Verilog
In lab we are currently drawing the devices and gates when implementing logic on the FPGA. We call it drawing “schematic capture”. As you’ve likely noticed, it is generally a lot easier to write the logic equations than draw the gates. As such people don’t generally use schematic capture. Rather they use a text-based scheme called a “hardware description language” (HDL). Not only does it make it easier to enter logic equations, but later we’ll see that we can specify some pretty high-level notions and ask the tools to design the hardware.

Basics – the assign statement
The most basic statement is the assign statement. Here we can specify gates and the like with simple symbols. & is AND, | is OR, ^ is XOR and ~ is NOT. For example:

```
assign a=(y|~x)^z;
```

Consider the following gates. Write a logic equation for each output.

**Figure 1a:** Half adder

**Figure 1b:** Full adder

Half adder:

- 
- 

Full adder:

- 
- 

**Basics – defining a module**

While our fundamental building block are gates, we often need something more complex (for example MSI devices). In prelab2 you’ll be asked to build “boxes” using schematic capture. In Verilog we often want to do the same thing. So we define a module in a way that is reminiscent of writing a function in C.

```verilog
module add_half (a, b, s, cout);
  input a, b;
  output s, cout;
  wire s, cout;
  assign s = a ^ b;
  assign cout = a & b;
endmodule
```

The half adder can then be used, along with a full adder, to make a 2-bit adder.

```verilog
module add_2bit (a, b, s, cout);

  input [1:0] a, b;  // Both a and b are 2 bit inputs
  output [1:0] s;   // s[1] = MSB of s, s[0] = LSB of s
  output cout;
  wire [1:0] s;
  wire cout;
  wire c0;        // intermediate carry between adders

  add_half a1(a[0], b[0], s[0], c0);
  add_full a2(a[1], b[1], c0, s[1], cout);
endmodule
```

- Write the code for the full adder.
- Modify the above add_2bit to make a 4-bit adder.