Intro to Digital Design
Verilog Basics

Instructor: Justin Hsia

Teaching Assistants:
Caitlyn Rawlings
Emilio Alcantara
Naoto Uemura

Donovan Clay
Joy Jung
Relevant Course Information

❖ Lab demo slots have been assigned on Canvas

❖ Lab 1 & 2 – Basic Logic and Verilog
  ▪ Digit(s) recognizer using switches and LED
  ▪ For full credit, find minimal logic
  ▪ Check the lab report requirements closely

❖ If you haven’t done so yet, pick up a lab kit ASAP from CSE 003 when TAs are present (labs + support hours)
  ▪ White lab kit + Okiocam
Practice Question:

- Write out the Boolean Algebra expression for Q for the following circuit. No simplification necessary.
Practice Question:

- Implement the Boolean expression $B(A + C)$ with the fewest number of a single universal gate. What does your solution look like?
Lecture Outline

❖ Thinking About Hardware
❖ Verilog Basics
❖ Waveform Diagrams
❖ Debugging in Verilog
Verilog

❖ Programming language for describing hardware
  ▪ Simulate behavior before (wasting time) implementing
  ▪ Find bugs early
  ▪ Enable tools to automatically create implementation

❖ Syntax is similar to C/C++/Java, but behavior is very different
  ▪ VHDL (the other major HDL) is more similar to ADA

❖ Modern version is SystemVerilog
  ▪ Superset of previous; cleaner and more efficient
Verilog: Hardware Descriptive Language

- Although it looks like code:

```verilog
module myModule (F, A, B, C);
  output logic F;
  input logic A, B, C;
  logic AN, AB, AC;
  nand gate1(AB,AN,B);
  nand gate2(AC,A,C);
  nand gate3(F,AB,AC);
  not not1(AN,A);
endmodule
```

- Keep the hardware in mind:
Verilog Primitives

- **Nets** (*wire*): transmit value of connected source
  - Problematic if connected to two different voltage sources
  - Can connect to many places (always possible to “split” wire)

- **Variables** (*reg*): variable voltage sources
  - Can “drive” by assigning arbitrary values at any given time
  - SystemVerilog: variable *logic* can be used as a net, too

- **Logic Values**
  - 0 = zero, low, FALSE
  - 1 = one, high, TRUE
  - X = unknown, uninitialized, contention (conflict)
  - Z = floating (disconnected), high impedance
Verilog Primitives

❖ **Gates:**

<table>
<thead>
<tr>
<th>Gate</th>
<th>Verilog Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT a</td>
<td>(~a)</td>
</tr>
<tr>
<td>a AND b</td>
<td>a &amp; b</td>
</tr>
<tr>
<td>a OR b</td>
<td>a</td>
</tr>
<tr>
<td>a NAND b</td>
<td>(~(a &amp; b))</td>
</tr>
<tr>
<td>a NOR b</td>
<td>(~(a</td>
</tr>
<tr>
<td>a XOR b</td>
<td>a ^ b</td>
</tr>
<tr>
<td>a XNOR b</td>
<td>(~(a ^ b))</td>
</tr>
</tbody>
</table>

❖ **Modules:** “classes” in Verilog that define *blocks*

- **Input:** Signals passed from outside to inside of block
- **Output:** Signals passed from inside to outside of block
Verilog Execution

❖ Physical wires transmit voltages (electrons) near-instantaneously
  ▪ Wires by themselves have no notion of sequential execution

❖ Gates and modules are constantly performing computations
  ▪ Can be hard to keep track of!

