Lab 2 Advanced: Hamming Code Encoder & Decoder

Submission Due Dates:

Demo:	2024/10/01 17:20
Source Code:	2024/10/01 18:30
Report:	2024/10/06 23:59

Objective

1. Getting familiar with Verilog sequential/combinational behavior modeling.
2. Getting familiar with the switch and LED control on the FPGA demo board.

Introduction About Hamming Code

Data transmission in real life will suffer from data corruption. To address this problem, scientists have come up with various error detection and correction algorithms.

In this lab, we will use Hamming codes to encode the data. This method involves adding parity bits to the original data and using the parity check to detect and correct the error bits. We use even parity to construct the error correction code.

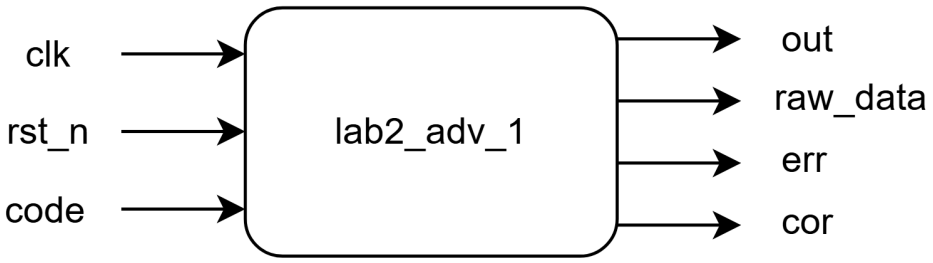
There are 2 different types of rules that are introduced in this lab. In 2-1, we will use Single Error Correcting (SEC), which can correct 1 bit error, and in 2-2, we will use Single Error Correcting, Double Error Detecting (SECDED), which can not only correct 1 bit error but also detect double bits error.
Rules for them will be given below.

For more information, please check the following link:
Hamming code: https://en.wikipedia.org/wiki/Hamming_code
Parity check: https://en.wikipedia.org/wiki/Parity_bit

Action Items

1. lab2_adv_1.v (25%)

Design a 12-bit hamming code Single Error Correcting decoder that can find no or 1 error bit on given inputs:



a. IO List and Specification

Input Signals	Bit Width
clk	1
rst_n	1
code	12

Output Signals	Bit Width
out	4
raw_data	8
err	1
cor	1

- clk:
  - A 100Mhz clock.
  - All Outputs should change at positive edges.
- rst_n:
  - Asynchronous negative reset; when rst_n == 1'b0, reset all outputs to 0;
- code:
  - A 12-bit data encoded using Hamming code.
- out:
  - Output the index of the error bit in binary.
  - Output 0 if all bits are correct or multiple errors are detected.
    (This special situation will be explained below)
- raw_data:
  - Output the 8-bit raw data without any bit error.
  - Output 0 if multiple errors are detected.
- err:
  - Output 1 if multiple errors are detected, 0 otherwise.
- cor:
  - Output 1 if no error bits, 0 otherwise.

b. Rules of Hamming Code Single Error Correcting (SEC) encoding & decoding:

In lab2_adv_1, we use Single Error Correcting (SEC)Code, which means we can correct 1-bit error from the encoded data.

a. Encoding rules:

As in the following table, the Hamming code adds 4 redundant bits to the 8-bit data, where p1, p2, p4, p8 are the parity bits and d0, d1, d2, d3, d4, d5, d6, d7 are the data bits.

Bit	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
Notation	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8

- 4 parity bits are added as the following equations:

  - $p1(B1) = B3 \oplus B5 \oplus B7 \oplus B9 \oplus B11$
  - $p1(B2) = B3 \oplus B6 \oplus B7 \oplus B10 \oplus B11$
  - $p1(B4) = B5 \oplus B6 \oplus B7 \oplus B12$
  - $p1(B8) = B9 \oplus B10 \oplus B11 \oplus B12$

For example, an input data 01100101_2 can be encoded into:

Bit	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
Notation	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
data	1	0	0	1	1	1	0	0	0	1	0	1

- p1, p2, p4, and p8 are calculated as follow:

  - $p1(B1) = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 1$
  - $p1(B2) = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$
  - $p1(B4) = 1 \oplus 1 \oplus 0 \oplus 1 = 1$
  - $p1(B8) = 0 \oplus 1 \oplus 0 \oplus 1 = 0$

b. Decoding rules:

