CSE141L: Building a microprocessor

Hung-Wei Tseng
You will design and implement a microprocessor!
Goals

- Practice what you will learn in CSE141
- Extend what you will learn in CSE141
  - Understand deeply how a processor works
  - See architecture play itself out in a real design
- Learn Verilog
- Get experience working on a large-scale project
- Have fun~~~
Course content

• You will implement a pipeline MIPS processor in Verilog using 5 weeks
  • It will be able to run simple but real programs compiled using gcc
  • It should be able to do simple I/O
• Make it your own processor
  • We will give you some code pieces
  • You have design the rest
  • We will give you the specifications for some others
  • You will invent, design, and implement some of your own
Course format

• Five labs
  • Due on every Friday
• Lectures — please check
  • Verilog coding
  • Discussing current or upcoming labs
  • Like group office hours
• No final exam, but have a short celebration on 9/7 11a
  • Pizza
  • Prize
  • Performance
Warning!

- The course is a lot of work
  - Don’t let the 2 units fool you
- Don’t fall behind
  - The labs build on each other
  - Hard to catch if you fall behind
- Don’t wait till the last minute
  - It’s called hardware for a good reason
  - The tools are complicated, buggy in some sense...
  - Your code will be buggy, too...
Lab 1: Familiar with the tools

• Two tutorials
  • Building projects in Quartus
  • Entering and compiling Verilog
  • Simulation using ModelSim
  • Measuring the performance of your design

• Start now!

• Due: this Friday — 8/9
Lab 2: Datapath elements

- Implementing the datapath elements required for a subset of MIPS instructions
- We will give you the design and some other key components
- You will implement the design
- Due next Friday — 8/16
Lab 3: Lights of life...

- Add control path to Lab 2
- Test your simple processor
- Execute simple programs
- Due on 8/23
Lab 4: It lives!

• Add missing pieces of MIPS
• You know how to have a working processor!
• Due 8/30
Lab 5: Let it live better!

- Pipeline your processor
- Measure the performance
- Due 9/6
Lab 6: Make it awesome!

- Optional
- Due 9/6 as well
- You can implement any other fancy features in your processor to get an A+
  - Cache
  - Dual-core
  - Branch predictor
  - Speculation
  - Dynamic scheduling
  - and etc...
Lab space and software

• We will use Altera tools for development (Quartus II)
  • Verilog editing
  • Design analysis
• We will use ModemSim for simulation
  • Simulation
  • Debugging
• Tools are huge pains
• The labs in the CSE basement have the tools installed
  • CSE B250-B270
• They are also available for free
Do the work

• Lab 1 should be done independently
• Lab 2-5 should be in group of 2
  • Choose your group carefully
  • You cannot merge groups
  • Splitting up is allowed (but not encouraged)
  • Schedule an interview with TA (or Hung-Wei) before Friday 5pm every week when you’re done with the lab assignment
  • We will be interviewing with the whole group
• No written report
# Grading

<table>
<thead>
<tr>
<th>Achievement</th>
<th>Final Grade</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden Level (Lab 6)</td>
<td>A+</td>
<td>9/6/2019</td>
</tr>
<tr>
<td>Lab 5</td>
<td>A</td>
<td>9/6/2019</td>
</tr>
<tr>
<td></td>
<td>A-</td>
<td></td>
</tr>
<tr>
<td>Lab 4</td>
<td>B+</td>
<td>8/30/2019</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Lab 3</td>
<td>B-</td>
<td>8/23/2019</td>
</tr>
<tr>
<td></td>
<td>C+</td>
<td></td>
</tr>
<tr>
<td>Lab 2</td>
<td>C</td>
<td>8/16/2019</td>
</tr>
<tr>
<td></td>
<td>C-</td>
<td></td>
</tr>
<tr>
<td>Lab 1</td>
<td>D</td>
<td>8/9/2019</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Instructor

• Instructor: Hung-Wei Tseng
  • Lectures: http://goo.gl/VSL97g
  • Lab hours: http://goo.gl/VSL97g
    @ CSE B250-B270 or by appointment

• Check the calendar on our website http://goo.gl/VSL97g
Teaching Assistants

• Harish Prasanth
  • Lab hours: TuTh 1p - 4p@ EBU3B B250-B270
  • E-mail: hgajendr@eng.ucsd.edu

• Cameron Foster
  • Lab hours: ??? @ EBU3B B250-B270
  • E-mail: cafoster@ucsd.edu

• Yunan (Andrew) Zhang
  • Lab hours: FSu 3p - 54p@ EBU3B B250-B270
  • E-mail: yuz057@ucsd.edu
Course resources

- Course webpage:
  http://cseweb.ucsd.edu/classes/su19_2/cse141L-a/

- TritonEd:
  http://tritoned.ucsd.edu
  we use tritoned to turn in reports, record grades.

