Agenda

- Lab/Section Info
- Lab Overview
- Why care?
- Verilog Tips and Pitfalls
- Verilog <-> Hardware Examples
Lab / Section

- When announced, we’ll have section for the 1st hour of your scheduled lab section.
- Otherwise, lab section == office hours
  - TA(s) & SLAs will be available in 003
  - Attendance != required
  - Use time wisely to get help w/ difficult issues
- So, make sure you’re on the class e-mail list and check that account!
Lab

- Goal: Build pipelined MIPS processor capable of running compiled C code

- Four Tasks
  1. Build single-cycle datapath + create jump & branch logic
  2. Build control logic for single-cycle CPU
  3. Add pipeline registers
  4. Complete pipeline with forwarding & hazard detection

- 2 weeks to complete each part
- Highly suggest you work in pairs
First Task

- Mostly following instructions and connecting components in block diagram
- Writing some Verilog for jump & branch logic
- Test benches are provided, but they're not 100% robust
- Demonstration: write short assembly program to blink lights on board
The Hardware
The Hardware – Details

- FPGA: Altera Cyclone II EP2C20F484C7N
  - 18,752 4-input lookup tables
  - 18.752 1-bit registers (aka flip-flops)
  - 240 KB of memory (enough to store several frames of an ASCII-encoded rickroll)
Useful Tools

- Aldec's Active-HDL lets us simulate Verilog and block diagrams (BDEs)
  - Its assembler turns your code into bits and provides useful console output and waveforms for debugging
- Altera's Quartus software does 3 things:
  i. Translates Verilog to hardware primitives
  ii. Arranges hardware primitives on the chip
  iii. Programs design to chip
Why Lab Matters

- In 370 you designed a bunch of special-purpose circuits to fulfill one specific role.
- In 378 you'll design the best-known and one of the most-useful general-purpose circuits: the processor!
- Important to understand hardware that your high-level code runs on and abstractions between them.
- Companies think so too!
- If nothing else, you're required to take this class to get a degree in CS/CE.
Short Verilog Quiz

- What are the two types of logic?
  - Combinational & sequential
- Inside an always block, which assignment operator should you use for combinational logic (= or <=)? Sequential?
  - Combinational: =     Sequential: <=
- What's the syntax for declaring an 8-bit-wide bus named fooBar?
  - wire [7:0] fooBar;
module foo(a, b, f, g);
    input wire a, b;
    output wire f;
    output reg g;

    assign f = a & b;
    always @ (*) begin
        g = a & b;
    end
endmodule

Questions:

Are f and g the same?

What does "always @ (*)" mean?
Hardware Primitives – Register

module foo(clk, a, bar);
    input wire a, clk;
    wire nextBar;
    output reg bar;

    always @ (posedge clk) begin
        bar <= nextBar ^ a;
    end
endmodule

Questions:

What does this represent in hardware?

Why did we use ”always @ (posedge clk)”?
assign \( f = s[1] \oplus (s[0] \oplus a : b) : (s[0] \oplus c : d) \);

always @ (*) begin
    case (s)
        2'b00: \( f = d \);
        2'b01: \( f = c \);
        2'b10: \( f = b \);
        2'b'11: \( f = a \);
    endcase
end

always @ (*) begin
    if (s == 2'b00)
        \( f = d \);
    else if (s == 2'b01)
        \( f = c \);
    else if (s == 2'b10)
        \( f = b \);
    else //s == 2'b11
        \( f = a \);
end
assign f = a + b;
assign g = a - b;

wire [8:0] sum;
wire [7:0] a, b;
assign sum = {0, a} + {0, b};       // picks-up carry-out

wire [7:0] foo, bar
wire [7:0] a;
wire [3:0] b;
assign foo = a + {4'b0, b};         // what's different between
assign bar = a + {b, 4'b0};        // foo and bar?
assign isZero = (a == 0);
assign isGreater = (a > b); // is this signed?
assign isLTZ = (a < 0); // is this ever true?

