CS203 – Design of Digital Systems Lab

A3 - Hardware Modules in Verilog - I

Points to Note

- Design and test each of the listed hardware modules in the Verilog HDL.
- In the comments section of each Verilog code, include a short description of the program, your name, roll number, date of writing the program and other information you deem relevant.
- (a) Report: Every question has a corresponding solution containing block diagram(s)/ gate-level diagram(s), short description, other relevant info. (b) Auxilary files to submit: Per question, include the following files along with the report: Verilog code, the testbench including the monitor and stimulus, execution screenshots, VCD dump, gtkwave screenshots. (c) Archive: Organize your submission into directories per question.
- Stress test your modules by generating all combinations of inputs in the testbench. Your testbench module should have capability to verify the output produced by your design.
- This is a team assignment. One submission per team.
- (a) Submission is due October 11, 9AM. Pack your report, code, screenshots and other files in an archive and email to `cs201.nitk@gmail.com`.
- Install iVerilog (http://iverilog.icarus.com/) and gtkwave. Ubuntu: `$ sudo apt-get install iverilog gtkwave`.
- Verilog Tutorials: http://vol.verilog.com/, http://www.asic-world.com/verilog/veritut.html, https://www.nandland.com/verilog/
- The DDS textbook *Digital Design - With an Introduction to the Verilog HDL*, 5e, Mano & Ciletti, Pearson, has some excellent Verilog introduction in Sections - 3.9, 4.12, 5.6, and 6.6.

Modules

1. **Hello World Program**. A hello world message should be printed out by a module periodically. The module should also print the current timestamp (in clock cycles). Module terminates after 100 clock cycles.
2. **Universal gates**. Implement 2-input, 3-input NAND and NOR gates.
3. **Basic gates**. Implement 2-input, 3-input AND, OR, NOT, XOR, and XNOR, gates using the universal gates from previous Q as submodules.
4. **Half Adder**. Implement a combinational half adder (HA).
5. **Full Adder**. Using the HA module from the previous question, Implement a combinational full adder (FA).
6. **4-bit Adder/Subtractor**. Using the FA module from the previous question, build a 4-bit Adder/Subtractor module. Extend this design to make an n-bit Adder/Subtractor module. n should be configurable. Possible values are 4,8,...,256.
7. **4-bit Carry Lookahead Adder**. Implement a 4-bit 4-bit Carry Lookahead Adder (use the design from the textbook).
8. **BCD Adder**. Implement the BCD Adder shown in class. Use either of the Adders implemented above.
9. **2-bit Multiplier**. Implement the 2-bit multiplier. Use modules designed above as submodules.
10. **Magnitude Comparator**. Implement the 4-bit magnitude comparator (shown in TB/class).
11. **Code Converters.** Implement a module to convert 4-bit Gray code to Binary code. Implement a module to convert 4-bit Binary numeral to Gray code. Repeat for Excess-3 to BCD code and vice-versa.
12. **Encoders and Decoders**. Implement a combinational 4-to-16 decoder and a 16-to-4 encoder. Extend both the designs to add the *Enable* signal.
13. **Mux and Demux**. Implement a 16:1 combinational mux. Implement a 1:16 combinational demux. Each input is 32 bits wide.

CS203 – Design of Digital Systems Lab

A4 - Hardware Modules in Verilog - II

Points to Note

- Design and test each of the listed hardware modules in the Verilog HDL.
- In the comments section of each Verilog code, include a short description of the program, your name, roll number, date of writing the program and other information you deem relevant.
- (a) Report: Every question has a corresponding solution containing block diagram(s)/ gate-level diagram(s), short description, other relevant info. (b) Auxilary files to submit: Per question, include the following files along with the report: Verilog code, the testbench including the monitor and stimulus, execution screenshots, VCD dump, gtkwave screenshots. (c) Archive: Organize your submission into directories per question.
- This is a team assignment. One submission per team.
- Stress test your modules by generating all combinations of inputs in the testbench. Your testbench module should have capability to verify the output produced by your design.
- (a) Submission is due October 21, 9AM. Pack your report, code, screenshots and other files in an archive and email to `cs201.nitk@gmail.com`.

Modules

1. **Single bit Sequential Memory Element**. Implement the following 3 versions of the 1 bit D-Flip Flop. All the three versions should work with the same Testbench code.
   (a) Use the Master-Slave D-Latch block diagram shown in Fig. 5.9, Page 198, Digital Design, 5e, Mano and Ciletti.
   (b) Code the gate-level D-FF shown in Fig. 5.10, Page 199, Digital Design, 5e, Mano and Ciletti.
   (c) Write the Behavioral code for the D-FF.
2. **Register**. Use the memory cell (D-FF) designed above to implement a 32 bit register (instantiate the 1-bit D-FF 32 times). The data input and data output are now 32 bits wide (instead of 1-bit in the case of the memory cell). The reset signal is still 1-bit signal as in the case of the memory cell. Perform repetitive Read-Write operations on the Register.
3. **Register File**. Use the Register from the previous Q and design a Register File containing 32 registers. The register file contains these inputs: (a) Address (identifies the register to read/write from, (b) Read/$\overline{Write}$ signal (indicates whether to Read from the register OR write into the register), (c) 32-bit Data - this is written into the register if the operation is Write, (d) Reset - Resets all registers to 0. Perform repetitive Read-Write operations on the Register. You may assume that all inputs, except Reset, are fed synchronously to the RF.