Simulation

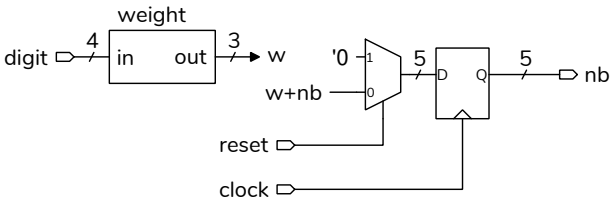
Introduction

In this lab you will design and simulate a circuit that computes the Hamming weight (defined below) of a sequence of 4-bit values.

Requirements

You must write a module named **lab5** that implements the block diagram below. The **nb** register is set to 0 when **reset** is asserted. Otherwise the output of the **weight** module is added to **nb**.

A **weight** module must be instantiated in **lab5**. It sets **out** to the Hamming weight of **in**.



You must test your design by writing a testbench that asserts reset for one clock cycle and then sets **digit** to the BCD values of the 8 digits of your BCIT ID on successive clock edges. Your testbench must check the **nb** output, print the test vector and the module outputs, and flag any differences.

For example, if your student number were A00123456 then successive **digit** inputs and expected **nb** outputs would be:

```
0,0
0,0
1,1
2,2
3,4
4,5
5,7
6,9
```

An additional test vector at the end should contain an error and so an error message will be printed. For example:

```
...
7,0
```

Hamming Weight

The Hamming Weight is the number of 1 bits in a number. For example, the Hamming weight of 0 (4'b0000) is 0, the weight of 1 (4'b0001) is 1, of 6 (4'b0110) is 2, etc. Your **lab5** module must instantiate a module named **weight** that computes the Hamming weight of a 4-bit input.

Some ways of computing the Hamming weight include: a lookup table (**out=lut[in];**), computing the sum of the bits (**out=in[0]+in[1]+...**), or conditional operators (**out=in==0?0:in==1?1:...;**).

Pre-Lab

Write the **lab5** module describe above and create the test vector file in **.csv** format. Your module must, at a minimum, have a correct module declaration.

Procedure

Write the **lab5** module described above and a testbench that applies test vectors read from a **.csv** file. You should be able to use the example testbench in the lecture notes with a few appropriate modifications.

Follow the course coding guidelines when writing the DUT, including adding the appropriate comment at the beginning of each file.

Your testbench should print:

- Each test vector as it is read from the file.
- The output of your DUT
- An error message if there are mismatches between the expected and actual outputs. See the example testbench for an example.

You can create the **.csv** file containing the test vectors using a text editor such as Notepad. You can also use a spreadsheet and export the test vectors to a **.csv** file[1].

[1] But don't use a UTF-8 (Unicode) output encoding.

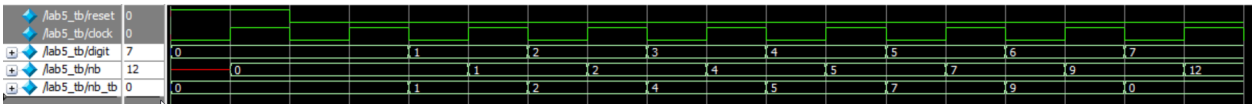


Figure 1: Simulation results (Modelsim).

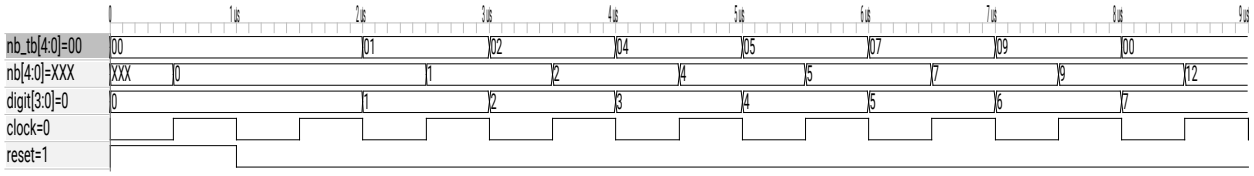


Figure 2: Simulation results (gtkwave).

Follow the procedure in the *Software Installation and Use* document and the video on the course web site to create a simulation project, add the file(s)[2] with your modules to the project and compile them. Copy the test vector file to the project folder. Add the clock, reset, input, and output signals to the Wave window. Run the simulation. The Transcript window should show any messages generated by the testbench and the Wave window should show the signal waveforms.

Report

Submit a PDF file to the appropriate Assignment folder that includes the following:

- Listing(s) of your DUT and testbench modules.
- A screen capture of the simulation waveforms similar to that shown above. Set the display format for the output waveform to unsigned decimal[3].
- A screen capture of the Transcript window showing the messages generated by running the simulation. For example:

```
run -all
#    digit   nb_tb      nb
#        0       0       0
#        0       0       0
#        1       1       1
#        2       2       2
#        3       4       4
#        4       5       5
#        5       7       7
#        6       9       9
#        7       0      12 *** Error
```

[2]The DUT modules and the test may be placed in the same file or in individual files but each file must be added to the project.

[3]Right-click on the waveform name and select Radix / Unsigned