Logic Design
Design of Finite State Machines
(Chapter 3)

Quick summary…
• Combinational logic circuits
  • Basic gates (OR, AND, …)
  • Combinational devices (Multiplexers, Decoders, Adders,…)

• We now have storage devices
  • D-Latch, RS Latch, D flip-flop

• Implementation of registers and memory
  • Using D-latches

• ok…now back to sequential circuits and state machines
  • Recall examples of Vending Machine and Counter
  • Refer to example circuits in fsm-examples.cdl
    • The link Examples of Finite State Machines

3 “Storage” Devices

RS Latch – Stores 1 Bit, Level-Triggered
-1 “forbidden” input: S=0, R=0
-Holds Data when RS=11

D-Latch – Stores 1 Bit, Level-Triggered
-No “forbidden” inputs (fixes RS Latch)
-D=Q when WE=1
-Holds Data when WE=0

D-Flip-Flop – Stores 1 Bit, Edge-Triggered
-No “forbidden” inputs
-D=Q when WE (CLK) transitions from 0 to 1
-Holds Data for WE=1 or WE=0
-Except when WE transitions from 0 to 1

Clocked Flip-Flops/Circuits
• Subsystem in a computer consists of a large number of combinational and sequential devices
  • Each sequential device is like latch which is in one of two states
  • As machine executes its cycle, the states of all sequential devices change with time

• To control large collection of devices in an orderly (synchronized) fashion, machine maintains a clock
  • Requires all devices to change their states at the same time
  • Clock generates sequence of pulses
  • Much easier to design, debug, implement, and test

• How do we change latches so that they allow change in state synchronized with the clock?
• Sequential logic circuits require a means by which events can be sequenced…..clock!
States in a FSM

- The concept of state
  - The state of a system is a "snapshot" of all relevant elements at a moment in time.
  - A given system will often have only a finite number of possible states.
  - For many systems, we can define the rule which determines under what conditions a system can move from one state to another.
    - So when do they change states?
      - Random times?
      - Only at specific times?

Finite State Machine

- In general a Finite State Machine consists of
  - An n-bit storage (register?) which stores the state of the machine
  - A block of logic that computes the next state as a function of the current state and the inputs, if any
  - A block of logic which computes the output based on the current state.

Latches and Flip-Flops

- Latch: basic circuit for storage
  - Operate on changes in Level (i.e., 1 or 0)
- Flip-flop:
  - Sequential circuits take input from output of storage
  - Latches that work on change of level can lead to unstable sequential circuits
    - As level changes the outputs change --- inputs change!
  - Flip-Flop circuits designed to operate properly when they are part of a sequential circuit
    - Flip flop becomes basic circuit for storage in clocked (synchronous) sequential circuits!

D Flip-Flop

- We can think of the D Flip-Flop as a 1 bit storage container with an input, D, and an output, Q.
- The D flip-flop takes a clock input (often denoted with a triangle)
- A set of D flip-flops can be grouped together with common Clock and WE inputs to form a register. A key component in our processor
One Last Thing…
D Flip-Flop with Additional Write Enable

• From previous slides, we attached clock to WE of the D-flip-flop
• Now, we add another WE line to the flip flop
  ➢ Just holds onto data already stored in DFF
• Give it the ability to “ignore” the clock!

Next: Design methodology for sequential logic circuits

• We have storage devices and method for synchronization
  • Flip flops and Clock
  • We can map from Finite state machine diagram (graph) to truth table
  • and we now have a storage device to store the state!

Examples

• Page 1 of fsm-examples.cdl

State Machine

• Type of sequential circuit
  • Combines combinational logic with storage
  • "Remembers” state, and changes output and state at each clock cycle based on inputs and current state
Finite State Machine
- The counter we designed is an example of a finite state machine.
- In general a Finite State Machine consists of
  - An n-bit register which stores the state of the machine
  - A block of logic that computes the next state as a function of the current state and the inputs, if any
  - A block of logic which computes the output based on the current state.

