Lab 2 - VHDL

Setup

Windows:

- Use any IDE for VHDL programming
- Install ModelSim Altera from this link. Do not forget to switch to the windows section.
- Follow the installation setup for ModelSim
- Watch posted tutorials for coding in VHDL and using ModelSim

Linux:

- We will be using GHDL and GTKWave
- Install them by : sudo apt-get install ghdl gtkwave
- Here is a simple tutorial for their usage

Mac:

- We will use GHDL and GTKWave
- Install them by: brew install --cask ghdl
  brew install --cask gtkwave

Here is a link to tutorials in VHDL. You are supposed to watch these tutorials before coming to the lab for doubts (on Monday).

If there are any issues regarding installation, feel free to post on piazza!

Question 1 (20 marks)

a) Write specification for the basic 2-input logic gates in VHDL. Define the entity and architecture of these. The gates which you have to code are: AND, OR, NOT. Generate a wave-form for the inputs and outputs using a simulation tool (modelsim or GTKWave). Take a screenshot of the wave-form.

b) Design a 4 x 2 multiplexer and a 4 x 2 encoder using the basic gates that you have designed above. Do not use the built-in keywords ‘and’, ‘or’, ‘not’ etc. You must use your designed basic gates in part a) as components. Generate a wave-form for the inputs and outputs using a simulation tool (modelsim or GTKWave). Take a screenshot of the waveform.

Use the following entity definitions:

- entity AND_Gate is
  port(x1: in std_logic;
  x2: in std_logic;

```
        y: out std_logic);
    end entity;
```
- ```
  entity OR_Gate is
        port(x1: in std_logic;
        x2: in std_logic;
        y: out std_logic);
    end entity;
  ```
- ```
  entity NOT_Gate is
        port(x: in std_logic;
        y: out std_logic);
    end entity;
  ```
- ```
  entity mux4x2 is
        port(D: in std_logic_vector (3 downto 0);
        S: in std_logic_vector(1 downto 0);
        Y: out std_logic);
    end entity;
  ```
- ```
  entity encoder4x2 is
        port(I: in std_logic_vector (3 downto 0);
        Y: out std_logic_vector(1 downto 0);
    end entity;
  ```

Note: For both parts a) and b), you have to write a test-bench. Note that you must cover all possible inputs in your test-bench.

Question 2 (30 marks)

Design a 8 bit Ripple Carry adder. Google what adders are and what a Ripple Carry Adder is. This must be designed in a structural way only (no sequential way of coding allowed). Try to modularize your code as much as possible. Use the following entity definition:

```
entity RCA8 is
    port(a, b: in std_logic_vector(7 downto 0);
    sum: out std_logic_vector(7 downto 0);
    cout: out std_logic);
end entity;
```

Description:

‘a’ and ‘b’ are the two input 8 bit numbers. ‘sum’ is the output addition of ‘a’ and ‘b’. ‘cout’ is the output carry. Write a testbench to test your adder. The testbench should showcase two examples of numbers being added (each example consists of two numbers to be added).

Question 3 (50 marks)

In this question, we will design a data compression circuit using run-length encoding. We want to design a data compression circuit using run-length encoding. It replaces continuously repeated occurrences of a byte by a repeat count and the byte value. The circuit receives a fresh byte at every positive transition of an externally supplied clock. We shall use the ‘ESC’ character (code = 1BH) to signal the use of a repeat count. Therefore, if the ‘ESC’ character itself appears in the input stream, it has to be handled in a special way. The output is one byte wide and every data byte being output is signaled by a rising transition on a DataValid line.

- If any character ‘c’ repeats ‘n’ times in the input stream such that 2<n<16 then we output the three-byte sequence “ESC n c”.
- If ‘n’ number of ‘ESC’ characters arrive contiguously in the input stream, we output the 3-byte sequence “ESC n ESC”, where n can be from 0<n<16. Otherwise, we just output the received characters without any change.
- Notice that in both of the above cases since the output of your circuit is only 1 byte wide, it cannot output all the 3 bytes at once. Only 1 byte is output per clock cycle and whenever a byte is being output, the Data Valid output line must go from low to high.
- If the repeat count is more than 15, we handle the first 15 characters as above and treat the 16th occurrence onwards as if a new character has been received.
- This circuit reads from an input file and sends the output to an output file. The reading and writing operations are done in the test bench itself. Here is the link to the testbench. Modify this test bench according to your need.
- The input file will have the following constraints:
    - The data will be present in binary ASCII format (Link to the ascii table).
    - Only small and capital letter alphabets will be used. Beside ‘ESC’ and ‘SPACE’, no other special character is permitted.
    - There will be only one character per line in the file. (That is every line will have only 8 binary bits).
    - The file will not have more than 32 bytes. (i.e. not more than 32 characters).

- The testbench will read the output from your circuit and print it to an output file. It will print only one character (8 bits) per line.

Need for a buffer: Fresh data is input at every rising edge of the clock. Depending on whether or not there is repetition in the input there may or may not be a fresh valid byte on the output lines. Since the output data rate may be less than or equal to or greater than the input data rate over short periods, provision must be made for buffering the input data and providing a data valid output line which becomes 1 in a clock cycle in which valid data is being output. Decide on a safe buffer size based on the constraints of the input file size mentioned above (Another approach is that you can choose whatever buffer size you want but then there should be a mechanism in your circuit for handling the situation when the buffer gets full).

Use the following entity definition:

```
entity RLE_Encoder is
        port(clk, rst: in std_logic;
        a: in std_logic_vector(7 downto 0);
        data_valid: out std_logic;
        z: out std_logic_vector(7 downto 0));
end entity;
```

Write a report on how you implemented the RLE encoder. It should contain your structure for the RLE encoder, assumptions (if any) and other details.

Important: Place the input and output files in the same directory as the vhdl files. Read input from the file: test.txt and write output to the file: out.txt (that is, do not change the input and output file names and positions)

For Q3, here is a sample input file and output file. If you are using ghdl, you will need to add the -fsynopsys option while compiling. We will verify your output using the command diff -Bw true_output.txt out.txt

Further, for Q3, note that the input may get processed completely whilst your output has not been completed yet. In that case you need to pump extra clock ticks to get the complete output. We will run the simulation for 1000 ns.

Note: Do not write sequential code anywhere other than the test-bench (For Q1 and Q2).
Note: You are expected to use sequential coding in Q3. It will make your life easier :)

Note: For Question 2, to get full marks, you must modularize your code to the best extent possible (Hint: Use a full adder as a component)

Submission:

```
roll_number
├── Q1
│   ├── and.vhd  and_tb.vhd  or.vhd  or_tb.vhd   not.vhd  not_tb.vhd
│       and_wave.jpg  or_wave.jpg  not_wave.jpg mux4x2.vhd enc4x2.vhd
│       mux4x2_tb.vhd enc4x2_tb.vhd mux4x2_wave.jpg enc4x2_wave.jpg
├── Q2
│   ├── RCA8.vhd  tb.vhd  wave.jpg *.vhd
├── Q3
│   ├── RLE.vhd  tb.vhd  wave.jpg report.pdf  *.vhd
```

Where *.vhd represents other necessary files for compilation. You can name the extra files as you like but DO NOT change the name and format of the original entity file.

Compress this directory using `zip -r roll_number.zip roll_number` and submit roll_number.zip on moodle.
Incorrect submission formats will result in no marks.

Note: Replace roll_number used throughout with your roll number.

Grading Scheme:
Question 1 and 2: binary grading
Question 3: Partial credit will be awarded based on your performance in the viva and your design structure for the RLE encoder. Note that this does not mean that the viva will test only Question 3.