Verilog Cheat Sheet (version 0.8)
for CS552 - Spring 2013

I) Literals

<size >'<base><number>

All of the following are the same:

```
8'b10101111
8'hAF
{4'hA,4'b1111}
{4'hA,{4{1'b1}}}
```

II) Module declaration

//Rule: Use separate file for each module (Filename should be module_name.v):

```
module module_name (A, B, C);
    input A; // Inputs to the module
    input [2:0] B;  //Convention: one per line
    output C;  // Output of the module
    ...
endmodule
```

III)Module instantiation

//In file outer_module.v

```
module outer_module (in1, in2, out);
    input in1;
    input [2:0] in2;
    output out;
    module_name md (.A(in1), .B(in2), .C(out));
    ...
endmodule
```

//Rule: Use port mapping. The port A of module module_name is connected to wire in1 in module outer_module.

IV) wire vs reg

a) Rules for usage within module

- LHS of "continuous assign" should be a wire See also VI
- LHS of "procedural assign" should be reg.

b) Rules for ports

- Ports are by default of type wire (you can override this by declaring it as reg)
- Input ports cannot be declared as reg (Input ports are always wires)
- Output ports can be either reg or wire

c) Rules for connecting ports while instantiating

- When instantiating a module, an output port should be connected to a wire (cannot be connected to reg)
- When instantiating a module, an input port can be connected to either a reg or a wire

V) Sequential logic

- Rule: To create a flip-flop, instantiate the provided dff.v module.
- Rule: Do not code sequential logic in any other way.

```
dff d0 (
    .q(flop_out),
    .d(flop_in),
    .rst(reset),
    .clk(clock)
);
```

VI) Combinational logic

a) Instantiate the provided logic gates (course webpage)

b) assign statement **(Continuous assign)**

```
wire result;
assign result = s ? A : B;
```

//result – must always be wire (refer above)

//s, A, B – can be wire or reg

c) **Procedural assign** (blocking = or non-blocking <=)

```
reg result;
reg err;
always @(s or A or B)
begin
    casex(s)
       1'b1:
          begin
             result = A;
             err = 1'b0;
          end
       1'b0:
          begin
             result = B;
             err = 1'b0;
          end
       default:
          begin
             result = 1'bx;
             err = 1'b1;
          end
    endcase
end
```

//result, err – must always be reg (refer above).

//s, A, B – can be wire or reg

VII) parameter

- Parameterized module definition (in register.v)

```
module register(out, in, wr_en, clk, rst);
    parameter WIDTH = 1;
    output [WIDTH-1:0] out;
    input [WIDTH-1:0] in;
    input wr_en;
    input clk;
    input rst;
    dff_en bits[WIDTH-1:0] (
      .q(out),
      .d(in),
      .en(wr_en),
      .clk(clk),
      .rst(rst)
    );//dff_en should instantiate the provided dff.v module
endmodule
```

- Instantiating a parameterized module: with default value of the parameter

```
register r0 (.....);
```

- Instantiating a parameterized module: override the default value

```
register #(32) r1 (.....);
register #(1) r2 (.....);
```

VIII)Array instantiation

The dff instantiation in the example above is an array instantiation. It instantiates many flops with names bits[0], bits[1], ... bits[n]. Note how the wire 'out' is split across many different instances. Also, the wire 'wr_en' is connected to multiple ports (one each of each instantiation).

IX) define

- Keep defines in a separate file (modname_config.v):

```
`define LAST_VALUE 4'b1010
`define WIDTH 4
```


Developed by: Vinay Gangadhar, Cherin Joseph & Prof. Karthikeyan Sankaralingam

Verilog Cheat Sheet (version 0.8)
for CS552 - Spring 2013

- Include the above file in verilog file (modname.v):

```
`include "modname_config.v"
module modname(.........);
    .......
    wire [`WIDTH-1:0] carry;
    assign wire = ~carry & `LAST_VALUE;
    .......
endmodule
```

X) Allowed keywords

assign, module, endmodule, input, output, wire, define, parameter

XI) Keywords allowed with stipulations

case, casex, reg, always, begin, end

a) case, casex:

- Have items for all possible combinations. Use default and err should be asserted in default.
- All outputs of case statement should be assigned in all case items.
- All nets used in RHS of all assigns within case statement and all nets used as the compare value in case statement should be specified in the sensitivity list.

b) reg:

- Can only be used to specify outputs of case/casex statement.

c) always, begin, end:

- Can only be used to introduce case/casex statement.

XII) Allowed Operators :

*In list below, shift operators should have the second argument as constant. (x<<4 is allowed whereas x<<y is not allowed)

~m	Inversion
m & n	Bitwise AND
m|n	Bitwise OR
m^n	Bitwise XOR
m~^n	Bitwose XNOR
&m	ReductionAND
~&m	Reduction NAND
|m	Reduction OR
~|m	Reduction NOR
^m	Reduction NOR
~^m	Reduction XNOR
m==n	Equality
m!=n	Inequailty
m===n	Identity
m!==n	Not Identical
m << const	Shift left by const bits
m >> const	Shift right by const bits
condition ? m : n	Ternary
{m, n}	concatenation
{m {n}}	replicate n (m times)

XIII) Testing your design

a) Design file template to be provided for HW2-HW6 and for the project (in file foo.v);

```
module foo (in, out, clk, rst, err);
    ...
endmodule
```

b) _hier.v file to be provided for HW2-HW6 and for the project. (foo_hier.v):

```
module foo_hier ( in, out )
    ...
    clkrst c0( .clk(clk), .rst(rst), .err(err) );
    foo f0 ( .out(out), .in(in), .clk(clk),
      .rst(rst), .err(err) );
endmodule
```

c) The testbench _hier_bench.v file to be developed and submitted by the student (foo_hier_bench.v)

```
module foo_hier_bench;
    foo_hier f0 (....);
    ...
endmodule
```