❖ In pure hardware, there is no notion of initialization
  ▪ A wire that is not driven by a voltage will naturally pick up a voltage from the environment
Lecture Outline

- Thinking About Hardware
- **Verilog Basics**
- Waveform Diagrams
- Debugging in Verilog
Using an FPGA

Verilog

// Verilog code for 2-input multiplexer
module AOI (F, A, B, C, D);
output F;
input A, B, C, D;
assign F = ~((A & B) | (C & D));
endmodule

module MUX (V, SEL, I, J);
// 2:1 multiplexer
output logic V;
input logic SEL, I, J;
logic SELB, VB;
not G1 (SELB, SEL);
AOI G2 (VB, I, SEL, SELB, J);
not G3 (V, VB);
endmodule

FPGA CAD Tools

Bitstream

00101010001010010
10010010010011000
10101000101011000
10101001010010101
00010110001001010
10101001111001001
01000010101001010
1001001000101010
10100101010010100
0101010101001010
010100100101001
// Verilog code for AND-OR-INVERT gate

module AOI (F, A, B, C, D);
    output logic F;
    input logic A, B, C, D;

    assign F = ~(A & B) | (C & D);
endmodule

// end of Verilog code
Verilog Wires

// Verilog code for AND-OR-INVERT gate

module AOI (F, A, B, C, D);
  output logic F;
  input logic A, B, C, D;
  logic AB, CD, O;  // now necessary

  assign AB = A & B;
  assign CD = C & D;
  assign O = AB | CD;
  assign F = ~O;
endmodule
Verilog Gate Level

// Verilog code for AND-OR-INVERT gate

module AOI (F, A, B, C, D);
    output logic F;
    input logic A, B, C, D;
    logic AB, CD, O;  // now necessary
    assign AB = A & B;
    assign CD = C & D;
    assign O = AB | CD;
    assign F = ~O;
endmodule
Verilog Hierarchy

// Verilog code for 2-input multiplexer

module AOI (F, A, B, C, D);
    output logic F;
    input logic A, B, C, D;
    assign F = ~(A & B) | (C & D);
endmodule

module MUX2 (V, SEL, I, J);  // 2:1 multiplexer
    output logic V;
    input logic SEL, I, J;
    logic SELN, VN;

    not G1 (SELN, SEL);
    AOI G2 (.F(VN), .A(I), .B(SEL), .C(SELN), .D(J));
    not G3 (V, VN);
endmodule
Technology Break
Lecture Outline

❖ Thinking About Hardware
❖ Verilog Basics
❖ **Waveform Diagrams**
❖ Debugging in Verilog
Signals and Waveforms

- **Signals** transmitted over wires continuously
  - Transmission is effectively instantaneous (a wire can only contain one value at any given time)
  - In digital system, a wire holds either a 0 (low voltage) or 1 (high voltage)

Stack multiple signals in same **waveform diagram** vertically (syncing times)
Signal Grouping

A group of wires when interpreted as a bit field is called a *bus*

“undefined” (unknown) signal
Circuit Timing Behavior

❖ **Simple Model:** Gates “react” after fixed delay
❖ **Example:** Assume delay of all gates is 1 ns (= 3 ticks)
Circuit Timing: Hazards/Glitches

- Circuits can temporarily go to incorrect states!
  - Assume 1 ns delay (3 ticks) for all gates

![Circuit Diagram]

<table>
<thead>
<tr>
<th>CAR</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC</td>
<td>1</td>
</tr>
<tr>
<td>PAR</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td></td>
</tr>
</tbody>
</table>
Verilog Buses

// Verilog code for AND-OR-INVERT gate

module AOI (F, A, B, C, D);
    output logic F;
    input logic A, B, C, D;
    logic [2:0] w;  // necessary

    assign w[0] = A & B;
    assign w[1] = C & D;
    assign w[2] = w[0] | w[1];
    assign F = ~w[2];
endmodule

Just for illustration – this is bad coding style!
Verilog Signal Manipulation

- **Bus definition:** \([n-1:0]\) is an \(n\)-bit bus
  - Good practice to follow bit numbering notation
  - Access individual bit/wire using “array” syntax (\(e.g., \text{bus}[1]\))
  - Can access sub-bus using similar notation (\(e.g., \text{bus}[4:2]\))

