_Off: begin
        X <= 0;
        if (B == 0)
          StateNext <= S_Off;
        else
          StateNext <= S_On1;
        end
    end
  end
endmodule
  
```

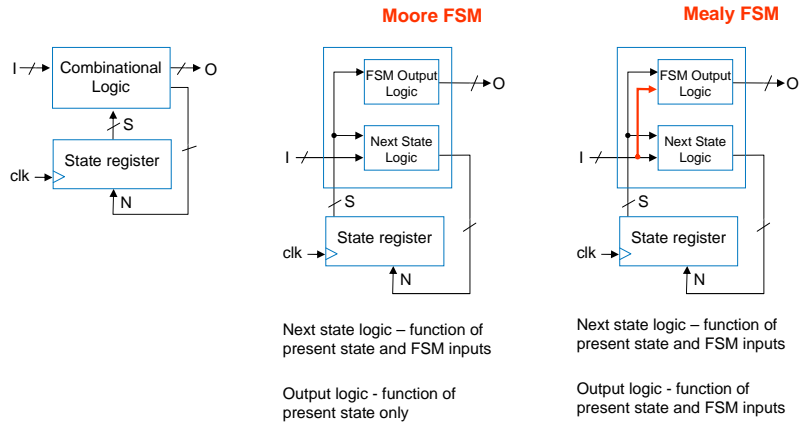
```

S_On1: begin
  X <= 1;
  StateNext <= S_On2;
end
S_On2: begin
  X <= 1;
  StateNext <= S_On3;
end
S_On3: begin
  X <= 1;
  StateNext <= S_Off;
end
endcase

// StateReg
always @(posedge Clk) begin
  if (Rst == 1) State <= S_Off;
  else State <= StateNext;
end
endmodule
  
```

Moore vs. Mealy FSM - Architecture

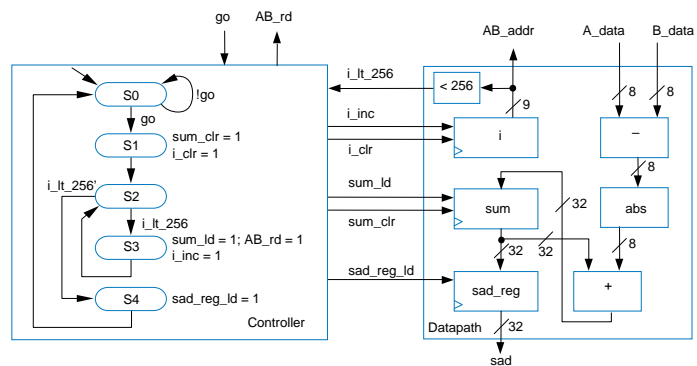
- Detailed view of architecture shows outputs derived from combinational logic



FSM+D

Sum of Absolute Differences Circuit Implementation

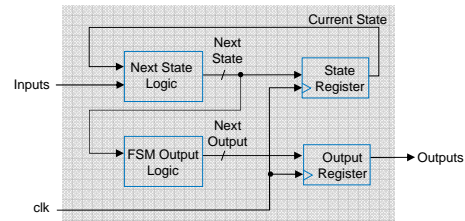
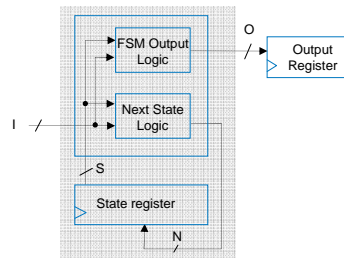
- Conversion of HLFSM to FSM+D says to instantiate a register for each data output to avoid glitching



Registered Output FSM

Architecture Overview

- FSM bit outputs can similarly be registered
 - Just add another register at the output of the FSM? Yes, but you will get a 1 cycle delay
 - Instead, add procedure to determine next output value
 - Outputs transition in parallel with states
 - Avoid glitching problems, race conditions, invalid logic operations, promotes resource sharing



R. Johnson, "Registered-output FSMs synchronize outputs to state transitions", EDN Design Feature, June 1998

Registered Output FSM

Verilog Implementation

- How to implemented following FSM?