To do signed comparison ALL signals used in the comparison must be additionally declared as ”signed”
  Ex: input wire signed [7:0] foo;

What other way do we know of for checking sign?
Verilog Tips - Constants

- `wire [7:0] foo = 127; // synthesis warning! Why?`
  - Missing number type (decimal, binary, hex)
  - Active will assume its decimal if not specified!
- **What other ways can we specify this?**
  - `wire [7:0] foo = 8’d127;`
  - `wire [7:0] foo = 8’b1111_1111;`
  - `wire [7:0] foo = 8’hff;`
  - `wire [7:0] foo = 8’hFF;`
Verilog Tips – Truncation

wire [7:0] a = 8'hAB;
wire b;
wire [7:0] c;

assign b = a;
assign c = a;

Questions:

What's wrong?

Will you get a synthesis warning?
Verilog Tips – reg vs. wire

wire f;
reg g, h;

assign f = a & b;

always @ (posedge clk)
    g <= a & b;

always @ (*)
    h = a & b;

Questions:

When do you declare something as a reg?

Are f and g the same? What about f and h?
Verilog Traps – Multiple always blocks

input wire a, b;
output reg f;

always @(posedge clk)
  if (a) f <= 1'b0;

always @(posedge clk)
  if (b) f <= 1'b1;

Questions:

What happens when a = 1 and b = 1?

How can we fix this?
One simple rule:

- If you want sequential logic, use `always @ (posedge clk) with <=`.
- If you want combinational logic, use `always @ (*) with =`.
Incomplete Sensitivity Lists

- What is a sensitivity list?
- Examples of problematic lists:
  - `always @ (a || b)`
  - `always @ (a)`
  - `f = a & b`
  - `always`
    - `f = a & b;`
- Tip: Use `always @ (*)` for combinational logic!
Latches!

always @ (posedge clk) begin
    if (a == 1)
        f <= 1;
    else if (a == 2)
        f <= 2;
    else if (a == 3)
        f <= 3;
end

Implicity this adds:
    else
        f <= f;

But we're okay...

always @ (*) begin
    if (a == 1)
        f = 1;
    else if (a == 2)
        f = 2;
    else if (a == 3)
        f = 3;
end

Implicity this adds:
    else
        f = f;

This is memory, but in a non-sequential circuit!
Displaying Stuff

- $display( ) is equivalent to C's printf( )
  - Same format strings
    - %d for a decimal
    - %h for hex
  - Ex: $display("%d in hex is: %h", foo, foo);
- For something which is assigned to with the non-blocking assignment operator (<=) you may want to use $strobe( )
X's

- X's are for undefined values
- Pins that are not connected will be X's. Often, 32'hxxxxxxxxxf4 indicates that you forgot to specify the bus's full width (Active-HDL defaults to 8-bit-wide buse)
  - 1'b1 & 1'bX ==> 1'bX
  - 1'b1 + 1'bX ==> 1'bX
Z's

- More than the things you won't be catching as much of at night, Z's are primarily for bus sharing.
- You don't need them in 378
- \( a \leq 1'bZ; \ b \leq 1'bZ \)
- \( a \leq 2'b0; \ b \leq 1'b1; \)
  - \( a \) will be 00 and \( b \) will be 1 in this case
- Sometimes Z's turn into X's!
  - \( 1'b1 \ & \ 1'bZ \Rightarrow 1'bX \)
  - \( 1'b1 \ + \ 1'bZ \Rightarrow 1'bX \)
Initial Values

- Synthesis doesn't always properly initialize wires/buses
- You can use an initial block but it's better design to have a reset input and reset logic to properly initialize things
- Initial block example:
  ```verilog
  Initial begin
    foo = 1'b1;
  end
  ```
Other

- We use Verilog 2001, your green sheet is in System Verilog. There are some syntactic differences:
  - Sign extension: foo = {16{bar[15]}, bar}; // S Verilog
    foo = {{16{bar[15]}}, bar}; // our Verilog

- Active-HDL uses a default bus width of 8 bits! Most of the buses in the lab need to be 32 bits wide!
  - Specify in the bus's declaration. Ex: wire [31:0] short;

- Give all of your buses names! This will alleviate many problems & makes debugging easier!
Verilog ⇐⇒ HW
Thanks for your attention!

For today: Make sure you can get into 003 and log-in.

Questions?