EECS151 L07 FSMS

September 7, 2021, EE Times: 5G Takes to the Stars

Get ready to never have an excuse to be off the grid again. The latest update to the 5G New Radio (5G NR) standard will enable compatible devices to connect with 5G capable satellites anywhere in the world, without requiring specialist phones to get networked.

Review

- **Combinational logic:**
  - The outputs only depend on the current values of the inputs (memoryless)
  - The functional specification of a combinational circuit can be expressed as:
    - A truth table
    - A Boolean equation
  - **Boolean algebra**
    - Deal with variables that are either True or False
    - Map naturally to hardware logic gates
    - Use theorems of Boolean algebra and Karnaugh maps to simplify equations
  - **Finite state machines:** Common example of sequential logic
  - **Common job interview questions:**

Sequential logic

- **Combinational logic:**
  - Memoryless: the outputs only dependent on the current inputs.
- **Sequential logic:**
  - Memory: the outputs depend on both current and previous values of the inputs.
  - Distill the prior inputs into a smaller amount of information, i.e., states.
  - **State:** the information about a circuit
    - Influences the circuit's future behavior
    - Stored in Flip-flops and Latches
  - **Finite State Machines:**
    - Useful representation for designing sequential circuits
    - As with all sequential circuits: output depends on present and past inputs
    - We will first learn how to design by hand then how to implement in Verilog.

FSM Example

- **Cat Brain (Simplified…)**
  - **Inputs:**
    - Feeding
    - Petting
  - **Outputs:**
    - Eyes: open or close
    - Mouth: open or close
  - **States:**
    - Eating
    - Sleeping
    - Annoyed...

FSM State Transition Diagram

- **States:**
  - Circles
- **Outputs:**
  - Labeled in each state
- **Arcs**
  - Inputs:
    - Arcs

FSM Symbolic State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eat</td>
<td>Feeding</td>
<td>Eat</td>
</tr>
<tr>
<td>Eat</td>
<td>Petting</td>
<td>Sleep</td>
</tr>
<tr>
<td>Sleep</td>
<td>Feeding</td>
<td>Sleep</td>
</tr>
<tr>
<td>Sleep</td>
<td>Petting</td>
<td>Annoyed</td>
</tr>
<tr>
<td>Annoyed</td>
<td>Feeding</td>
<td>Eat</td>
</tr>
<tr>
<td>Annoyed</td>
<td>Petting</td>
<td>Annoyed</td>
</tr>
</tbody>
</table>

FSM Encoded State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input Encoding</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eat</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>Sleep</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>Annoyed</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

\[
S_0 = S_1 \cdot X + S_1 \cdot X + S_1 + S_1 \cdot X = S_1 + S_1 \cdot X \\
S_1 = S_1 \cdot X + S_1 \cdot X = (S_1 \cdot X)
\]
FSM Output Table

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eat</td>
<td>00</td>
</tr>
<tr>
<td>Sleep</td>
<td>01</td>
</tr>
<tr>
<td>Annoyed</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Outputs Encoding

<table>
<thead>
<tr>
<th>Output</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyes</td>
<td>Open</td>
</tr>
<tr>
<td>Mouth</td>
<td>Open</td>
</tr>
<tr>
<td></td>
<td>Close</td>
</tr>
<tr>
<td></td>
<td>Open</td>
</tr>
</tbody>
</table>

E = S1'S0 + S1'S0 M = S1 S0

FSM Gate Representation

FSM Design Process

- Specify circuit function
- Draw state transition diagram
- Write down symbolic state transition table
- Write down encoded state transition table
- Derive logic equations
- Derive circuit diagram
  - Register to hold state
  - Combinational logic for next state and outputs

FSM State Encoding

- Binary encoding:
  - i.e., for four states, 00, 01, 10, 11
- One-hot encoding
  - One state bit per state
  - Only one state bit TRUE at once
  - i.e., for four states, 0001, 0010, 0100, 1000
  - Requires more flip-flops
  - Often next state and output logic can be simpler

Administrivia

- Homework 3 is due next Monday
  - Homework 4 will be posted this week, due before midterm 1
- Lab 4 this week
- Lab 5 next week
- Midterm 1 on October 7, 7-8:30pm

Moore and Mealy FSMs

Moore’s vs Mealy’s FSMs

- Next state is always determined by current state and inputs
- Differ in output logic:
  - Moore FSM: outputs depend only on current state
  - Mealy FSM: outputs depend on current state and inputs

Example: Edge Detector

- Input:
  - A bit stream that is received one bit at a time.
- Output:
  - 0/1
- Circuit:
  - Asserts its output to be true when the input bit stream changes from 0 to 1.
State Transition Diagram Solution A