- Discussion board:
  - Search before ask
  - https://piazza.com/class/jxfmqa8wajk20v
Verilog

Hung-Wei Tseng
Verilog

• Verilog is a hardware description language (HDL).
• In this class, we use Verilog to implement and verify your processor.
• C/Java like syntax
Data type in Verilog

- Bit vector is the only data type in Verilog
- A bit can be one of the following
  - 0: logic zero
  - 1: logic one
  - X: unknown logic value, don’t care
  - Z: high impedance, floating
- Bit vectors expressed in multiple ways
  - binary: 4'b11_10 ( _ is just for readability)
  - hex: 16'h034f
  - decimal: 32'd270
Operators

- Arithmetic: + - * / % ** (don’t use the last three)
- Logic: ! && ||
- Relational: > < >= <=
- Equality: == != === !===
- Bitwise: ~ & | ^ ^~
- Reduction: & ~& l ^ ^~
- Shift: >> << >>> <<<
- Concatenation: { }
- Conditional: ? :

High-level view of hardware

wires

1-bit Module Adder
1-bit Module Adder
1-bit Module Adder
1-bit Module Adder

4-bit Carry Look Ahead

A3 B3 A2 B2 A1 B1 A0 B0
S3 S2 S1 S0 C0

p3 g3 C3 p2 g2 C2 p1 g1 C1 p0 g0 PG GG
Wire to connect things together!

- wire is used to denote a hardware net
  - single wire
    ```
    wire my_wire;
    ```
  - array of wires
    ```
    wire[7:0] my_wire;
    ```

- For procedural assignments, we will use reg
  - again, can either have a single reg or an array
    ```
    reg[7:0] result; // 8-bit reg
    ```
  - reg is not necessarily a hardware register
  - you may consider it as a variable in C
A Verilog module has a name and a port list
  • ports: must have a direction (input, output, inout) and a bitwidth

Think about an 1-bit adder
  • input: 1-bit * 3
  • output 1-bit * 1 and 1-bit * 1

```verilog
class FA(
    input a,
    input b,
    input cin,
    output cout,
    output sum
);
assign sum = a\^b\^cin;
assign cout = (a\&b) | (a\&cin) | (b\&cin);
endmodule
```

Adapted from Arvind & Asanovic’s MIT 6.375 lecture
Always block

- Executes when the condition in the sensitivity list occurs

```verilog
always@(posedge clk)
begin
...
...
end

module FA(
input a,
input b,
input cin,
output cout,
output sum );

reg s, cout
always@(a or b or cin)
begin
  sum = a^b^cin;
  cout = (a&b) | (a&cin) | (b&cin);
end
endmodule
```
Blocking and non-blocking

• Inside an always block, = is a blocking assignment
  • assignment happens immediately and affect the subsequent statements in the always block
• <= is a non-blocking assignment
  • All the assignments happens at the end of the block

Initially, \( a = 2, b = 3 \)

```verilog
reg a[3:0];
reg b[3:0];
reg c[3:0];
always @(posedge clock)
begin
  a <= b;
  c <= a;
and
Afterwards: a = 3 and c = 2
```

```verilog
reg a[3:0];
reg b[3:0];
reg c[3:0];
always @(*)
begin
  a = b;
  c = a;
and
Afterwards: a = 3 and c = 3
```

sequential logic

combinational logic
Initial block

- Executes only once in beginning of the code

    initial
    begin
    ...
    ...
    end
A verilog module can instantiate other modules

```verilog
module adder(input [3:0] A,
              input [3:0] B,
              output carry,
              output [3:0] sum);

wire c0, c1, c2
FA fa0(A[0],B[0],cin,c0,sum[0]); // implicit binding
FA fa1(.a(A[1]), .b(B[1]), .cin(c0), .sum(sum[1]), .cout(c1)); // explicit binding
FA fa2(A[2],B[2],c1,c2,sum[2]);
FA fa3(A[3],B[3],c2,cout,sum[3]);
endmodule
```

Adapted from Arvind & Asanovic’s MIT 6.375 lecture
testing modules

```
`timescale 1ns/1ns // Add this to the top of your file to set time scale
module testbench();
reg [3:0] A, B;
reg C0;
wire [3:0] S;
wire C4;
adder uut (.B(B), .A(A), .sum(S), .cout(C4)); // instantiate adder

initial
begin
A = 4’d0; B = 4’d0; C0 = 1’b0;
#50 A = 4’d3; B = 4’d4; // wait 50 ns before next assignment
#50 A = 4’b0001; B = 4’b0010; // don’t use #n outside of testbenches
end

endmodule
```
Resources

• Check out MIT’s 6.375 course webpage  
  http://csg.csail.mit.edu/6.375/  
  • thanks to Asanovic & Arvind for slides

• Tips for using Altera tools  
  https://sites.google.com/a/eng.ucsd.edu/using-the-altera-tools/  
  • Thanks to Steven Swanson and other CSE141L winter 2012 staffs
Q & A