Today

- Designing Digital System
- Efficient Hardware Design
- HDL Simulation
- Blocking vs. Non-Blocking
- Administrative Info
- Lab #3: The Combo Lock
- FSMs in Verilog

Designing Digital System (1)

- High Level Design
  - Top-Down Design
  - Partitioning & Interfaces
- Implementing the Partitioned Digital Logic
  - Start with the formal description
  - Identify Inputs
  - Determine State
  - Generate Outputs

Designing Digital System (2)

- Formal descriptions
  - Mathematical Logic
  - Data Flow Model
  - Petri Net
  - Finite State Machine
  - ...

Implementation of FSMs

EECS150 Spring 2006 – Lab Lecture #3
Guang Yang
Greg Gibeling
Designing Digital System (3)

Example: Finite State Machine

<s, s₀, i, o, t>
S: States
S₀: Initial State
I: Inputs
O: Outputs
T: Transition Function

Designing Digital System (4)

- Identify Inputs
  - What are they?
    - Possible Values and Don’t Cares
    - Timing
  - Process Them
    - Raw inputs are often not what you need
    - Might need delay/timing change
    - Might look for a specific value/range

Designing Digital System (5)

- Determine States
  - What does the module need to remember?
    - Has it seen a particular input?
    - How many cycles have passed?
  - Design Memory for State
    - Standard D Register
    - Counter
    - Shift Register

Designing Digital System (6)

- Generate Outputs
  - What are they?
    - Possible Values
    - Timing
  - Create the outputs
    - They’re always there
    - Compute them from state (and inputs)
    - Learn to think in Boolean equations
    - assign is helpful
Designing Digital System (7)

- Mealy Machines
  - Output based on input and current state
  - Can have major timing problems
- Moore Machines
  - Output based on current state
  - Easier to work with
  - Slightly harder to build

Efficient Hardware Design (1)

```
always @ (a or A or B or C) begin
  if (a) aux = B; else aux = C; Z = A + aux;
end
```

```
always @ (a or A or B or C) begin
  if (a) Z = A + B;
  else Z = A + C;
end
```

Efficient Hardware Design (2)

```
assign B = 3;
asign Z = A + B;
asign Z = A + (2 * A);
asign Z = A + (A << 1); 
asign Z = A + {A, 1'b0};
```

Efficient Hardware Design (3)

```
assign aux = A + {1'b0, A[n-1:1]};
asign Z = {aux, A[0]};
```

```
A[n-1:1]
\hline
n bit adder
\hline
\hline
A[0]
\hline
\hline
aux
\hline
\hline
Z
```
### HDL Simulation (1)

- **Software Based Simulation**
  - Simple and accurate
  - Allows for simulation at any precision
  - Easy to see any signal - perfect Visibility

- **Drawbacks**
  - Slow
  - Simulator Dependant
  - Deadlocks are Possible!

- **Simulation != Synthesis**

### HDL Simulation (2)

- **Event Driven Simulation**
  - Virtual time axis
  - Maintain a queue of events
  - Pull next event off the queue
  - Determine its consequences
  - Add more events to the queue

- **Implications**
  - Verilog is not executed!
    - Things don’t necessarily happen in order
  - Verilog is SIMULATED

### Blocking vs. Non-Blocking (1)

<table>
<thead>
<tr>
<th>Verilog Fragment</th>
<th>Result</th>
<th>Hardware</th>
</tr>
</thead>
</table>
| always @ (a) begin  
b = a;  
c = b;  
end | C = B = A | A —— B —— C |
| always @ (posedge Clock) begin  
b <= a;  
c <= b;  
end | B = A  
C = Old B | B —— D —— C |

### Blocking vs. Non-Blocking (2)

- **Use Non-Blocking for FlipFlop Inference**
  - posedge/negedge require Non-Blocking
  - Else simulation and synthesis won’t match

- **Use #1 to show causality**

```verilog
clock
always @ (posedge Clock) begin  
b <= #1 a;  
c <= #1 b;  
end
```
Administrative Info

- You could get checked off during your lab or during any lab TA’s office hours, though the TA would give higher priority to his own students
- Or in the first 10 minutes in your next lab session
- Try NOT to get checked off in other labs. As the lab gets busier, you may not be checked off at all

Administrative Info (2)

- Cardkey forms distributed on 2/2 lecture
  - See the office in Cory 253 or Soda 390
- To download files from course website, cs150-temp no longer works
  - If you cs150-xx does not work either, please email David

Administrative Info (3)

