LAB 7 // Software
HAMMING CODE

INSTRUCTIONS and RUBRIC

CSE 2301 – Fall 2024

INTRODUCTION

Prerequisites

Though it is not a part of this lab, timing diagrams will be among your questions for the report.

Objectives

The purpose of this assignment is to introduce you to error detecting and error correcting codes in the context of data reliability. You will be working again with LogicWorks symbols to create two separate circuits: one that generates parity bits, and one that checks for and fixes single errors.

Background

Recall that in a parity check, extra "parity" bits are added in certain locations to the existing data bits in order to create redundancy, and ultimately ensure that errors can fixed, or at least detected. For the sake of this assignment, we will be using EVEN parity and the following single-error correction scheme…

Bit position	**1**	**2**	**3**	**4**	**5**	**6**	**7**
ID	P 1	P 2	D 8	P 3	D 4	D 2	D 1

Parity Bit	***Formula***
P1	$Checks\ D8, D4, D1$
P2	$Checks\ D8, D2, D1$
P3	$Checks\ D4, D2, D1$

Check Code	***Formula***
A	$Checks\ D8, D4, D1, P1$
B	$Checks\ D8, D2, D1, P2$
C	$Checks\ D4, D2, D1, P3$

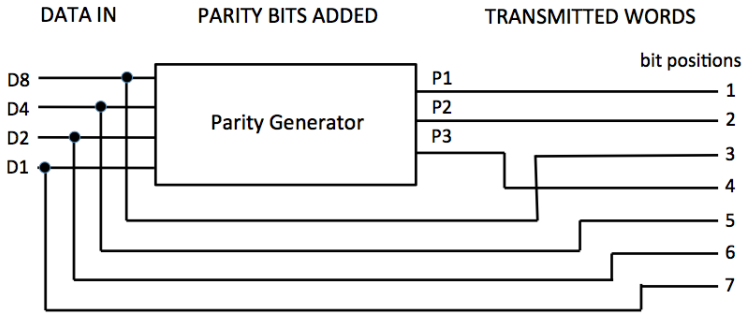
In this case, when generating the parity bits, the P1 bit checks position 3, 5, and 7, and if the sum of these bits is not even, then P1 will become 1, making it even. The same is true for P2 for positions 3, 6, and 7. For P3 these positions are 5, 6, and 7. When checking for errors, the

aforementioned bits are checked in addition to the parity bit. If the sum of these numbers is not even, as it should be, then an error must have occurred.
The full check code tells you where that error is. Specifically, given CBA, weighted 4-2-1, the bit position of the error is provided, and can be flipped, resolving the issue. It will be up to you to decide how you want to check for evenness, and subsequently build the circuits that generate parity bits, indicate error positions, and then fix the relevant line.

PART 1:

Parity Bit Generator

YOUR TASK – Create a circuit that takes four inputs (D8, D4, D2, D1), presumably from binary switches, and generates the 3 parity bits for this code. Your final design should look like the encoding block below. The parity generator part of the circuit should be wired in a symbolic subcircuit much like your 2's Complement converter was last lab. Refer to those instructions for a reminder on the process. Demo with part 2.



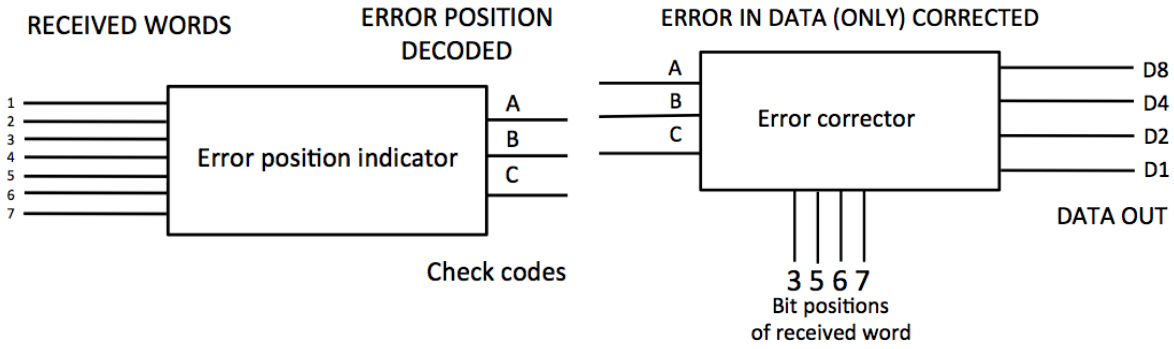
> HINT: If you need to XOR three or more variables, $X \oplus (Y \oplus Z) \equiv X \oplus Y \oplus Z$. This means that you can chain multiple XOR gates together to get the same result as, say, a 3-input XOR.

PART 2:

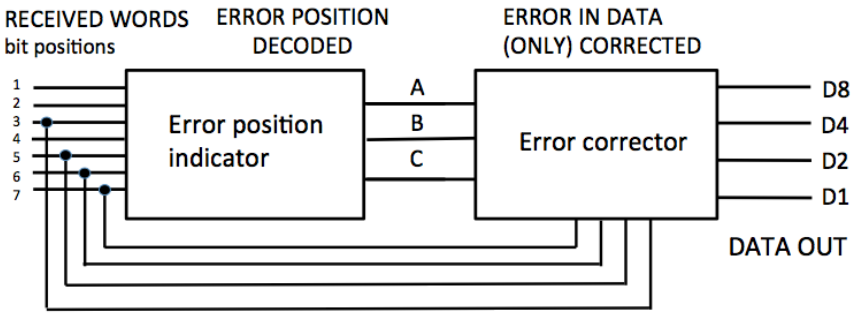
Error Indicator/Corrector

YOUR TASK – Now, develop two more modules capable of detecting and subsequently resolving errors in a 7-bit code, much like the one generated in part 1. You should generate each of these bits by a binary switch.

