Lab 2 – Encoder Design
Four bit, Error Correction Code (ECC) using Hamming(7,4) Code

Introduction

As discussed in the lectures, videos and textbook, parity is used as the simplest form of error detection. An expansion of this concept was developed by Richard Hamming at Bell Laboratories in 1950. His basic algorithm can detect all single and two bit errors, and identify to correct a single bit error. The algorithm you will be using in this lab is called the Hamming(7,4) code. You will be adding 3 code bits to a 4 bit data word. The 3 code bits are basically parity bits of selected bits of the data word. The basic algorithm is provided below. The theory behind it is available from many sources and will not be covered here.

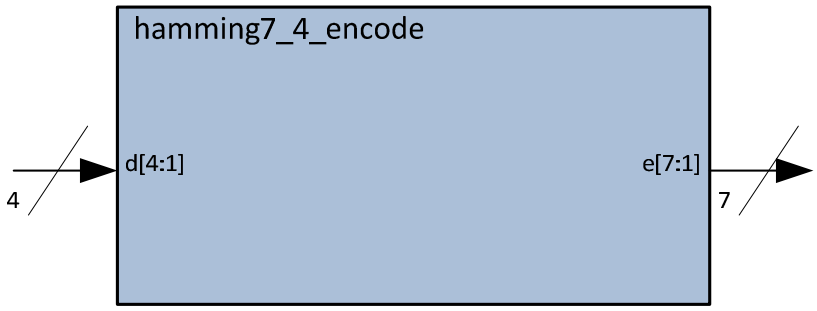
The input to your design will be four data bits (**d[4:1]**). The output (**e[7:1]**) will be a seven bit word as described in the truth table below. [Note that we are using indices that start from 1, not 0. This is because most all references of Hamming's implementation begin index counting with 1, not 0 – so to avoid confusion we follow this convention for this design.]

d[4:1]	e[7:1]
4'b0000	7'b0000000
4'b0001	7'b0000111
4'b0010	7'b0011001
4'b0011	7'b0011110
4'b0100	7'b0101010
4'b0101	7'b0101101
4'b0110	7'b0110011
4'b0111	7'b0110100
4'b1000	7'b1001011
4'b1001	7'b1001100
4'b1010	7'b1010010
4'b1011	7'b1010101
4'b1100	7'b1100001
4'b1101	7'b1100110
4'b1110	7'b1111000
4'b1111	7'b1111111

You are provided a skeleton design in your ***lab2*** directory path. The project name is ***lab2.xpr***. This project will contain the basic files from which you will build the design. They are described below.

Design/Module

The block diagram for the design is given here. Note the module name and the internal signal names.



The implementation algorithm is described below.

The three code bits are:

```
p1 = d[1] xor d[2] xor d[4]

p2 = d[1] xor d[3] xor d[4]

p3 = d[2] xor d[3] xor d[4]
```

The encoded multi-bit output signal, **e**, has these three code bits interleaved with the data bits, such that:

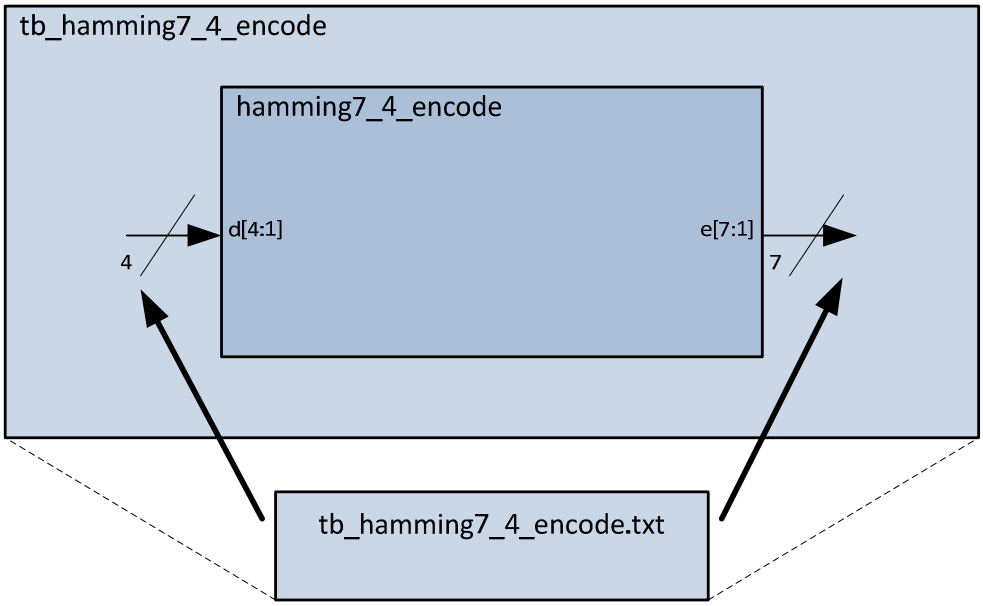
```
e[7:1] = {d[4], d[3], d[2], p3, d[1], p2, p1}
```

