Design Discussion

- How would you start building a house?
  - Grab hammer, nails, wood... get started!? 
  - How about a tool shed?

- “Puzzles” vs Design

- Properties of good designs?
  - Art of design...

- Top down vs bottom up

Pre-Design

- What is the first step?

P37X Processor Design Process

- How would you approach designing a P37X processor?
  - What needs to happen?
Logic Design

- Logic design style
  - Suggestions? Goals?

- When to make a module or use hierarchy?

Testing & Debugging

- How?

Design Tuning

- How do you meet your requirements?
  - Performance, power, cost, etc.

- Tips
  - Make it slow & correct, then fast & correct
  - Change one thing at a time: either function or performance
    - Re-test after each change
  - Need to measure results of performance optimization

Final Two Projects

- Lab 2 - Implement Non-pipelined Processor
  - Worksheet - ~1 week
    - Note: use colored pen or pencil
  - Simulation demo - ~1 week
  - Hardware demo - ~1 week

- Lab 3 - Design & Implement Pipelined Processor
  - Much less guidance

- Both labs done in groups

- Today: Talk about “design”, P37x processor design
P37X Non-Pipelined Datapath

- Non-pipelined
  - Think of it as a single-cycle design

- Instruction memory and data memory
  - Actually a single two-ported memory
  - One read port, one read/write port

- ALU from Lab 1

- Control
  - 11 muxes, two write-enable signals
  - Fully determined by 4-bit opcode and 1-bit branch outcome

Verilog for the Datapath

- Use fully-structural Verilog
  - **Give all muxes and wires GOOD names**
  - Instantiate modules, muxes, wires to connect them
    - Your ALU
      - You can now use +, * to use Xilinx-optimized blocks
    - Memory
      - Provide by us (includes device registers, video interface)
    - A register file (8 registers, 16-bits each)
      - Use “register” module we provide
    - Controller
    - Branch Logic
    - Other
      - 16-bit PC register
      - Lots of Muxes
Verilog for Controller

- Controller
  - Inputs: 4-bit opcode, 1-bit branch outcome
  - Outputs: all mux and write enable signals

- Part 1: decode instructions
  - wire is_Arithmetic_Op = (opcode == 4'b0000);
  - wire is_ST = (opcode == 4'b1111);

- Part 2: set control signals
  - assign mem_we = (is_STR | is_ST);
  - Can use negation: assign sig = ~(... | ... | ...);

- Only ~40 lines of code!

Memory Module

- Processor storage
  - 2^6 location, each 16-bits
  - Used “Block RAM” on the FPGAs

- Memory mapped I/O
  - Memory mapped display (much like LC-3)
    - Only difference: 128x120 (rather than 128x124)
  - Timer registers
  - Keyboard registers
  - Read switches
  - Set LEDs
  - Set 7-segment display
  - Like “register”, memory specified using behavioral Verilog

New “Register” Module

```verilog
module register(out, in, we, gwe, rst, clk);
    parameter a = 1;
    parameter reset_value = 0;
    output [n-1:0] out;
    input [n-1:0] in;
    input    clk, we, gwe, rst;
    reg [n-1:0] state;
    assign #(1) out = state;
    always @(posedge clk)
        begin
            if (rst)
                state = reset_value;
            else if (gwe & we)
                state = in;
        end
endmodule
```
Next Steps...

- Assignment description available soon
- Work on the datapath worksheet