A Brief Intro to Verilog
Brought to you by: Sat Garcia

Meet your 141(L) TA

- Sat Garcia
  - sat@cs.ucsd.edu
  - Nth Year Ph.D. Student
  - Office Hours: (Tentative)
    - Place: TBA (somewhere in CSE basement)
    - Wednesday: 5-6pm
    - Friday: 2-3pm (right after lecture)
What is Verilog?

- Verilog is:
  - A hardware design language (HDL)
  - Tool for specifying hardware circuits
    - Also for simulating circuits
  - Syntactically, a lot like C or Java
  - An alternative to VHDL (and more widely used)
  - What you'll be using in 141L
  - Awesome

* If you are totally into hardware design languages

Verilog in the Design Process

Adapted from Arvind & Asanovic's MIT 6.375 lecture
Styles of Verilog

- RTL
  - Specific description on how module operates
    - Adds, shifts, etc.

- Structural
  - Describe how modules connect together
    - Instantiate modules written in RTL and wire them together

- Behavioral
  - High level description of what module is doing
    - Looks a lot like C++ or Java
  - Not always synthesizable
    - If it is, it is probably inefficient

A simple example (comb. circuit)

- Let's design a 1 bit full adder (RTL style)

```verilog
module FA( input a, b, cin, output s, cout);
assign s = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

- Ok, but what if we want more than 1 bit FA?

*** Note: red means new concept, blue and green are just pretty colors :-p

Adapted from Arvind & Asanovic's MIT 6.375 lecture
A 4-bit Full Adder

- We can use 1 bit FA to build a 4 bit full adder
  - Structural style

```
module 4bitFA( input [3:0] A, B, input cin,
               output [3:0] S, output cout);
  wire c0, c1, c2;
  FA fa0(A[0],B[0],cin,S[0],c0); // implicit binding
  FA fa1(.a(A[1]), .b(B[1]), .cin(c0), .s(S[1]), .cout(c1)); // explicit binding (use this!)
  FA fa2(A[2],B[2],c1,S[2],c2);
  FA fa3(A[3],B[3],c2,S[3],cout);
endmodule
```

Adapted from Arvind & Asanovic’s MIT 6.375 lecture

A simple D flip flop (seq. circuit)

- For sequential circuits, use always blocks
- Always blocks (and assign) are executed in parallel!

```
module DFF( input clk, d,
            output q, q_bar);
  reg q, q_bar;
  always @(posedge clk) // triggered on the rising edge of the clock
  begin
    q <= d; // non-blocking assignment (LHS not updated until later)
    q_bar <= ~d;
    /* q_bar <= ~q will not function correctly! Why not? */
  end
endmodule
```

Adapted from Arvind & Asanovic’s MIT 6.375 lecture