```

module FSM(clk, mrb, start, cnt16, initb, acc_en);
input clk, mrb, start, cnt16;
output reg initb, acc_en;

parameter IDLE = 0, INIT = 1, LOOPING = 2,
          DONE = 3;

reg [1:0] State, StateNext;
reg INITb_Next, ACC_EN_Next;

// next-state combinational logic
always@(State, start, cnt16) begin
end

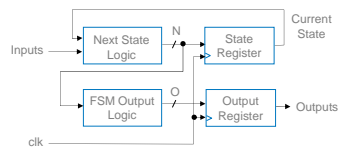
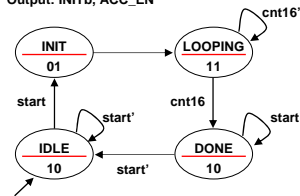
// next-output combinational logic
always@(StateNext) begin

end

// state register
always@(posedge clk, negedge mrb) begin
end

// output register
always@(posedge clk, negedge mrb) begin
end
endmodule
    
```

Inputs: start, cnt16
Output: INITb, ACC_EN



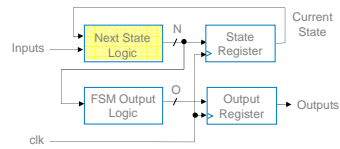
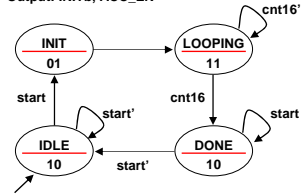
Registered Output FSM

Verilog Implementation

```
// next-state combinational logic
// -----
always@(State, start, cnt16) begin
  case (State)
    IDLE : begin
      if (start == 1)
        StateNext <= INIT;
      else
        StateNext <= IDLE;
      end
    INIT : begin
      StateNext <= LOOPING;
      end
    LOOPING : begin
      if (cnt16 == 1)
        StateNext <= DONE;
      else
        StateNext <= LOOPING;
      end
    DONE : begin
      if (start == 0)
        StateNext <= IDLE;
      else
        StateNext <= DONE;
      end
  endcase
end
```

```
DONE : begin
  if (start == 0)
    StateNext <= IDLE;
  else
    StateNext <= DONE;
  end
endcase
end
```

Inputs: start, cnt16
Output: INITb, ACC_EN



ECE 474a/575a
Susan Lysecky

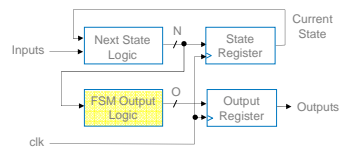
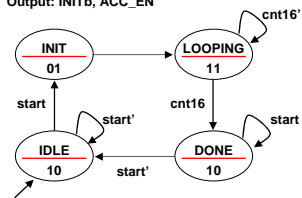
7 of 19

Registered Output FSM

Verilog Implementation

```
// next-output combinational logic
// -----
always@(StateNext) begin
  case (State)
    IDLE : begin
      INITb_Next <= 1;
      ACC_EN_Next <= 0;
      end
    INIT : begin
      INITb_Next <= 0;
      ACC_EN_Next <= 1;
      end
    LOOPING : begin
      INITb_Next <= 1;
      ACC_EN_Next <= 1;
      end
    DONE : begin
      INITb_Next <= 1;
      ACC_EN_Next <= 0;
      end
  endcase
end
```

Inputs: start, cnt16
Output: INITb, ACC_EN



ECE 474a/575a
Susan Lysecky

8 of 19

Registered Output FSM

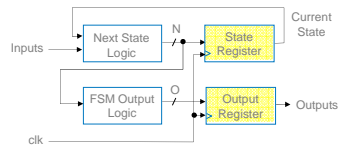
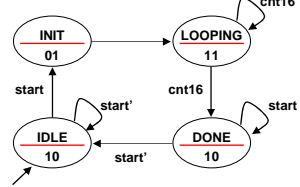
Verilog Implementation

```

// state register
// -----
always@(posedge clk, negedge mrb) begin
    if(mrb == 0)
        State <= IDLE;
    else
        State <= StateNext;
end

// output register
// -----
always@(posedge clk, negedge mrb) begin
    if(mrb == 0) begin
        initb <= 0;
        acc_en <= 0;
    end
    else begin
        initb <= INITb_Next;
        acc_en <= ACC_EN_Next;
    end
end
    