XIV) Scripts

a) Verilog rules check script (Not foolproof)

```
vcheck-all.sh
```

b) Name convention check script:

```
name-convention-check
```

c) Command line verilog simulation script:

```
wsrun.pl foo_hier_bench *.v
wsrun.pl -wave foo_hier_bench *.v
```

d) Synthesis script:

```
synth.pl-cmd=synth -type=other -top=foo -opt=yes -
file=foo.v,foo_submodule1.v,foo_submodule2.v
```

XV) Submission rules

HW2-HW6 and project submissions require absolute compliance with guidelines. **Here is how you can do almost everything right, but still score ZERO:**

- By tampering with the provided template for foo.v and/or foo_hier.v
- By not submitting the provided testbench components foo_hier.v file and clkrst.v along with the other verilog files.
- By not submitting the other provided modules like dff.v, not1.v, nand3.v etc. which are required to compile your design.
- By submitting a tar file with a directory structure not matching the guidelines.

(eg, if you have a wrapper directory over hw1_1, hw1_2 and hw1_3)
(eg, if your verilog files are located within subdirectories inside hw1_1)
(eg, if you submit hw1_1, hw2_2, hw3_3 when you are asked to submit hw1_1, hw1_2 and hw1_3)

- By submitting .tar.gz or .zip when you are asked to submit .tar
- By not running vcheck on verilog files or by not checking the results after running the script.
- By not turning in .vcheck.out files for each .v file (except the testbench components and provided module.)
- By forgetting to click on the 'Submit' button in dropbox after clicking the 'Upload' button.

XVI) Modelsim waveform viewing/debugging cheats?

Course bonus points abound ☺

Developed by: Vinay Gangadhar, Cherin Joseph & Prof. Karthikeyan Sankaralingam

Verilog Cheat Sheet (version 0.8)
for CS552 - Spring 2013

Code example (dyser_stage.v)

```
`include "dyser_config.v"
// dff_rn is a d-flip-flop with negative reset
// dff_rne is a d-flip-flop with negative reset and enable
module stage(
    /* inputs */
    ready_in, valid_in, credit_in, data_in, clk, rst_n,
    /* outputs */
    credit_out, data_out, valid_out, ready_out, err
    );

   parameter                ID = 0;
   parameter                EDGE = 0;

   input                    ready_in;
   input                    valid_in;
   input                    credit_in;
   input [`DATA_WIDTH:0]    data_in;
   input                    clk;
   input                    rst_n;

   output                   credit_out;
   output [`DATA_WIDTH:0]   data_out;
   output                   valid_out;
   output                   ready_out;
   output                   err;

   // wires and reg

   reg                      credit_out;
   reg                      data_en;
   reg [`DATA_WIDTH:0]      data;
   reg                      valid;
   reg                      ready_out;
   reg                      state;

   parameter CN    = 1'b0;
   parameter NR    = 1'b1;

   // state + next state logic
   wire next_state;
   dff_rn  state_ff( .din(next_state), .q(state),
                     .rst_n(rst_n) );
   assign next_state = (state == CN) ? ready_in ? NR : CN :
                       /*state == NR*/ credit_in ? CN : NR;

   // output logic (Mealy)
   always @(state or credit_in or ready_in)
     //always @(*)
     case ({state,credit_in,ready_in})
       //for state CN
       3'b0_0_0:
         begin
            ready_out     = 1'b0;
            data_en       = 1'b0;
            credit_out    = 1'b1;
            err           = 1'b0;
         end
       3'b0_0_1:
         begin
            ready_out     = 1'b0;
            data_en       = 1'b1;
            credit_out    = 1'b1;
            err           = 1'b0;
         end
       3'b0_1_0:
         begin
            ready_out     = 1'b0;
            data_en       = 1'b0;
            credit_out    = 1'b1;
            err           = 1'b0;
         end
       3'b0_1_1:
         begin
            ready_out     = 1'b0;
            data_en       = 1'b1;
            credit_out    = 1'b1;
            err           = 1'b0;
         end
       3'b0_0_0:
         begin
            ready_out     = 1'b1;
            data_en       = 1'b0;
            credit_out    = 1'b0;
            err           = 1'b0;
         end
       3'b0_0_1:
         begin
            ready_out     = 1'b1;
            data_en       = 1'b0;
            credit_out    = 1'b0;
            err           = 1'b0;
         end
       3'b0_1_0:
         begin
            ready_out     = 1'b1;
            data_en       = 1'b0;
            credit_out    = 1'b0;
            err           = 1'b0;
         end
       3'b0_1_1:
         begin
            ready_out     = 1'b1;
            data_en       = 1'b0;
            credit_out    = 1'b0;
            err           = 1'b0;
         end
       default:
         begin
            //$display("ERROR time: %d", $time );
            ready_out     = 1'b0;
            data_en       = 1'b0;
            credit_out    = 1'b0;
            err           = 1'b1;
         end
     endcase

   // data and valid FF
   dff_rne_data_width data_ff( .din(data_in)
                               .q(data)
                               .en(data_en)
                               .rst_n(rst_n));

   dff_rne valid_ff( .din(valid_in)
                     .q(valid)
                     .en(data_en)
                     .rst_n(rst_n));

   assign data_out  = data;
   assign valid_out = valid;

endmodule
```


Developed by: Vinay Gangadhar, Cherin Joseph & Prof. Karthikeyan Sankaralingam