Simulation

Design Verification

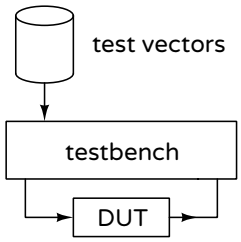
Verification is the process of checking that a digital circuit will meet its requirements. This involves testing both the logical design (e.g. correct outputs and state transitions) as well as the electrical requirements (e.g. voltage levels and setup times).

It's often more practical to verify a design by simulating it than by building it.

In this lab you will simulate the operation of a "moving-average" circuit that outputs the sum of the current input and the previous input.

Testbenches

A "testbench" is HDL code that applies inputs to the design being tested (often called the "device under test" or DUT) and checks that the outputs are correct:



The values of the inputs and expected outputs are called "test vectors." Test vectors are often read from a file generated by other software. In this lab you will use a spreadsheet to generate a text file containing the test vectors.

In a previous course you may have use a stimulus-only testbench that applied inputs to the DUT and displayed the outputs. These simulations are mainly useful during the initial design process.

However, this approach is not practical for complex designs or if we want to automate testing to ensure new errors ("regressions") are not introduced. Once the expected output has been determined, a "self-checking" testbench is typically used to check the outputs and flag any differences.

Writing a testbench requires HDL language features and programming skills that are beyond the scope of this course so you will be supplied with a self-checking testbench and asked to create a file with test vectors.

Procedure

You will use the Modelsim simulator. The "Web" edition can be downloaded for free from the same web site as Quartus.

Create a project directory (folder) for the simulation. Download the **`lab7_tb.vhd`** testbench from the course web site to this folder.

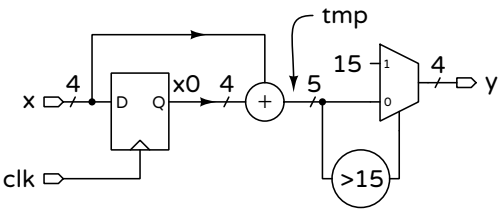
Design the Moving Average Filter

A moving-average filter is a circuit that outputs the average of previous values. In this lab you will design and test a circuit that outputs the sum of the current input and the previous input (the input at the most recent rising edge of the clock).

Additionally, the filter should use "saturation arithmetic." This means that values that would exceed the maximum possible output value should output that value rather than overflowing.

For example adding the 4-bit unsigned values 8 and 9 would result in the value 17. In a conventional 4-bit adder the overflow would be lost and the result would be 1. With saturation arithmetic the result would be 15 – the maximum unsigned value that can be represented with 4 bits.

The following diagram shows a block diagram of the moving-average filter with a saturation arithmetic adder:



Write the architecture for an entity named **`lab7`** that has the following declaration:

```
entity lab7 is
  port (
    x : in unsigned (3 downto 0) ;
    clk : in std_logic ;
    y : out unsigned(3 downto 0) ) ;
end ;
```

and save it to a file named **lab7.vhd**.

Note that VHDL requires that the operands of a sum be of the same length as the result. Thus to obtain a 5-bit result from adding two 4-bit values you must extend each operand by concatenating zeros in the most-significant bit position. For example:

```
tmp <= ('0'&x0) + ('0'&x) ;
```

Follow the course coding guidelines, including adding a comment at the beginning of the file with your name and the date.

Create the Test Vectors

For this circuit the test vectors are pairs of values of the input, **x**, and the expected output, **y**. The **y** value is the (saturated) sum of the value of **x** and the previous value of **x**. The testbench applies the **x** value and checks the **y** value before generating a rising clock edge.

The supplied testbench reads the test vectors from a text file, **lab7.tv**, which you must create.

You can create the file by hand with a text editor or using a spreadsheet and then copy/pasting the values into a text file.

The sequence of input **x** values should be 0, 7, 8, and 9 followed by the eight digits of your BCIT ID. For example, if your BCIT ID is A00123456 the test vector file should contain:

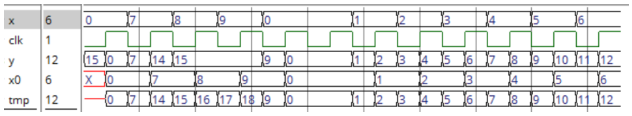
```
0 0
7 7
8 15
9 15
0 9
0 0
1 1
2 3
3 5
4 7
5 9
6 11
```

A convenient way to generate this file is to use a spreadsheet. The first column should have the **x** values the second column should have a formula that computes the **y** values. For example, the second **y** value could be computed with the formula **=MIN(15,A1+A2)**[1]

Run the Simulation

Follow the procedure in Appendix A to create a simulation project, add the **lab7.vhd** and **lab7_tb.vhd** files to the project and compile them. After fixing any errors, run the simulation. The Transcript window should show the messages generated by the testbench and the Wave window should show the signal waveforms.

The waveforms for the above test vectors would look as follows:



Submit Results

Submit the following files to the Lab 7 Assignment folder:

- your **lab7.vhd** VHDL file
- your **lab7.tv** test vector file
- a screen capture of the waveforms similar to that shown above
- a screen capture of the Transcript window showing the messages generate from running the simulation similar to that shown below:

```
VSIM 189> run -all
# ** Warning: NUMERIC_STD."<": metavalue detected, returning FALSE
#    Time: 0 ps  Iteration: 0  Instance: /lab7_tb/dut
# ** Warning: NUMERIC_STD."<": metavalue detected, returning FALSE
#    Time: 0 ps  Iteration: 1  Instance: /lab7_tb/dut

VSIM 190>
```

[1] Since this formula uses relative addresses you can copy it to the other cells in the **y** column.

Appendix A - Simulation with Modelsim

A simplified procedure is as follows:

1. start the program (on Windows select Altera ... > Modelsim ...)
2. if this is the first time you simulate these files, select **File** > **New** > **Project**..., select the folder where your files are located as the Project Location and enter a suitable Project Name (e.g. `lab7`), then click OK,
3. select Add Existing File and select the file(s) that contain the entities you want to simulate, including the testbench, then select Close
4. otherwise, if you had already created a simulation project and it's not already open, select **File** > **Open**, select Files of type: Project Files (*.mpf) and select the project file,
5. select **Compile** > **Compile All** to compile all the files in your project into the `work` library,
6. if there are syntax errors you will need to fix the error(s), save the file and go back to step 5,
7. otherwise select **Simulate** > **Start Simulation**; select your testbench module from the `work` library and select **OK**,
8. drag the signals you wish to view from the 'Sim' or 'Object' windows to the 'Wave' window (use the **View** menu to open windows)
9. select **Simulate** > **Run -All**; this will run the simulation until it is complete
10. the Transcript window will contain output from the testbench
11. the Wave window will show the waveforms (select the Wave window, click on '+' and **Wave** > **Zoom** > **Full**); you can use a screen capture utility (e.g. Windows Snip tool) to save the waveforms
12. if the results are not as expected, correct the errors, run **Compile** > **Compile All**, **Simulate** > **Restart**..., click **OK** and **Simulate** > **Run -All**