```

Inputs: start, cnt16
Output: INITb, ACC_EN

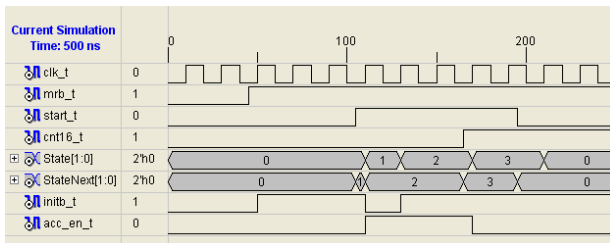
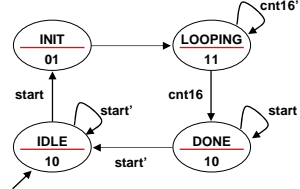


Registered Output FSM

Timing Diagram

- Simulation Result
 - Desired behavior achieved
 - Output changes with state

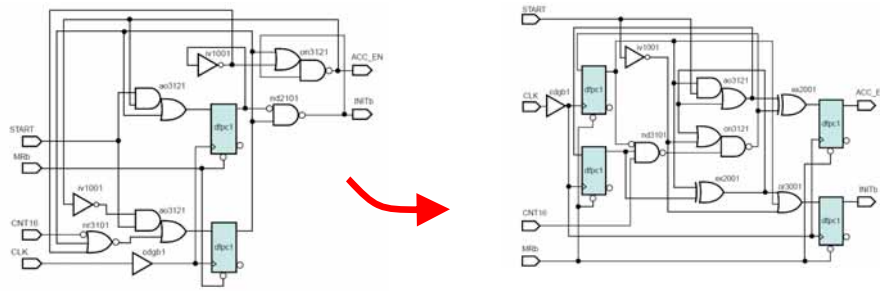
Inputs: start, cnt16
Output: INITb, ACC_EN



Registered Output FSM

Synthesis Implementation

- Synthesis Result
 - All output derived from flip-flops
 - Output value change associated with state
 - Outputs stable for one flip-flop propagation delay after clock, no glitching



R. Johnson, "Registered-output FSMs synchronize outputs to state transitions", EDN Design Feature, June 1998

ECE 474a/575a
Susan Lysecky

11 of 19

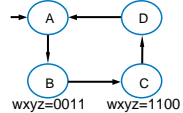
Registered Output FSM

Output Encoded FSM

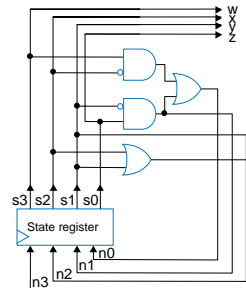
- Revisit Output Encoding
 - Choosing state encodings to force the state register to drive the output

Outputs: w, x, y, z

wxyz=0001 wxyz=1000



	Inputs				Outputs			
	s3	s2	s1	s0	n3	n2	n1	n0
A	0	0	0	1	0	0	1	1
B	0	0	1	1	1	1	0	0
C	1	1	0	0	1	0	0	0
D	1	0	0	0	0	0	0	1



Digital Design, Copyright © 2006
Frank Vahid

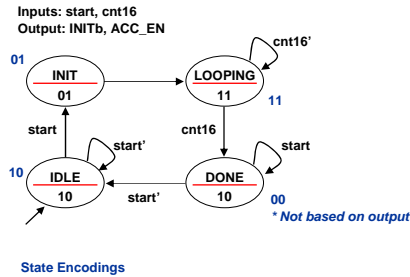
ECE 474a/575a
Susan Lysecky

12 of 19

Registered Output FSM

Output Encoded FSM

- If multiple states have same outputs, we arbitrary choose a free state encoding



	Inputs				Outputs			
	s1	s0	start	cnt16	n1	n0	INITb	ACC_EN
DONE	0	0	0	-	1	0	1	0
	0	0	1	-	0	0	1	0
INIT	0	1	-	-	1	1	0	1
IDLE	1	0	0	-	1	0	1	0
	1	0	1	-	0	1	1	0
LOOPING	1	1	-	0	1	1	1	1
	1	1	-	1	0	0	1	1

Registered Output FSM

Output Encoded FSM

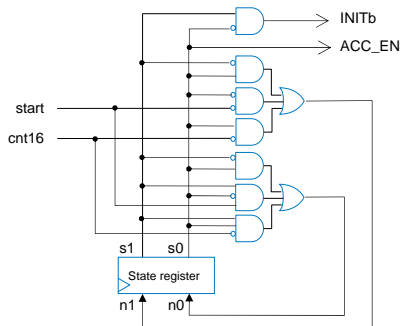
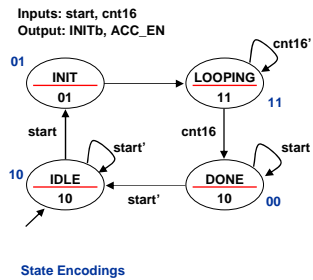
- Outputs may not be registered
 - ACC_EN registered, INITb not registered

$$n1 = s1's0 + s0'start' + s0cnt16'$$

$$n0 = s1's0 + s1s0'start + s1s0cnt16'$$

$$INITb = s1 + s0'$$

$$ACC_EN = s0$$

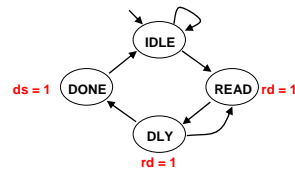


Registered Output FSM

Algorithm

- Introduce a structured method for encoding the outputs as part of the state encoding to ensure FSM outputs are registered

- Count the number of outputs (x) and the number of states (y). Start by making a table with y+1 rows and x+1 columns
- Starting at 2nd row in the left-hand column, make a list of FSM states
- Starting at the first row, second column and working to the right, list each FSM output as a separate column
- Place a "1" in each output column where an output is high for the listed state, 0 otherwise
- Search for output patterns that are the same for more than one state. If no duplicate patterns, you are done.
- Circle duplicate output patterns



State	ds	rd
IDLE	0	0
READ	0	1
DLY	0	1
DONE	1	0

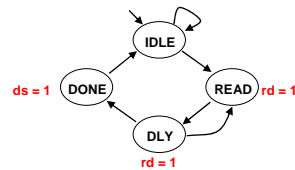
C. Cummings. "Coding and Scripting Techniques For FSM Designs with Synthesis-Optimized, Glitch-Free Outputs." SNUUG, 2000.

Registered Output FSM

Algorithm

- Introduce a structured method for encoding the outputs as part of the state encoding to ensure FSM outputs are registered

- Count the number of outputs (x) and the number of states (y). Start by making a table with y+1 rows and x+1 columns
- Starting at 2nd row in the left-hand column, make a list of FSM states
- Starting at the first row, second column and working to the right, list each FSM output as a separate column
- Place a "1" in each output column where an output is high for the listed state, 0 otherwise
- Search for output patterns that are the same for more than one state. If no duplicate patterns, you are done.
- Circle duplicate output patterns
- If two patterns are the same, add one additional state bit to create unique encodings (3-4 patterns, add 2 state bits, ...)
- Add blank column between state names and output columns. Fill added columns with zeros in non-circled rows. Add binary encodings for redundant encodings.



State	x0	ds	rd
IDLE	0	0	0
READ	0	0	1
DLY	1	0	1
DONE	0	1	0

C. Cummings. "Coding and Scripting Techniques For FSM Designs with Synthesis-Optimized, Glitch-Free Outputs." SNUUG, 2000.

Registered Output FSM

Output Encoded FSM Implementation