In the decoding process, we calculate 4 1-bit data, named h1, h2, h4, h8, to check the correctness of the data, and find out the error bit if there exists a 1-bit error.

h1, h2, h4, h8 are calculated as the following equations:

- $h1 = B1 \oplus B3 \oplus B5 \oplus B7 \oplus B9 \oplus B11$
- $h2 = B2 \oplus B3 \oplus B6 \oplus B7 \oplus B10 \oplus B11$
- $h4 = B4 \oplus B5 \oplus B6 \oplus B7 \oplus B12$
- $h8 = B8 \oplus B9 \oplus B10 \oplus B11 \oplus B12$

H=h8 h4 h2 h1, which represents the error bit index.

For example, we received a 12-bit Hamming code encoded data 100111001101_2:

Bit	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
data	1	0	0	1	1	1	0	0	1	1	0	1

- $h1 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$
- $h2 = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$
- $h4 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$
- $h8 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$

So H=1001, which indicates that the 9th bit is wrong, and the correct raw data is 01100101_2.

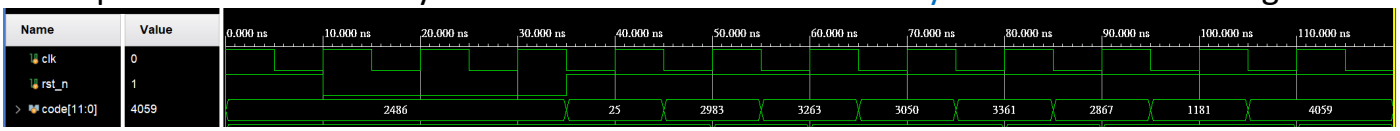
c. Multiple errors detected:

This mechanism can only correct 1 bit error. However, since the encoded data only has 12 bits, that means if H is bigger than the length of the data, we can assume that the encoded data has more than 1 bit that have been corrupted during the transmission.

2. lab2_adv_1_t.v (20%)

Write a testbench (lab2_adv_1_t.v) by yourself to verify the design. In your report, please explain how you built your testbench.

The input of the module in your testbench should look exactly the same as this image:



Here are some brief text description of the image above:

- The duration of the simulation is 120ns.
- rst_n signal should fall at 10ns and last for 25 ns.
- There are 8 data that you should test during the simulation, the value of these data are provided in the template.
- Start testing the first data when rst_n rises(at 35ns).
- Change the data of the input 'code' at negative edges.
- Remember that all outputs change their value at positive edges. That is, the value of 'code' changes at negative edges, but 'out', 'raw_data', 'err', and 'cor' should update at positive edges.

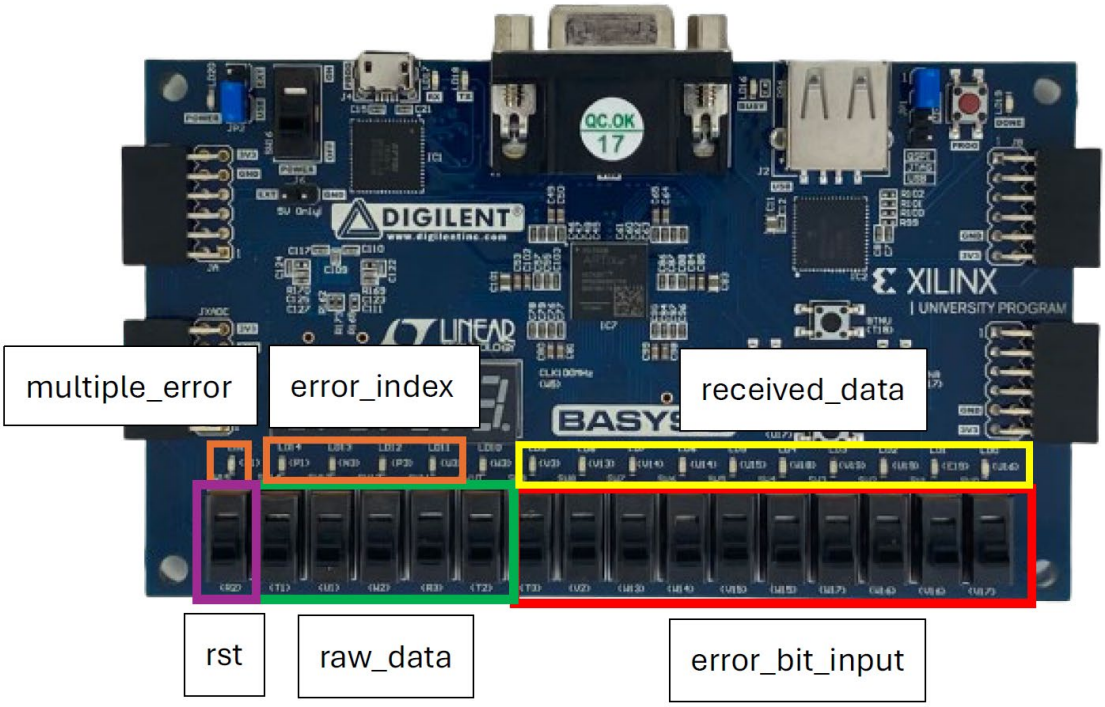
3. lab2_adv_2.v (55%)