States in a FSM
- The concept of state
  - The state of a system is a "snapshot" of all relevant elements at a moment in time.
  - A given system will often have only a finite number of possible states.
  - For many systems, we can define the rule which determines under what conditions a system can move from one state to another.
    - So when do they change states?
      - Clock!
  - Encode each state with a binary label

Designing and implementing a FSM
1. First draw the state diagram
   - Encode each state in binary using N bits
   - These N bits correspond to N "state variables" that need to be stored. Call them $S_{N-1} S_{N-2} \ldots S_1 S_0$
   - State diagram will show transitions from state to state based on value of inputs
2. Next, derive the truth table (from state diagram)
   - "Inputs" in truth table are N current state variables and the inputs
   - "Outputs" are the values of the state variables in the next state and the output at each state -- common notation is $S'$ but confusion with complement operator, so let's use $S^*$
3. From truth table, derive combinational circuit (boolean function) for each of the next state values
   - State variables are stored in your N storage elements

Counter from 0 to 3 (2-bit counter)
- Circuit has a "ON" switch (input)
  - If ON=0 then machine goes to 00
  - 00
  - 01
  - 11
  - 10
  - Clock
**FSM Representation of 3-bit Counter**

- **Bubbles** represent all possible "states" for the machine (aka your flip-flop based circuit).
- **Arrows** show movement from one state to the next.
- **Transitions** occur at pulse of the clock.

**Storage**

- Each D flip flop stores one state bit.
- The number of storage elements (flip-flops) needed is determined by the number of states (and the representation of each state).
  - Each bit can be 0 or 1 = 2 states
  - N bits can represent \(2^N\) states
- Example: If a FSM has 12 states, then the circuit needs \(\log_2{12} = 4\) storage elements.
  - Fewer states, less hardware needed
  - Concept of Minimization of States for a given FSM

**Designing and implementing a FSM**

1. **First draw the state diagram**
   - Encode each state in binary using N bits
   - These N bits correspond to N "state variables" that need to be stored. Call them \(S_{N-1}, S_{N-2}, \ldots, S_1, S_0\)
   - State diagram will show transitions from state to state based on value of inputs
2. **Next, derive the truth table (from state diagram)**
   - "inputs" in truth table are N current state variables and the inputs
   - "outputs" are the values of the state variables in the next state and the output at each state -- common notation is \(S'\) but confusion with complement operator, so let's use \(S^*\)
3. **From truth table, derive combinational circuit (boolean function) for each of the next state values**
   - State variables are stored in your N storage elements

**Truth Table for a 2-bit Counter**

1. **Derive Truth table**
   - Using ON, A, B as inputs and A*, B* as outputs
   
<table>
<thead>
<tr>
<th>ON</th>
<th>A</th>
<th>B</th>
<th>A*</th>
<th>B*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Designing and implementing a FSM

1. First draw the state diagram
   - Encode each state in binary using N bits
   - These N bits correspond to N “state variables” that need to be stored. Call them $S_{N-1}, S_{N-2}, \ldots, S_0$
   - State diagram will show transitions from state to state based on value of inputs

2. Next, derive the truth table (from state diagram)
   - “inputs” in truth table are N current state variables and the inputs
   - “outputs” are the values of the state variables in the next state and the output at each state — common notation is $S'$ but confusion with complement operator, so let’s use $S^*$

3. From truth table, derive combinational circuit (boolean function) for each of the next state values
   - State variables are stored in your N storage elements

---

**Truth Table Representation of 3-bit Counter**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_2(t)$</td>
<td>$D_1(t)$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Functions and Circuit for a 2-bit Counter**

**functions:**
- $A, B$ current state
- $A^*, B^*$ next state

- $A^* = ON.(A'B) + (A B') = ON.(A \ XOR B)$
- $B^* = ON.(A'B') + (AB') = ON.B'$

**Example 2-bit Counter**

- Page 9
- Output of the two flip flops are current state variables
  - $A$ is top flip flop, $B$ is lower one
- Input to Flip flop for $A$ is function for $A^*$
  - i.e., output of circuit for $A^*$
- Input to flip flop for $B$ is function for $B^*$
  - i.e., output of circuit for $B^*$