For the first block, the output will be the check code CBA (which indicate the position of the error). If there is no error, CBA should equal **000**. The corrector will take as input the ABC check code, as well as bits 3, 5, 6, and 7 (D8, D4, D2, D1) from the received 7-bit code, and output the corrected data bits, if any are in need of correction.



Finally, wire these two symbols together to create a complete circuit that both detects and corrects the error. Check you implementation by giving a correct code, then changing just one bit. Regardless of which bit you flip, the output should stay static. Demo with part 1.



> NOTE: The error checking code is also capable of fixing erroneous parity bits, but for our purposes, you do not need to do so. Additionally, be sure to give the inputs and outputs of your subcircuit symbols different names, as the software will get confused otherwise.

SCORING

This is a shorter lab report, but to compensate, you will be given more practice questions, some of which cover timing diagrams, a topic not explicitly part of any lab, but one which is important for the exam.

Demo Requirements [5 pts]:

- ✓ Present your parity bit generator circuit. Also show and explain to TAs the inner workings of the symbol you developed.
- ✓ Present your combined error position indicator and corrector circuit. Similarly present the process you used to design the error symbols.

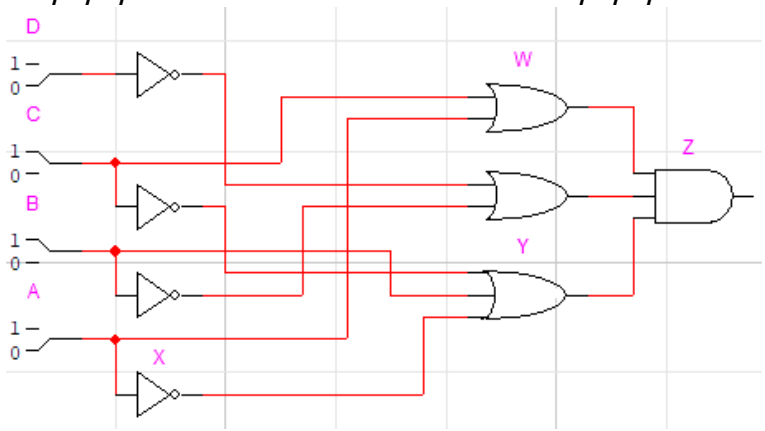
Report Requirements [2 pts]:

- ✓ [0.8] Theory:
  - [0.4] What do we mean by hamming distance?
  - [0.4] What logic needs to be added to detect double errors?

- ✓ [0.8] Deliverables:
  - [0.8] Develop a graph of the number of parity bits needed for the range of **data** (non-parity) bits from 4 to 67. This will be a non-linear step graph. Make sure to label your steps. Start by identifying the pattern, but do not ask TAs for the formula. *HINT: You know that the first 3 parity bits are located at positions 1, 2, and 4. If you continue adding data bits, the fourth parity bit will lie at position 8. What does this tell you?*
- ✓ [0.4] Discussion section. Should conform to standard lab report guidelines.

Practice Questions [3 pts]:

- ✓ [0.8] *Question 1:*
  The circuit below consists of gates in which each INV/NOT has a delay of 3 ns. The OR and AND gates have 5 ns delays. Given an initial condition of A=0, B=1, C=0, and D=0, complete a timing diagram where A becomes 1 at 4 ns. Assume **transport** delay, and that the initial W, X, Y, and Z values are based on initial A, B, C, and D conditions.



- ✓ [0.6] *Question 2:*
  Complete the Hamming code below for the given 4-bit hex values. Insert the **ODD** parity bits into positions 1, 2, and 4. For this problem, **P means POSITION,** not parity.

  The parity bit in P1 checks bits P1, P3, P5, and P7.
  The parity bit in P2 checks bits P2, P3, P6, and P7.
  The parity bit in P4 checks bits P4, P5, P6, and P7.

Hex	P1	P2	P3	P4	P5	P6	P7
0:			0		0	0	0
6:			0		1	1	0
B:			?		?	?	?

- ✓ [0.4] *Question 3:*
  Write a logic expression that generates an **ODD** parity bit from four data bits A, B, C, and D. Optimal solutions only require XOR and NOT gates.

✓ [0.5] *Question 4:*
There is an error in the following 7-bit Hamming code with **EVEN** parity bits.

P1	P2	P3	P4	P5	P6	P7
1	0	0	0	1	1	1

a) What is the position of the erroneous bit? (0.3 pts)
b) What is the correct code? (0.2 pts)

✓ [0.7] *Question 5:*
Find the values for all the check/parity bits in the following **ODD** triangular code.

1 0 0 0 0 1 C_1	C_1= ?
0 1 0 1 1 C_2	C_2= ?
1 0 1 0 C_3	C_3= ?
1 1 0 C_4	C_4= ?
0 1 C_5	C_5= ?
0 C_6	C_6= ?
C_7	C_7= ?