Hamming Code

Prerequisites

To enable you to complete this lab assignment you need to be able to design combinational logic circuits and be familiar with the use of XOR gates for parity bit generation.

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 base your understanding on the concept of parity checks where an extra bit is added to a binary word to make the number of 1's or 0's Even (or Odd). The extra added bits create redundancy in the transmitted words. You will design and build a full Hamming code, for the KB Secret Agency, that allows error detection and error correction of single errors.

Theory

Error Detection

Consider an example using the BCD code. If any of the bits in the 4-bit binary word 0001 were changed, as a result of an error condition, then we could not detect this, since another valid binary word would have been created, viz. 0000 or 0011 or 0101 or 1001. By adding an extra bit to make the number of 1's in the word even, then a single bit in error can be detected by counting the number of 1's. This extra added bit is called the parity bit.

For example, if we want to transmit the two consecutive ASCII characters '**KB**', we would take the 7-bit codes from the binary ASCII table and add an extra bit to make the number of 1's in each word **even**. For example,

Letter	7 Bit ASCII Code	8 Bit ASCII with 1 Parity Bit
K	1001011	10010110
B	1000010	10000100

If the 8-bit ASCII coded concatenated words (KB) 1001011010000100 are transmitted and received as 1001011010100100, then the number of 1's are counted in each byte to check for evenness.

K = 10010110 has an EVEN number of 1's
B = 10100100 has an ODD number of 1's

Therefore, we know that an error exists in the second ASCII character. We cannot however, with a simple parity check digit like this, correct an error.
The principle of adding extra bits is known as adding redundancy which is defined as the total number of bits used divided by the data bits. If the number of data bits is (n-1) then, for a single parity system, redundancy is n/(n-1). For low redundancy we want to use long messages but for high reliability short messages are better.

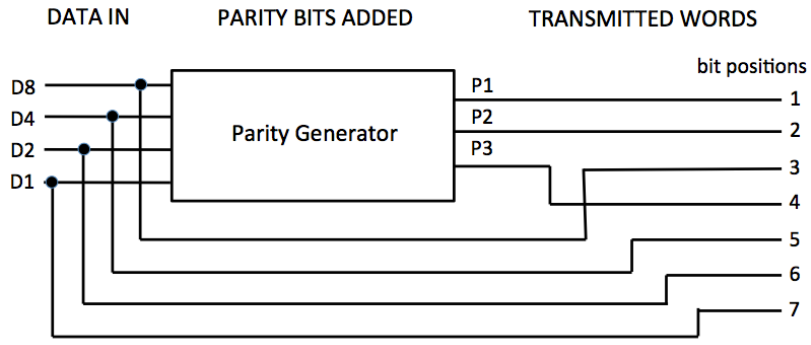
Procedure

The specific Hamming code we are to use is given below. The first step is to check that this table is correct before starting. See 'Encoding Process' below.

1	2	3	4	5	6	7	bit position
P1	P2	D8	P3	D4	D2	D1	
							decimal
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1
0	1	0	1	0	1	0	2
1	0	0	0	0	1	1	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	5
1	1	0	0	1	1	0	6
0	0	0	1	1	1	1	7
1	1	1	0	0	0	0	8
0	0	1	1	0	0	1	9

Encoding Process

A block diagram of the encoding process is given below.



The Parity Generator in this diagram is responsible for calculating parity bits P1, P2, and P3. The value for each parity bit is based on a certain combination of data bit values. This combination is shown for each of the parity bits in the table below.

For example, the first parity bit, P1, is generated by counting the 1's in data bits D8, D4, and D1 and adding a 1 if the number of 1's within these three positions is odd or a 0 if the number is even (we are using even parity).

A similar process occurs for parity check bits P2 and P3 using the appropriate data bits as indicated in the table below. The check codes in the rightmost column are used to locate an error in the decoding process.

