EECS 151/251A
SP2022 Discussion 1

GSI: Yikuan Chen, Dima Nikiforov

Slides modified from Alisha Menon’s and Sean Huang’s slides
My job:

- To help you get the most out of this class!
- Discussion sections
  - Cory 540 every Friday
  - Review this week
  - Answer questions
  - Example problems
- Office hours
  - Yikuan’s: Thursday 10-11am
  - Office hour of each staff listed on course website:
    - https://inst.eecs.berkeley.edu/~eeecs151/sp22/
- Piazza
  - Please contact GSIs here instead of email so your question could bring value to all students
  - GSIs will try to respond within 24 hours
How to success

- Put the most effort into labs/project
  They make you a great engineer, not just a good IC student

- Understand abstraction leverage it for productive design
  Stay in circuit design: Apple shows you how desperate they are!

- Choose final project partners wisely. It Takes Two to make a team
What else

- Homework stresses understanding
  All questions will be graded on correctness, but open-ended questions have higher weighting

- Stay up-to-date on industry & research trends!
  IEEE Computing & Semiconductors, EE Times, Semiconductor Engineering, TechInsights
  IEEE CAS & Computer Societies, ACM, etc.
Agenda

- Administrative
- ASIC vs FPGA
- Verilog intro
Administrative

- Homework 1 will be posted this week due on Friday.
- Labs are in Cory 111/117 – There’s a Monday Lab 2-5pm, GSI Seah Kim.
  Western Entrance
ASIC vs FPGA
**ASIC**

- **Application-Specific Integrated Circuit**
- Optimized for the application
- Use standard cells, SRAM, custom analog circuits

2-input NAND gate

Apple’s A11 Bionic SoC

https://www.macworld.com/article/3275567/iphone-ipad/apple-7nm-a12-chip-iphone.html
FPGA

- Field-Programmable Gate Array.
- Can be programmed on the fly, hence Field-Programmable Look-up Table (LUT, /\(\lambda\)\) based, pre-placed
- e.g. 6 bit in, 1 bit out

Typical FPGA Structure

Xilinx Virtex Family

EECS 151/251A Discussion 1
# ASIC vs FPGA

<table>
<thead>
<tr>
<th></th>
<th>ASIC</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>higher</td>
<td>med</td>
</tr>
<tr>
<td><strong>Design cost</strong></td>
<td>expensive (very!)</td>
<td>cheap</td>
</tr>
<tr>
<td><strong>Per-unit cost</strong></td>
<td>lower</td>
<td>higher</td>
</tr>
<tr>
<td><strong>Design time</strong></td>
<td>long</td>
<td>short  ~ minute</td>
</tr>
<tr>
<td><strong>Custom blocks</strong></td>
<td>high customizibility</td>
<td>low</td>
</tr>
<tr>
<td><strong>Job perspective</strong></td>
<td>Big Tech companies</td>
<td>Research, non-EE field</td>
</tr>
</tbody>
</table>
Hardware description language (HDL)

- Describes a digital system
- Tools can synthesize the code to emulate a circuit
- Keep reminding yourself: I’m describing hardware, not writing a program
- Always sketch out your circuit, even if just at a high level (modules, ports, connections, aka boxes, texts and lines)
- Learn by hands-on practice!
Keep in mind

- Not a programming language!
  - Only the syntax is based on C for familiarity
  - Circuits are not programs and follow different rules
  - Think about it from a circuit perspective
- Not "programming a circuit"
- Writing a description
Different sources might prefer a different flavor of Verilog syntax & formatting, we recommend that you keep consistent with the lectures.
**Verilog Basics**

- **Examples of differences**
- **Combinational logic**
  - All combinational blocks are always running in parallel
  - Output updates immediately* with input
  - Just because something is assigned at a later line doesn’t mean it runs later!
- **Sequential logic**
  - Many registers can update on same clock edge
  - Usually drive combinational blocks
  - Need to be careful not to have conflicts

---

*Immediate only in RTL simulation. In gate-level simulation, there will be some gate delay before the output updates*
Basic Verilog module

module module1_name (a, b, o);
  input [1:0] a;
  input b;
  output reg o;

  // we call it an “always block”
  always @ (*) begin
    if(condition) begin
      o = expression1 of a and b;
    end else begin
      o = expression2 of a and b;
    end
  end
endmodule

Two Equivalent Modules
Basic Verilog module

**Instantiating a module in another module**

```verilog
module sub_module_name (    
    input [1:0] a, 
    input b, 
    output o 
);

assign o = (condition)?expression1 of a and b : expression 2 of a and b;

endmodule

module big_module_name (    
    input [1:0] input, 
    output output 
);

wire b = 1’b1; //local wire with fixed value

sub_module_name my_sub(    
    .a(input), 
    .b(b), 
    .o(output) 
);

endmodule
```
• **Structural Verilog vs Behavioral Verilog**

  • **Structural Verilog**
    - Directly describe the physical relationship & connection in code
    - Typically combinational logic circuits build out of basic gates and module instances
  
  • **Behavioral Verilog**
    - Describe the action of the module
    - Use assign statements for continuous assignment
    - Use always@(some_condition) blocks to describe how a circuit behaves under certain conditions
Verilog modules

- **Black box** design unit with a specific purpose
- Designer specifies **inputs, outputs, and behavior**
- Example: 2-input multiplexer

```verilog
module mux(
    input a,
    input b,
    input sel,
    output o
);
assign o = sel? a : b;
endmodule
```
Wire vs. Register

- wire

- Register (reg)

```verilog
wire b = a;
assign b = a;
always b = a;
if (sel) b = a;
else b = ...
reg b;
```

```
D Q
A B
clk

always @ (posedge clk) begin
end

Select

A

... B

OR

reg b;
```
Examples (with common syntax errors)

```verilog
module mux2 (in, select, out, out_b);
assign out = (select==1) ? in[1] : in[0];
endmodule
```

```verilog
module mux2_behav (in, out, out_b);
input [0:1] in;
input select;
output reg out;
output reg out_b;
always @(in)
begin
  if (select == 1)
  begin
    out = in[1] ;
    out_b = ~in[1] ;
  end
  else
  begin
    out = in[0] ;
    out_b = ~in[0] ;
  end
endmodule
```
module mux2 (in, select, out, out_b);
input [0:1] in;
input select;
output reg out;
output reg out_b;
always @(*) begin
    if (select == 1) begin
        out = in[1]
        out_b = ~in[1]
    end else begin
        out = in[0]
        out_b = ~in[0]
    end
end
endmodule

Examples (error corrected)

Example 1
module mux2 (in, select, out, out_b);
input [1:0] in;
input select;
output reg out;
output reg out_b;
always @(*) begin
    if (select == 1) begin
        out = in[1]
        out_b = ~in[1]
    end else begin
        out = in[0]
        out_b = ~in[0]
    end
end
endmodule

Example 2
assign out = (select==1)? in[1] : in[0];

Example 3
module mux2 (in, select, out, out_b);
input [1:0] in;
input select;
output reg out;
output reg out_b;
always @(*) begin
    if (select == 1) begin
        out = in[1]
        out_b = ~in[1]
    end else begin
        out = in[0]
        out_b = ~in[0]
    end
end