UNIVERSITY OF CALIFORNIA IRVINE
COURSE: INTRODUCTION TO DIGITAL LOGIC LAB (31L) FALL 2015

Lab 2 Report - Group 9

Kelvin Phan 51197373

Patrick Skoury 75202200

Aaron Liao 90811748

Shawn Howen 60029119

Andrew Mehta 44592437

November 1, 2015

1 CREATING THE HAMMING ENCODER

1.1 ARCHITECTURE

Our design for the 11 to 15 Hamming Encoder mainly consisted of multiple XOR statements and the concatenation of our input data with parity bits. One input and output were defined as "11 DOWNTO 1," and "15 DOWNTO 1" respectively. Originally we had defined the vectors using "TO", so position 1 would be the most significant bit. Upon seeing the C Reference Model, we changed the input and output to "DOWNTO" in order to remain consistent and obtain the correct output. In the Architecture, the four parity bits were defined as signals. Each parity bit was set to the result of several input bits being XOR with each other. In order to determine which input bits were used for each parity, we followed the chart given through the lecture. The final statement set the output equal to the concatenation of the input bits and parity bits. The parity bits were implemented from the right side of the output vector, starting with the least significant bit position.

2 CREATING THE HAMMING DECODER

2.1 ARCHITECTURE

Our design for the 15 to 11 Hamming Decoder followed a similar structure as our encoder, but also included the ability to detect and fix a single bit error in the input. This time, the input was 15 bits and the output was 11 bits, which is the opposite of the encoder. Once again, we had to change the vectors to "DOWNTO." In the Architecture, we defined the parity bits once again, as well as a "data_temp" and "syndrome" signal. These two new signals were 15 and 4 bit vectors respectively. Each parity bit was set to the result of multiple XOR statements using the input bits, similar to the encoder. The signal "syndrome" was set to the concatenation of our parity bits, with the 4th parity being the most significant bit. This vector helped us locate an error, so that we could invert that particular bit. Our signal "data_temp" was set to equal 12 possible cases, depending on the value of "syndrome." The first case set "data_temp" equal to the input when "syndrome" was locating an error in one of the parity bits or there was no error at all. The rest of the cases set "data_temp" equal to the input, except with the single-bit error inverted. The output of the encoder was set to "data_temp" with all of the parity bits removed, so that the final result would be the 11-bit input that was initially fed to the encoder.

2.2 ERRORS ENCOUNTERED

2.2.1 SIMULATION

```
# vsim -c tb_top -sv_lib /users/ugrad2/2015/spring/kelvinhp/EECS_31L/Assignment_2/
assignment-2/sim/so_libs/TEST -do "sim.do"
# Start time: 11:21:28 on Oct 30,2015
...
# do sim.do
# waveform.wlf
# ** Fatal: (vsim-3471) Slice range (1 to 2147483647) does not belong to the prefix
index range (1 to 15).
#    Time: 0 ns  Iteration: 0  Process: /dpi_example/L2/line__55 File: ../design/
ham1511_decode.vhd
# Fatal error in Architecture Structural at ../design/ham1511_decode.vhd line 58
#
# HDL call sequence:
# Stopped at ../design/ham1511_decode.vhd 58 Architecture Structural
#
# End time: 11:21:29 on Oct 30,2015, Elapsed time: 0:00:01
# Errors: 2, Warnings: 13
```

Resolution: During simulation phase, a fatal error was encountered at line 58 of the Hamming Decoder. The error stated that the slice range did not belong to the prefix index range. This error occurred before we switched from "TO" to "DOWNTO." Originally, we had used "err_loc" as an integer, rather than "syndrome," to locate and fix the error. The error occurred at this line:

```
data_temp <= data_in(1 TO err_loc-1) & not(data_in(err_loc)) & data_in(err_loc+1 TO 15);
```

This was the third case for "data_temp." The other cases involved when "err_loc" was zero, one of the parity bits, or the least significant bit. We believe the error resulted because "err_loc" was not the number we thought it was, but rather a large integer that was not within the range of "data_in." When defining "err_loc," we used the line:

```
err_loc <= conv_integer(p1)*1 + conv_integer(p2)*2 + conv_integer(p3)*4 + conv_integer(p4)*8
```

Our intention was to convert each parity bit into an integer, multiply each by its position, and add the integers up in order to determine the error location. The problem was that subtracting "err_loc - 1" and adding "err_loc + 1" resulted in a number that was out of bounds. In order to solve this error, we concatenated the parity bits into the vector "syndrome" and used that signal to find the error location. Rather than using a shortcut, we listed out all 12 cases for "data_temp," which was described in the Architecture of our decoder. Then, we could set the output of the decoder to "data_temp" without the parity bits.

3 Creating the Testbench

3.1 Architecture

Our design for the testbench, written in SystemVerilog, allows us to verify the functionality of our encoder and decoder, as well as generate random inputs. The testbench uses DPI with the C-language, which allows us to compare our VHDL encoder output with the output of a C Reference Model. The first step in creating the test bench was modifying the template given in class. We changed the "rtl_data_in" and "rtl_data_out" to packed arrays, which were fed into the C program. All of the arrays also had to be changed to 15-11 bits, rather than 7 to 4. After defining all logic, a clk was defined and the entire simulation was set to 80 ns under the "inital begin" statement. The clk was generated with a 2 ns period using an always statement. An always statement, dependent upon the positive edge of the clock, was used to generate "rtl_data_in" as 11 random bits. This stream of bits was displayed using the "$display" function. Data was sent to the C Reference Model under the method "encoder," then "exp_data_out" was set as the output of the method concatenated with parity bits. Our Hamming Encoder and Decoder were then instantiated and given the proper input and output. The next always statement was used to compare the C program output with the Hamming Encoder output at the negative edge of the clock. A "pass" or "fail" message was displayed, depending on whether both outputs were equal. The final always statement compared the output of the Hamming Decoder with the first input, "rtl_data_in." This also occurred at the negative edge of the clock and displayed a "pass" or "fail" message. In total, we fed 40 random inputs to the design over a span of 80 ns.