**next state functions:**
- $A, B$ current state
- $A^*, B^*$ next state

- $A^* = (A'B) + (A B') = A \ XOR B$
- $B^* = (A'B') + (AB') = B'$

---

Functions and Circuit for a 2-bit Counter

<table>
<thead>
<tr>
<th>ON</th>
<th>A</th>
<th>B</th>
<th>$A^*$</th>
<th>$B^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example 2-bit Counter

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$A^*$</th>
<th>$B^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Example Circuit for 2-bit counter**

- Page 2 of fsm-examples.cdl
- Implements the circuits from the truth table:
  - A, B current state: stored in two D flip flops
  - Read current state from output of D flip flop
  - A*, B* next state
    - Write next state into input of Flip flop – will be written at end of clock cycle
  - A* = (A'B) + (A B') = A XOR B
  - B* = (A'B') + (AB') = B'

**Truth Table Representation of 3-bit Counter**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2 (t)</td>
<td>D1 (t)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**3-bit counter: Boolean functions for values of next state**

- For each state variable D_i, derive function that determines next state D'_i, for that variable:
- Note: D_2 (t+1) also denoted as D_2' or D_2^*．
- Current states: D_2 D_1 D_0
- Complete the circuit on Page 4 of fsm-examples.cdl

**Example: Blinking Traffic Sign**

- (from textbook) Design control circuitry for a blinking traffic sign (to show “move right” message):
- Page 3 of fsm-examples.cdl shows implementation
Example: Traffic Sign

- A blinking traffic sign: How many lights = 5
- How many states?
- 4 states
  - No lights on
  - 1 & 2 on
  - 1, 2, 3, & 4 on
  - 1, 2, 3, 4, & 5 on
  - (repeat as long as switch is turned on)
- How many bits to represent the 4 states
  - $S_1S_0$
    - With $S_1S_0$ values: 00, 01, 10, 11
- How many 'outputs' (to control the 5 lights) = 5?
  - If the sign is switched off then all lights turn off

Traffic Sign State Diagram

Transition on each clock cycle.

Outputs

- Note we really have 3 groups of lights to be controlled = 3 control lines X, Y, Z
  - Group 1: Lights 1 and 2; controlled by Z
    - If Z=1 then Group 1 lights (1 and 2) are switched on
  - Group 2: lights 3 & 4; controlled by Y
  - Group 3: Light 5; controlled by X
- In this example, we associate each state with an output
  - Depending on the current state, we switch on specific groups of lights
• When is group 1 on?
  • in states 01, 10 and 11 - but only when the switch IN is on!
  • Logic expressions for X,Y,Z
    • Depends on S0 and S1 and Input is on
      > if Input is off then X,Y,Z are all 0
    • can you come up with a logic expression for next state values of S0 and S1?
      • Depends on current values of S0 and S1 and Input is on
      > Input off then both bits are set to 0 since next state is 00
      • Next state value of S0 denoted S’0 = 1 if current state is 00 or current state 10 and In=1
• When do we switch to the next state?
  • the two bits of S[1:0] are updated at every clock cycle
  • we have to make sure that the new state does not propagate to the combinational circuit input until the next clock cycle.

---

### Traffic Sign Truth Tables

<table>
<thead>
<tr>
<th>S1 S0</th>
<th>Z</th>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Switch</th>
<th>In</th>
<th>S0</th>
<th>S1</th>
<th>S’0</th>
<th>S’1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Whenever In=0, next state is 00.

---

### Boolean functions for light control bits

• from truth table, consider all rows where outputs =1

  • Z = ((NOT S1)S0 + S1 (NOT S0) + S1S0 ).In
  • Y = (S1S0 + S1 (NOT S0) ).In
  • X = (S1S0 ).In

---

### Traffic Sign Logic
**Summary of Digital Logic**

- Combinational logic
  - Basic gates
  - Combinational devices

- Sequential logic
  - Storage element...Flip flop
  - Theory behind design of finite state machines
    - They act like controllers of a circuit

- Sequential logic devices
  - Memory, ROM, RAM, Registers

So, what is the purpose of all this?........

**The Agenda**

- Take the elements that we have encountered so far
  - Combinational Elements
    - Gates, Adders, Muxes, decoders
  - Storage Elements
    - Flip flops, registers, memories

- And use them to build a circuit that can perform a sequence of arithmetic operations.
  - In essence, we will build a very simple CPU