CMU 18-447: Introduction to Computer Architecture

Handout: Moving from System Verilog to Verilog

There are no always_comb, always_ff or always_latch blocks in Verilog: a simple always statement needs to be used in place of all of these.

Combinational logic is specified as:

```
reg [7:0] b, c,
sum, diff;

always @(*) begin

     sum = b+c;

     diff = b-c;

end
```

The always @(*) statement is the same as the always_comb statement: it tells the parser that you're intending to make a combinational model, updates the value of the outputs whenever any of the inputs changes. However, if you forget to assign a value to a variable along a path, the complier assumes it to be a latch: thus, if you intend to simulate combinational logic, it is important to **insure that variables are assigned a value along every path**.

Flip flops are specified in a manner similar to System Verilog, except for using the always statement instead of always_ff:

```
reg [7:0] q,d

always @(posedge clk)
begin

     if(~rst)

          q <= 0;

     else

          q <= d;

end
```

Verilog does not have logic variables. Instead, **reg** or **wire** variables are used.

reg variables actually hold state. They are the variables on the left hand side of a statement in an always or initial block.

On the other hand, wire variables are used as interconnect between two points. They are used to connect modules together and in assign statements.

```
reg [7:0] b, c, sum;

wire [7:0] diff, out

always @(*) begin

     sum = b+c;

end

assign diff = b – c;

calc c1(out, sum,
diff);

testbench t1(out);
```

```
module calc(output [7:0] out,
            input  [7:0] sum,
                         diff)

…

module testbench(input [7:0]
                       out)

…
```

Quick tip: In most cases, any net assigned to in an always or initial block is a reg. Any other net is a wire. Both kinds of variables can be used in conjunction on the right hand side of an assignment. For more details, refer to:

http://inst.eecs.berkeley.edu/~cs150/Documents/Nets.pdf

Quick Notes:

- When instantiating modules the (.*) implicit port connection method is not legal in Verilog. You need to name all the wires. For instance, in the example above,`calc c1(.*)` would not have been valid.
- There are **no enums** in Verilog: use **`define** statements instead.
- No interfaces, no assertions, no classes.