Using verilog

You need to be alert to (usually minor) changes that may be made to the assignment statement or to the guidelines after the assignment is first put up. Refresh this frame and re-read the assignment carefully before you make your final submission.

Verilog hardware description language (HDL)

Behaviour of hardware can be described using verilog or VHDL (another HDL). Here we focus on verilog only. Use of verilog is explained through examples of known circuits. Verilog specifications may be used by computer aided design tools to synthesise circuits which may be realised on suitable physical devices such a field programmable gate arrays (FPGAs) or even application specific integrated circuits (ASICs).

Structural specification of a full adder without delays

```
module FA_Struct(a, b, cin, cout, sum);
 input  a, b, cin;      // inputs
 output cout, sum;      // output
 wire   w1, w2, w3, w4; // internal nets

 xor U1  (w1, a, b);
 xor U2  (sum, w1, cin);
 and U3  (w2, a, b);
 and U4  (w3, a, cin);
 and U5  (w4, b, cin);
 or  U6  (cout, w2, w3, w4);
endmodule
```

In the above description the following may be noted:

- The full adder is described as a *module* named FA_Struct; a module must start with the keyword module and end with the keyword endmodule
- Module name is followed by the port declarations; in this case the ports are: a, b, cin, cout and sum
- Next, the direction of each port is specified; in this case ports are either input or output; bidirectional ports are also possible
- Ports in a structural description are connected via nets; each net should have only *one active driver at a time*; multiple drivers are also possible, but only one of those should actively drive the net at a time; that could be achieved using tri-state drivers
- Apart from the nets declared via the ports, other nets are: w1, w2, w3 and w4
- A module may have other module instantiations which may be other user defined modules or primitive gates; in this case the primitive gates used are: xor, and and or
- Encapsulation of designs as modules and their use to create hierarchical designs makes the design well structured and easier to understand
- An instanted module may be optionally given a name such as: U1, U2, U3, U4, U5 or U6
- For the primitive gates, note that the first port serves as the output while the rest of the ports serve as the inputs

Structural specification of a full adder with delays

```
module FA_Struct_Delay(a,b,cin,cout,sum);
 input  a, b, cin;    // inputs
 output cout, sum;    // output
 wire w1, w2, w3, w4; // internal nets

 xor #(10)    (w1, a, b); // delay time of 10 units
 xor #(10)    (sum, w1, cin);
 and #(8)     (w2, a, b);
 and #(8)     (w3, a, cin);
 and #(8)     (w4, b, cin);
 or  #(10, 8) (cout, w2, w3, w4); // (rise time of 10, fall 8)
endmodule
```

Real gates have finite delays; those may also be modelled using verilog; the following may be noted:

- Properly specified delays can make the simulation more realistic
- Specification of delays is optional
- Delays are not synthesisable, meaning that while those can be simulated, a CAD tool does not create a implementation having those delays, as specified

Structural specification of a 4-bit ripple carry adder

```
module rca4(A, B, cin, S, cout);
 input[3:0] A, B;
 input cin;
 output[3:0] S;
 output cout;

 wire c1, c2, c3;
 // 4 instantiated 1-bit full adders
 FA_Struct fa0 (A[0], B[0], cin, c1, S[0]);
 FA_Struct fa1 (A[1], B[1], c1, c2, S[1]);
 FA_Struct fa2 (A[2], B[2], c2, c3, S[2]);
 FA_Struct fa3 (A[3], B[3], c3, cout, S[3]);
endmodule
```

- This is a recursively described 4-bit ripple carry adder in terms of full adders
- Note that bundles of related bits can be represented as bit vectors; in this case the inputs and outputs are represented as bit vectors

Structural specification of a DFF

```
`timescale 10us/1ms

module dff_Struct(clk, D, Q);
 input  clk, D;
 output Q;

 wire   clk, s0, s1, D, Q, Q_BAR; // internal nets

 nand U1 (s0,D,clk);   // U1: local name of the nand instance
 nand U2 (s1,s0,clk);  // U2: local name of the nand instance
 nand U3 (Q,Q_BAR,s0); // U3: local name of the nand instance
 nand U4 (Q_BAR,Q,s1); // U4: local name of the nand instance

endmodule

module top();
// Testbench of above code
 wire   clk, D, Q, Q_BAR; // internal nets

dff_Struct U1 (clk, D, Q);

initial begin
 $monitor( "clk = %b D = %b Q = %b Q_BAR = %b" ,clk, D, Q, Q_BAR);
 clk = 0;
 D = 0;
 #3 D = 1;
 #3 D = 0;
 #3 $finish;
end

always #2 clk = ~clk;

endmodule
```

Here a structural description of a simple D-FF is shown along with the top module that governs the simulation

- By convention, the topmost module is named top
- It has port and wire specifications as usual
- The module to be simulated is instantiated, in this case: dff_Struct
- Note the use of the sequential procedural statement: initial; it executes only once at the beginning
- It's used in conjunction of a sequentially interpretted group of statements within a block demarcated with begin and end
- The $monitor statement is much like printf in C; $monitor displays the values of its parameters *every* time *any* of its parameter changes value
- It's convenient for tracking the values of the signals marked for monitoring
- The simulation is orchestrated as follows:
    1. clk and D are bot set to 0
    2. After a time delay of 3 units, D is set to 1
    3. After a time delay of another 3 units, D is set to 0
    4. After a time delay of another 3 units, the simulation is terminated
- Also, note that use of the concurrent statement: always
- As the simulation proceeds, always the value of clk is complemented after 2 time units to simulate a free running clock, until the simulation is terminated
- `timescale <time_unit>/<time_precision> helps the simulator to interpret the delays

Assignment statement

- Encode the (hierarchical) implementation of the sequential double dabble algorithm for eight bits in verilog
- Write the top module to simulate the working of the convertor in the requisite number of clock cycles
- Use iverilog – see the notices for the software and also more information on verilog

Marking guidelines

Assignment marking is to be done only **after** the deadline expires, as submissions gets blocked after the assignment is marked. Enter the breakup of marks while marking.

Using verilog	
Properly coded D-FF, multiplexer, the input shift register, a FA, a 4-bit RCA and the nibble converter	**5×7=35**
Proper coding of the top module	**10**
Correct simulation	**5**
Total Marks	***50***

Assignment submission

A PDF report, as appropriate, should be submitted. Submit all your files together.

Use electronic submission via the WBCM link

You should keep submitting your incomplete assignment from time to time after making some progress, as you can submit any number of times before the deadline expires. **You should submit all your files together**.

Warning

Cases of copying will be dealt with seriously and severely, with recommendation to the Dean to de-register the student from the course.