```

module FSM(ds, rd, go, ws, clk, rst);
input go, ws, clk, rst;
output ds, rd;
parameter [2:0] IDLE = 3'b000,
                READ = 3'b001,
                DLY = 3'b101,
                DONE = 3'b010;

```

```

reg [2:0] State, StateNext;

// State Register
always @(posedge clk, posedge rst) begin
if( rst ) State <= IDLE;
else State <= StateNext;
end

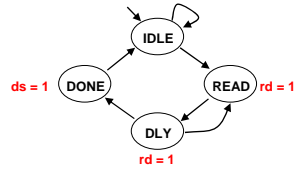
```

```

// CombLogic
always @(State, go, ws) begin
case (State)
IDLE: begin
if (go) StateNext <= READ;
else StateNext <= IDLE;
end
READ:
StateNext <= DLY;
DLY: begin
if (ws) StateNext <= READ;
else StateNext <= DONE;
end
DONE:
StateNext <= DONE;
endcase
end

assign {ds, rd} = State[1:0];
endmodule

```



State	x0	ds	rd
IDLE	0	0	0
READ	0	0	1
DLY	1	0	1
DONE	0	1	0

Output assignment based on state register

ECE 474a/575a
Susan Lysecky

17 of 19

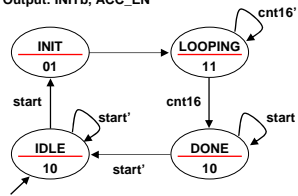
Registered Output FSM

Example 2

- Try output encoding method on same FSM

- Table with 5 rows (states+1) and 3 columns (outputs+1)
- List of FSM states
- List FSM outputs
- Fill in output values for each state
- Search for duplicate patterns
- Circle duplicate output patterns
- Add additional state bit(s) to create unique encodings
- Fill added columns with zeros in non-circled rows. Add binary encodings for redundant encodings.

Inputs: start, cnt16
Output: INITb, ACC_EN



State	start	cnt16
INIT	0	1
LOOPING	1	1
IDLE	1	0
DONE	1	0

Need 1 more state bit

State	x0	start	cnt16
INIT	0	0	1
LOOPING	0	1	1
IDLE	1	1	0
DONE	0	1	0

ECE 474a/575a
Susan Lysecky

18 of 19

