Hi

Arya Reais-Parsi
aryap@berkeley.edu

- 1st year Berkeley PhD student
- From Iran, New Zealand and Australia (so far)
- Received BE in 2010 from Victoria University of Wellington (New Zealand)
- Worked for Google in Sydney for 6 years
- Super fun guy

  aryap@berkeley.edu
Happy Australia Day!

- Public holiday
- Actually controversial
- Here’s 9 News using the wrong flag →
Reminder

- Homework 1 due tonight at 11:59 pm
- Homework 2 posted, due next Friday at 11:59 pm
- Not too late to get help for homework 1
Discussions

- Enrich class material
  - Provide some review
  - Cover additional examples
- Will feed off questions on Piazza
- Are here to help
- Will address topics you want to cover
Agenda

- ASIC vs FPGA
- Verilog
ASIC vs FPGA

Use standard cells, SRAM, custom analog circuits

Use LUTs, block RAM, on-die IP cores
### Bitcoin mining platform progression

1. Software
2. GPU
3. FPGA
4. ASIC

**Why?**

[Image: https://www.bitcoinmining.com/images/usb-bitcoin-miner-gekko-science.jpg]
## ASIC vs FPGA

<table>
<thead>
<tr>
<th>ASIC</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use standard cells, SRAM, custom analog</td>
<td>Use LUTs, block RAM, on-die IP cores</td>
</tr>
<tr>
<td>Maximum performance ☺</td>
<td>Lower performance ☹</td>
</tr>
<tr>
<td>High design cost (NRE*) ☹</td>
<td>Low design cost ☺</td>
</tr>
<tr>
<td>Low per-unit cost ☺</td>
<td>High per-unit cost ☹</td>
</tr>
<tr>
<td>Year(s) to design and manufacture ☹</td>
<td>Minutes to reprogram ☺</td>
</tr>
<tr>
<td>Access to custom analog circuits ☹</td>
<td>Restricted to whatever is on the FPGA ☹</td>
</tr>
<tr>
<td>Jobs at chip companies and big tech players (Google, FB, Apple)</td>
<td>Wide range of jobs both in EE fields and outside</td>
</tr>
</tbody>
</table>

*NRE: Non-Recurring Engineering cost*
Verilog

- One of several Hardware Design Languages (HDLs)
- Not a traditional programming language
- Have to learn mindset from scratch

- Don’t think about it like a traditional programming language
- Stop thinking about it like a traditional programming language
- Stop it
- This isn’t C/Java/Python/Ruby/JavaScript/...
- You’re describing a circuit
  - not a sequence of instructions to execute
Verilog

Two general ways of description:

1. Structural Verilog
   ● Describe the schematic in text format

2. Behavioral Verilog
   ● Describe the desired functionality
   ● Two assignment types:
     ○ Continuous assignments (assign …)
     ○ Non-continuous assignments (always @(…))
Sequential Logic

**Combinational**

```verilog
module comb(input a, b, sel, output reg out);
    always @(*) begin
        if (sel) out = b;
        else out = a;
    end
endmodule
```

**Sequential**

```verilog
module seq(input a, b, sel, clk, output reg out);
    always @(posedge clk) begin
        if (sel) out <= b;
        else out <= a;
    end
endmodule
```

**event driven**
Blocking vs. non-blocking assignment

Blocking (=): evaluation and assignment are immediate; sequence matters. Value of expressions at that point in the description matters.

```verbatim
always @(*) begin
  x = a | b;         // 1. evaluate a|b, assign result to x
  y = a ^ b ^ c;     // 2. evaluate a^b^c, assign result to y
  z = b & ~c;        // 3. evaluate b&(~c), assign result to z
end
```

Non-blocking (<=): all assignments deferred to end of simulation time step after all right-hand expressions have been evaluated.

```verbatim
always @(*) begin
  x <= a | b;        // 1. evaluate a|b, but defer assignment to x
  y <= a ^ b ^ c;    // 2. evaluate a^b^c, but defer assignment to y
  z <= b & ~c;       // 3. evaluate b&(~c), but defer assignment to z
  // 4. end of time step: assign new values to x, y and z
end
```
Blocking vs. non-blocking example

Guideline: use non-blocking assignments for sequential always blocks
References

- Ali Moin, EECS 151 Spring 2017 Discussion 1