Design a 5-bit Hamming codes Single Error Correcting, Double Error Detecting encoder & decoder on FPGA, which uses switches to input data, and leds to output data.

a. IO List and Specification

Input Connection	
rst	connected to SW15
raw_data	connected to SW14-SW10
error_bit_input	connected to SW9-SW0

Output Connection	
received_data	connected to LD9-LD0
error_index	connected to LD14-LD11
multiple_error	connected to LD15



- rst:
  - Asynchronous positive reset; when rst == 1'b1, reset all outputs to 0.
- raw_data:
  - A 5-bit binary raw data, waiting to be encoded.
- error_bit_input:
  - Inputs corresponding to the bits that are corrupted during the data transmission. For example, if SW7 is set to 1, that means the 3th bit of the encoded data is corrupted and should be flipped.
  - Counting from left to right, that is, SW9 is the first bit, while SW0 is the 10th bit.

- received_data:
  - After encoding the 5-bit raw data from the input, outputs the 10-bit encoded data with corruption (if any) onto the LEDs.
  - Counting from left to right, that is, LD9 is the first bit, while LD0 is the 10th bit.
- error_index:
  - After decoding, show the error bit index on LEDs in binary. For example, if the 7th bit is incorrect, LD14-LD11 should be 0111.
  - Output 0 if multiple_error == 1.
- multiple_error:
  - Output 1 if any double-bit error, which is detected by pn, or multiple errors, which is the case we mentioned in lab2_1_adv, are detected. 0 otherwise.

b. Rules of Hamming Code Single Error Correcting, Double Error Detecting (SECDED) encoding & decoding:

In lab2_adv_2, we will use Single Error Correcting, Double Error Detecting (SECDED) code to detect if there is a double number of wrong bits.

1. Encoding rules:

Based on the rule of SEC code, we add an additional even parity bit, named pn, for the whole word. For example, an input data 10100_2 can be encoded into:

Bit	B1	B2	B3	B4	B5	B6	B7	B8	B9
Notation	p1	p2	d1	p4	d2	d3	d4	p8	d5
data	1	0	1	1	0	1	0	0	0

and we add pn, which should be 0, so the whole word has even 1's.
Now the 10-bit data is:

Bit	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
Notation	p1	p2	d1	p4	d2	d3	d4	p8	d5	pn
data	1	0	1	1	0	1	0	0	0	0

2. Decoding rules:

In the decoding process, we calculate h1, h2, h4, h8, to find out the error bit if there exists an 1-bit error. Additionally, we calculate hn to detect double errors.

h1, h2, h4, h8, and hn are calculated as the following equations:

- $h1 = B1 \oplus B3 \oplus B5 \oplus B7 \oplus B9$
- $h2 = B2 \oplus B3 \oplus B6 \oplus B7$
- $h4 = B4 \oplus B5 \oplus B6 \oplus B7$
- $h8 = B8 \oplus B9$
- $hn = B1 \oplus B2 \oplus \ldots \oplus B9 \oplus B10$

H=h8 h4 h2 h1, which represents the error bit index.
There are 4 cases, representing different types of error:

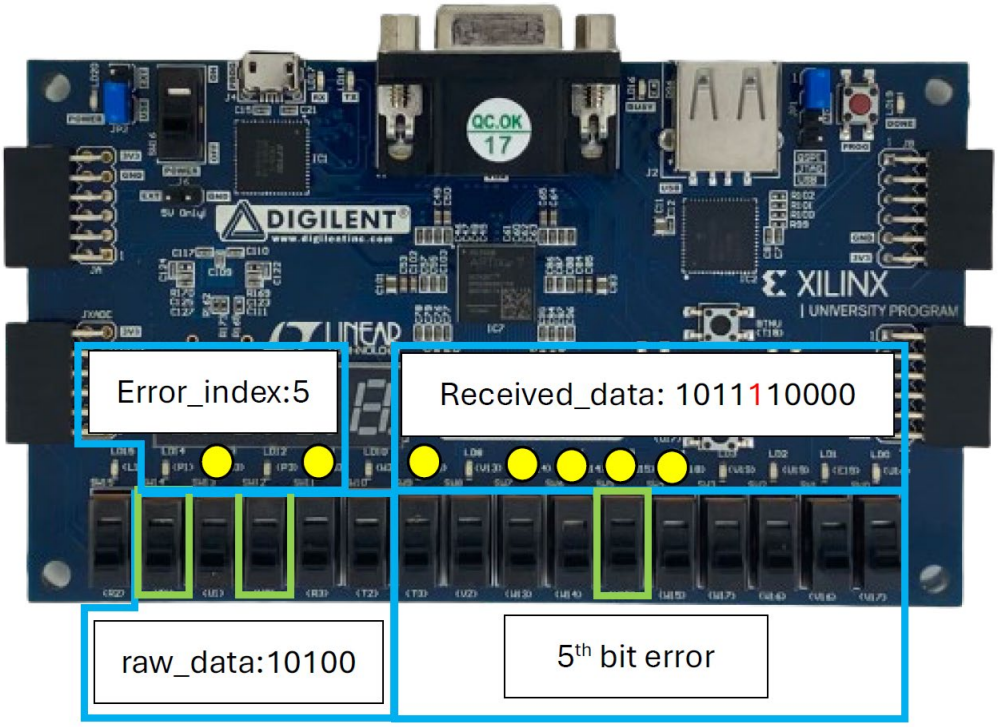
i. $H = 0, hn = 0$: no error.
ii. $H = 0, hn \neq 0$: error at pn (10^{th} bit).
iii. $H \neq 0, hn \neq 0$: error at H^{th} bit.
iv. $H \neq 0, hn = 0$: double bits error.

Same as in lab2_adv_1, if H is bigger than the length of the data , we can assume that the encoded data has more than 1 bit that have been corrupted during the transmission.

And here are the rules of the two outputs, “error_index” and “multiple_error”:

Case	error_index	multiple_error
No error	0	0
Single bit error at 1^{st}~10^{th} bit	Index of the error bit	0
More than 1 bit error	0	1

For example, if the raw data is 10100_2, and the received data is 1011110000_2:
H = h8 h4 h2 h1 = 0101
hn = 1 (there are odd 1’s in the 10-bit data)
So the 5th bit is incorrect.
Here’s what the FPGA board should look like in this case:



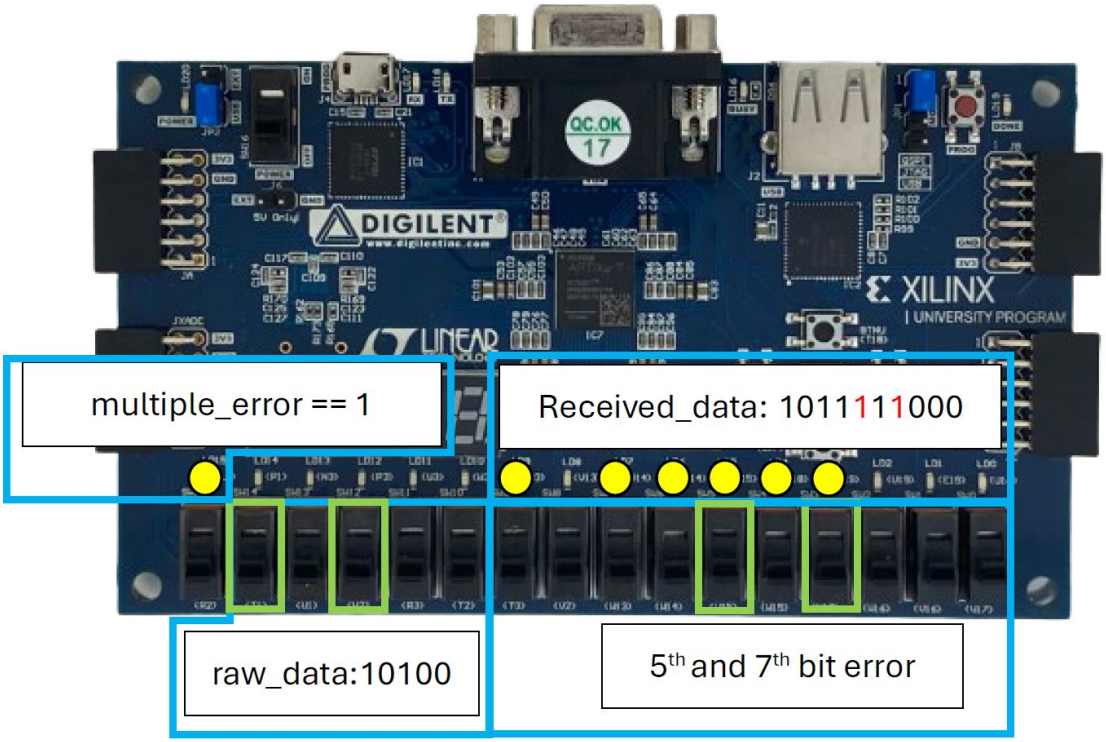
Here's the same example with 2 bits error, the received data is 1011111000_2:

H = 0010

hn = 0 (there are even1's in the 10-bit data)

Since H≠0 and hn≠1, we detect double bits error.

Here's what the FPGA board should look like in this case:



4. Questions and Discussion

Please answer the following questions in your report.

1. What are the differences between a combinational circuit and a sequential circuit? Please explain how each of them works in detail.

2. We use even parity to construct the error correction code in lab2_adv_1. If we use odd parity to encode the 8-bit raw data into a 12-bit data, how can you modify your Verilog code to check errors? Briefly explain in words (you can add pictures & parts of your code with description to describe if you want, but please do not paste a screenshot of all the modified code on your report).

3. In lab2_adv_2, please briefly explain why you shouldn't direct wire h1, h2, h4 and h8 to LD14~LD11.

5. Report Guidelines

Your report should include but not limit to the following items.

A. Lab Implementation (35%)

1. Draw the block diagram of the design with explanation;
2. Explain each essential block;
3. Draw the state diagram(s) for the FSM(s) with the explanation (if any);
4. Explain how you test your design: waveform screenshot.

Use drawing tool(s) for block and state diagrams, e.g., Draw.io <https://draw.io/> or MS PowerPoint. Do not draw them by hand or using sketching/note-taking tools.

B. Questions and Discussions (50%)

Provide your answer to the Questions and Discussions in the lab assignment.

C. Problem Encountered (10%)

Describe the problems you encountered, solutions you developed, and the discussion. Explaining them with code segments or diagrams is recommended.

D. Suggestions (5%)

Any suggestions for this course are more than welcome.
(If not, you may also post a joke (a funny one, please). It is not mandatory and has nothing to do with the grading. But it would undoubtedly amuse us.)

Please also read the guidelines in the report template.

Attention

- ✔ DO NOT copy-and-paste code segments from the PDF materials to compose your design. It may also paste invisible non-ASCII characters, leading to hard-to-debug syntax errors.
- ✔ You should hand in three source files, including **lab2_adv_1.v, lab2_adv_1_t.v, lab2_adv_2.v**. Upload each source file individually. DO NOT hand in any compressed ZIP files, which will be considered an incorrect format.
- ✔ You should also hand in your report as **lab2_adv_report_StudentID.pdf** (i.e., lab2_adv_report_111456789.pdf).
- ✔ You should be able to answer questions from TA during the demo.
- ✔ You need to prepare the bitstream files before the lab demo to make the demo process smooth.