- Partners
  - You MUST have one for Lab4 and later…
    - Try to keep the same one for the project
    - You must have one in your lab section
  - If you do not have a partner:
    - See a TA right after this lab lecture
    - Post to the newsgroup

Lab #3: The Combo Lock (1)

- Used to control entry to a locked room
  - 2bit, 2 digit combo (By Default 11, 01)
  - Set code to 11, Press Enter
  - Set code to 01, Press Enter
  - Lock Opens (Open = 1)
Lab #3: The Combo Lock (2)

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Dir</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>2</td>
<td>I</td>
<td>Code from the dipswitches</td>
</tr>
<tr>
<td>Enter</td>
<td>1</td>
<td>I</td>
<td>Enter button (examine the code)</td>
</tr>
<tr>
<td>ResetCombo</td>
<td>1</td>
<td>I</td>
<td>Reset to the default combination</td>
</tr>
<tr>
<td>Clock</td>
<td>1</td>
<td>I</td>
<td>System Clock</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>I</td>
<td>System Reset, doesn’t affect the combo</td>
</tr>
<tr>
<td>Open</td>
<td>1</td>
<td>O</td>
<td>Indicates the lock is open</td>
</tr>
<tr>
<td>Error</td>
<td>1</td>
<td>O</td>
<td>Indicates a bad combination</td>
</tr>
<tr>
<td>Prog1</td>
<td>1</td>
<td>O</td>
<td>Reprogramming the first digit</td>
</tr>
<tr>
<td>Prog2</td>
<td>1</td>
<td>O</td>
<td>Reprogramming the second digit</td>
</tr>
<tr>
<td>LED</td>
<td>8</td>
<td>O</td>
<td>Use these for debugging</td>
</tr>
</tbody>
</table>

Lab #3: The Combo Lock (3)

- Example 1:
  1: Press ResetCombo, Combo: 2'b11, 2'b01
  2: Set 2'b11, Press Enter
  3: Set 2'b01, Press Enter, LEDs: “OPEN”
  4: Press Enter, LEDs: “Prog1”
  5: Set 2'b00, Press Enter, LEDs: “Prog2”
  6: Set 2'b10, Press Enter, LEDs: “OPEN”
  7: Combo: 2'b00, 2'b10

Lab #3: The Combo Lock (4)

- Example 2:
  1: Press ResetCombo, Combo: 2'b11, 2'b01
  2: Set 2'b01, Press Enter
  3: Set 2'b01, Press Enter, LEDs: “Error”
- Why doesn’t “Error” show until step 3?

Lab #3: The Combo Lock (5)
Lab #3: The Combo Lock (6)

Lab #3: The Combo Lock (7)

- Debugging with LEDs
  - A powerful way to debug
  - Easy to understand
  - Lower overhead than other debugging tools
  - A great way to see NextState/CurrentState

- Drawbacks
  - Slow, can't see fast events
  - No timing information, no waveform
  - Limited number
    - Dipswitches!

FSMs in Verilog (1)

- Mealy Machines
  - Output based on input and current state
  - Can have major timing problems
- Moore Machines
  - Output based on current state
  - Easier to work with
  - Slightly harder to build

FSMs in Verilog (2)

- Two or Three always blocks
  - 1st: CurrentState Register
    - Clocked
    - Handles Reset
  - 2nd: Generates NextState (+ Outputs in Mealy)
    - Uses CurrentState and Inputs
    - Combinational
  - 3rd: Generates Outputs (Optional)
    - Uses CurrentState only (for Moore Machines)
    - Might be replaced with a few assigns
FSMs in Verilog (3)

module MyFSM(In, Out, Clock, Reset);
  input   In, Clock, Reset;
  output  Out;
  parameter STATE_Idle = 1'b0,
              STATE_Run = 1'b1;
  reg      CurrentState, NextState, Out;

  always @ (posedge Clock) begin
    if (Reset) CurrentState <= STATE_Idle;
    else CurrentState <= NextState;
  end
endmodule

FSMs in Verilog (4)

... always @ (CurrentState or In) begin
  NextState = CurrentState;
  Out = 1'b0;
  // The case block goes here
  // Its on the next slide...
end
endmodule

FSMs in Verilog (5)

case (CurrentState)
  STATE_Idle: begin
    if (In) NextState = STATE_Run;
    Out = 1'b0;
  end
  STATE_Run: begin
    if (In) NextState = STATE_Idle;
    Out = 1'b1;
  end
  default: begin
    NextState = STATE_X;
    Out = 1'bX;
  end
endcase