- **Multi-bit constants:** \(n'b#…#\)
  - \(n\) is width, \(b\) is radix specifier (\(b\) for binary), \(#\)s are digits of number
  - \(e.g., \text{4'd12, 4'b1100, 4'hC}\)

- **Concatenation:** \{\text{sig, …, sig}\}
  - Ordering matters; result will have combined widths of all signals

- **Replication operator:** \{\text{n\{m\}}\}
  - repeats value \(m\), \(n\) times
Practice Question

```
logic [4:0] apple;
logic [3:0] pear;
logic [9:0] orange;
assign apple = 5'd20;
assign pear = {1'b0, apple[2:1], apple[4]};
```

❖ What’s the value of pear?

❖ If we want orange to be the *sign-extended* version of apple, what is the appropriate Verilog statement?

```
assign orange =
```
Lecture Outline

❖ Thinking About Hardware
❖ Verilog Basics
❖ Waveform Diagrams
❖ **Debugging in Verilog**
Testbenches

- Needed for simulation only!
  - Software constraint to mimic hardware

- ModelSim runs entirely on your computer
  - Tries to simulate your FPGA environment without actually using hardware – no physical signals available
  - Must create fake inputs for FPGA’s physical connections
    - e.g., LEDR, HEX, KEY, SW, CLOCK_50
  - Unnecessary when code is loaded onto FPGA

- Need to define both input signal combinations as well as their timing
Verilog Testbenches

module MUX2_tb ();
logic SEL, I, J; // variables remember values
logic V; // acts as net for reading output

initial // build stimulus (test vectors)
begin // start of "block" of code
  {SEL, I, J} = 3'b100; #10; // t=0: S=1, I=0, J=0 -> V=0
  I = 1; #10; // t=10: S=1, I=1, J=0 -> V=1
  SEL = 0; #10; // t=20: S=0, I=1, J=0 -> V=0
  J = 1; #10; // t=30: S=0, I=1, J=1 -> V=1
end // end of "block" of code

"Device Under Test"
MUX2 dut (.V, .SEL, .I, .J);

endmodule // MUX2_tb
Better Verilog Testbench

module MUX2_tb ();

logic SEL, I, J; // registers remember values
logic V; // acts as net for reading output

int i;
initial // build stimulus (test vectors)
begin // start of "block" of code
for(i = 0; i < 8; i = i + 1) begin
    {SEL, I, J} = i; #10;
end
end // end of "block" of code

MUX2 DUT (.V, .SEL, .I, .J);
endmodule // MUX2_tb
Debugging Example (Demo)

- **Check MUX2_tb waveforms:**

- **Identify buggy component(s) and dive deeper as necessary until bug is found**
  - Demo: check NOT gates then check AND, OR gates
Debugging Circuits

- Complex circuits require careful debugging
  - Test as you go; don’t wait until the end (system test)
  - *Every* module should have a testbench (unit test)

1) Test all behaviors
   - All combinations of inputs for small circuits, subcircuits

2) Identify any incorrect behaviors

3) Examine inputs & outputs to find earliest place where value is wrong
   - Typically trace backwards from bad outputs, forwards from inputs
   - Look at values at intermediate points in circuit
Hardware Debugging

- Simulation (ModelSim) is used to debug logic design and should be done thoroughly before touching FPGA
  - Unfortunately ModelSim is not a perfect simulator
- If interfacing with other circuitry (e.g., breadboard), will also need to debug circuitry layout there
  - Similar process, but with power sources (inputs) and voltmeters (probe the wires)
  - Often just a poor electrical connection somewhere
- Sometimes things simply fail
  - All electrical components fail eventually, whether you caused it to or not
Summary

❖ Verilog is a hardware description language (HDL) used to program your FPGA
  ▪ Programmatic syntax used to describe the connections between gates and registers

❖ Waveform diagrams used to track intermediate signals as information propagates through CL

❖ Hardware debugging is a critical skill
  ▪ Similar to debugging software, but using different tools