The result of this algorithm exactly matches the truth table above.

Open the module, ***hamming7_4_encode.v***. Create your Verilog design that meets the above specifications. For more information as to how to write Verilog **`assign`** statements from Boolean equations, refer to section 2.1.3 in the textbook by Ashenden.

Simulation/Verification

A self-checking testbench file/module has been provided called ***tb_ hamming7_4_encode.v***. Open the file and look it over – there are no edits required in this file. You will note that besides instantiating your *hamming7_4_encode* design, it also reads a text file, ***tb_hamming7_4_encode.txt***, and inside a *for* loop it applies the input stimuli and compares expected output values to those generated by your design. A block diagram of this process is provided below. You can also open the text file and observe the format of the data that represents that in the truth table above.

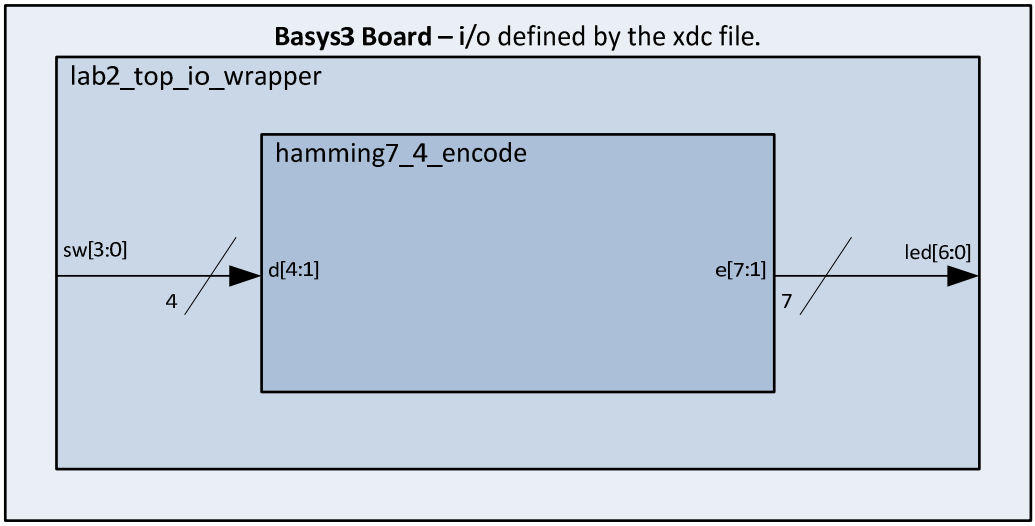


Run your simulation, check for mismatches, check the text output and timing diagram. Correct your design as needed.

If you need more information on how to use this tool for debugging, follow the instructions on the course web page.

Synthesis

When your simulation runs with no mismatches, you can then go through the synthesis steps and bit file generation process. If you want to see how the i/o wrapper module instantiates your design module, you can open the file: ***lab2_top_io_wrapper.v***. Note how the switches and LED are connected to your design: ***hamming7_4_encode***. It is illustrated graphically below:



You can also look over the user configuration file, ***lab2.xdc***. This was copied from a file available on Digilent's web site and edited to meet our needs. It describes how the Verilog top level signals are mapped to the FPGA pins for the Basys3 board.

Finally, generate a bit file as you did in Lab 1 (*right click:* ***lab2_top_io_wrapper.tcl***). For this lab the generated file will be called: ***lab2_top_io_wrapper.bit***, in the local directory path. Copy it to your memory stick to load and run on your Basys3 board.

As shown in a above figure, the lower 4 switches will select the input data pattern and the lower 7 LED's will represent the Hamming output code. Verify that your code is working properly by comparing the LED outputs of the sixteen possible switch inputs from the truth table provided.