Input | Current State | Next State | Output
--- | --- | --- | ---
0 | Zero (00) | Zero | 0
1 | Change (01) | Zero | 1
0 | Change (01) | One | 1
1 | One (11) | Zero | 0
1 | One (11) | One | 0

State Transition Diagram Solution B

Input | Current State | Next State | Output
--- | --- | --- | ---
0 | Zero (0) | Zero | 0
1 | Zero (0) | One | 1
0 | One (1) | Zero | 0
1 | One (1) | One | 0

Edge Detection Timing Diagrams

- Solution A (Moore): both edges of output follow the clock
- Solution B (Mealy): output rises with input rising edge and is asynchronous wrt the clock, output fails synchronous with next clock edge

FSM Comparison

Solution A
- Moore Machine
- Output function only of current state
- Maybe more states (why?)
- synchronous outputs
- Input glitches not sent to output
- one cycle "delay"
- Full cycle of stable output

Solution B
- Mealy Machine
- Output function of both current = & input
- Maybe fewer states
- Asynchronous outputs
  - If input glitches, so does output
  - output immediately available
  - Output may not be stable long enough to be useful (below):

If output of Mealy FSM goes through combinational logic before being registered, the CS might delay the signal and it could be missed by the clock edge (or violate setup time requirement)

Quiz: Which of the diagrams are Moore machines?

A. AC  B. BD  C. AD  D. BC

FSMs in Verilog

www.yellkey.com/oil
Implement FSM with Verilog

- Specify circuit function
- Draw state transition diagram
- Write down symbolic state transition table
- Assign encodings (bit patterns) to symbolic states
- Code as Verilog behavioral description
  - Use parameters to represent encoded states
  - Use separate always blocks for register assignment and combinational logic block
- Use case statement for combinational logic
- Within each case section (state), assign outputs and next state based on inputs
- Moore: outputs only dependent on states not on inputs

Finite State Machine in Verilog

module FSM1(clk, rst, in, out);
inout clk, rst; // Must use least to force to initial state.
output out;
input in;
wire clk, rst;
parameter S1 = 2'b10;
parameter S0 = 2'b01;
parameter IDLE = 2'b00;

// Defined state encoding:
output out;
input in;
input clk, rst;
wire out;
input in;
input clk, rst;
constant S1 = 2'b10;
constant S0 = 2'b01;
constant IDLE = 2'b00;

// For each state define output and next state based on inputs
always @(in, current_state)
begin
  // always block for combinational logic portion
  case (current_state)
    S1: begin
      out = 1
      next_state = IDLE;
    end
    S0: begin
      out = 1'
b1;
      next_state = S0;
    end
    IDLE: begin
      out = 1
      next_state = S1;
    end
    default: begin
      out = 1
      next_state = IDLE;
    end
  endcase

endmodule

Finite State Machine in Verilog (cont.)

begin
  // always block for combination logic portion
  always @(posedge clk)
  begin
    // For each state define output and next state
    case (current_state)
      IDLE: begin
        if (in == 1
          out = 1
          next_state = IDLE;
        end
      end
      S0: begin
        if (in == 1
          out = 1
          next_state = S0;
        end
      end
      S1: begin
        if (in == 1
          out = 1
          next_state = S1;
        end
      end
      default: begin
        out = 1
        next_state = IDLE;
      end
    endcase
  end
end

Endmodule

Finite State Machine in Verilog (cont.)

always @*
begin
  // For each state define output and next state
  case (state)
    IDLE: if (in == 1
      out = 1
      next_state = IDLE;
    end
    S0: if (in == 1
      out = 1
      next_state = S0;
    end
    S1: if (in == 1
      out = 1
      next_state = S1;
    end
    default: begin
      out = 1
      next_state = IDLE;
    end
  endcase
end

// For each state define output and next state
always @(posedge clk)
begin
  // For each state define output and next state
  case (current_state)
    IDLE: begin
      if (in == 1
        out = 1
        next_state = IDLE;
      end
    end
    S0: begin
      if (in == 1
        out = 1
        next_state = S0;
      end
    end
    S1: begin
      if (in == 1
        out = 1
        next_state = S1;
      end
    end
    default: begin
      out = 1
      next_state = IDLE;
    end
  endcase
end

Endmodule

Summary

- Finite state machines: Common example of sequential logic
  - Moore’s machine: Outputs depend only on the current state
  - Mealy’s machine: Output depends on the current state and the input
- Large state machines can be factored
- Common Verilog patterns for FSMs
- Common job interview questions Ø