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.

Artist’s rendering of an Inmarsat-6 satellite, which will support 5G. (Source: Inmarsat.)
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 😊
Finite-State Machines
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

---

**Eat**
- Eyes: T
- Mouth: T

**Sleep**
- Eyes: F
- Mouth: F

**Annoyed**
- Eyes: T
- Mouth: F

Inputs:
- Feeding
- Petting
# 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>

[Diagram showing state transitions: Eat → Feeding → Eat, Eat → Petting → Sleep, Sleep → Feeding → Sleep, Sleep → Petting → Annoyed, Annoyed → Feeding → Eat, Annoyed → Petting → Annoyed]
### FSM Encoded State Transition 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>Input X</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1'</td>
<td>S0'</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
S0' = \overline{S1}S0X + \overline{S1}S0\overline{X} = S1(S0X + S0\overline{X}) = S1(S0\oplus X)
\]

\[
S1' = \overline{S1}S0X + S1\overline{S0}X = (S1\oplus S0)X
\]

### State Encoding

- Eat: 00
- Sleep: 01
- Annoyed: 10

### Examples of State Transitions

- **Town:**
  - Current State: Eat
  - Input: Feeding
  - Next State: Eat
- **Road:**
  - Current State: Sleep
  - Input: Feeding
  - Next State: Sleep
- **Rural:**
  - Current State: Annoyed
  - Input: Feeding
  - Next State: Eat
- **Rural:**
  - Current State: Annoyed
  - Input: Petting
  - Next State: Annoyed
### 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>0</td>
</tr>
</tbody>
</table>

#### Outputs Encoding

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

\[
E = S1S0 + S1S0 = S0 \\
M = S1 S0
\]
FSM Gate Representation

\[ S_1' = X(S_0 \oplus S_1) \]
\[ S_0' = \overline{S_1}(S_0 \oplus X) \]
\[ E = \overline{S_0} \]
\[ M = \overline{S_1}S_0 \]
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.

000111010
Bitstream

Edge Detector

Edge Found

Nikolić, Fall 2021
**State Transition Diagram Solution A**

- **Input Current State**
- **Next State**
- **Output**

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Zero (00)</td>
<td>Zero</td>
<td>0</td>
</tr>
</tbody>
</table>
State Transition Diagram Solution A

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Zero (00)</td>
<td>Zero</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Zero (00)</td>
<td>Change</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Change (01)</td>
<td>Zero</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Change (01)</td>
<td>One</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>One (11)</td>
<td>Zero</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>One (11)</td>
<td>One</td>
<td>0</td>
</tr>
</tbody>
</table>
State Transition Diagram Solution A

CS = \{CS1, CS0\}

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Zero (00)</td>
<td>Zero</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Zero (00)</td>
<td>Change</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Change (01)</td>
<td>Zero</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Change (01)</td>
<td>One</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>One (11)</td>
<td>Zero</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>One (11)</td>
<td>One</td>
<td>0</td>
</tr>
</tbody>
</table>

CS = \{CS1, CS0\}

\[\text{NS}_0 = \text{IN}\]

\[\text{NS}_1 = \text{IN AND CS0}\]

\[\text{OUT} = \overline{\text{NOT(CS1) AND CS0}}\]
State Transition Diagram Solution B

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Zero (0)</td>
<td>Zero</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Zero (0)</td>
<td>One</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>One (1)</td>
<td>Zero</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>One (1)</td>
<td>One</td>
<td>0</td>
</tr>
</tbody>
</table>
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 CL 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

www.yellkey.com/oil
FSMs in Verilog
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

State Transition Diagram

Circuit Diagram

Holds a symbol to keep track of which bubble the FSM is in.

\[ \text{CL functions to determine output value and next state based on input and current state.} \]

\[ \text{out} = f(\text{in, current state}) \]

\[ \text{next state} = f(\text{in, current state}) \]
module FSM1(clk, rst, in, out);
input clk, rst;
input in;
output out;

// Defined state encoding:
parameter IDLE = 2'b00;
parameter S0 = 2'b01;
parameter S1 = 2'b10;

reg out;
reg [1:0] current_state, next_state;

// always block for state register
always @(posedge clk)
  if (rst) current_state <= IDLE;
  else current_state <= next_state;

A separate always block should be used for combination logic part of FSM. Next state and output generation. (Always blocks in a design work in parallel.)
Finite State Machine in Verilog (cont.)

```verilog
// always block for combinational logic portion
always @(current_state or in)
case (current_state)
    // For each state def output and next
    IDLE   : begin
        out = 1'b0;
        if (in == 1'b1) next_state = S0;
        else next_state = IDLE;
    end
    S0     : begin
        out = 1'b0;
        if (in == 1'b1) next_state = S1;
        else next_state = IDLE;
    end
    S1     : begin
        out = 1'b1;
        if (in == 1'b1) next_state = S1;
        else next_state = IDLE;
    end
    default: begin
        next_state = IDLE;
        out = 1'b0;
    end
endcase
endmodule
```

For each state define:
- Output value(s)
- State transition

Use "default" to cover unassigned state. Usually unconditionally transition to reset state.
Finite State Machine in Verilog (cont.)

```verilog
always @(*) begin
    next_state = IDLE;
    out = 1'b0;
    case (state)
        IDLE   : if (in == 1'b1) next_state = S0;
        S0     : if (in == 1'b1) next_state = S1;
        S1     : begin
            out = 1'b1;
            if (in == 1'b1) next_state = S1;
        end
        default: ;
    endcase
end
endmodule
```

Nominal values: used unless specified below.

Within case only need to specify exceptions to the nominal values.

Note: The use of "blocking assignments" allow signal values to be "rewritten" (evaluated immediately), simplifying the specification.
Edge Detector Example

**Mealy Machine**

```verilog
always @(posedge clk)
  if (rst) ps <= ZERO;
  else ps <= ns;
always @(ps in)
  case (ps)
    ZERO: if (in) begin
      out = 1'b1;
      ns = ONE;
    end
    else begin
      out = 1'b0;
      ns = ZERO;
    end
    ONE: if (in) begin
      out = 1'b0;
      ns = ONE;
    end
    else begin
      out = 1'b0;
      ns = ZERO;
    end
    default: begin
      out = 1'bX;
      ns = default;
    end
```

**Moore Machine**

```verilog
always @(posedge clk)
  if (rst) ps <= ZERO;
  else ps <= ns;
always @(ps in)
  case (ps)
    ZERO: begin
      out = 1'b0;
      if (in) ns = CHANGE;
      else ns = ZERO;
    end
    CHANGE: begin
      out = 1'b1;
      if (in) ns = ONE;
      else ns = ZERO;
    end
    ONE: begin
      out = 1'b0;
      if (in) ns = ONE;
      else ns = ZERO;
    end
    default: begin
      out = 1'bX;
      ns = default;
    end
```
Summary

• Finite state machines: Common example of sequential logic
  • Moore’s machine: Output depends 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 😊