Parity Bit	Based on Data Bits	Check Code
P1	D8, D4, D1	C
P2	D8, D2, D1	B
P3	D4, D2, D1	A

To generate the even parity bit for three inputs X, Y and Z consider the following truth table.

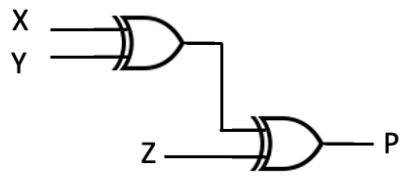
3 data bits			Even parity bit
X	Y	Z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$P = X'Y'Z + X'YZ' + XY'Z' + XYZ$

Using Boolean algebra, the above logic expression can be simplified as bellow.

$P = X'[Y'Z + YZ'] + X[Y'Z' + YZ]$
$= X'[Y \oplus Z] + X[Y \oplus Z]'$
$= X \oplus Y \oplus Z$

There for a logic circuit for generating an even parity bit for three inputs X, Y, Z is as follows.



When each of the parity bits are calculated, the data is coded into a 7-bit word and the 7-bit word is then transmitted in the following order:

Bit Position	1	2	3	4	5	6	7
Parity/Data Bits	P1	P2	D8	P3	D4	D2	D1

as shown in the outputs of the block diagram.

Decoding Process

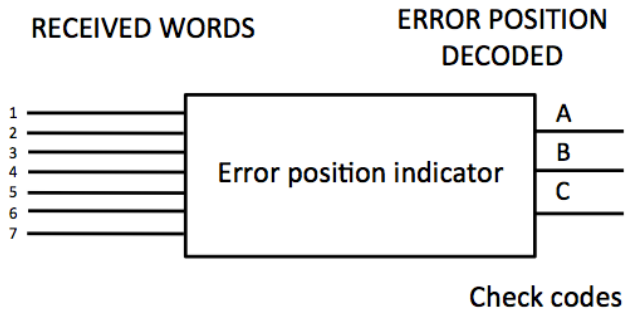
The decoding process involves performing a validity check on the received word, and executing an error correction technique if an error was detected. To check whether an error occurred in the transmission process, the parity checks are performed on the received word, in reverse order.

Do we have even parity checks as defined in the encoding or, in other words, does P1 XOR D8 XOR D4 XOR D1 give a 0 (correct) or a 1 (incorrect)?

Each of the checks contributes to the location of the error itself. If we evaluate the binary order of the checks in the sequence ABC, where A, B, and C are defined in the table above, this will give the position of the error. For example, if ABC = 001 this indicates an error in bit position 1 which correlates to parity bit P1 in the bit position table. If ABC = 101 this indicates an error in bit position 5 which is data bit D4. If ABC = 000 then there is no error.

a) Design and build Error Position Indicator module

The next step is to design and create the LogicWorks circuitry which locates the error position. A block diagram of this module is given below. The seven inputs to the Error Position Indicator module represent the seven bits of the received word. The error position is indicated by the 3-bit output which corresponds to the binary value of the bit position of the bit in error. The outputs A, B, and C, represent the check codes listed in the parity table as described in the decoding process.

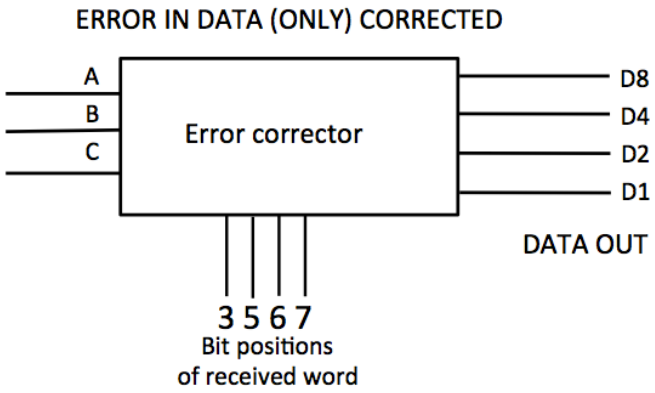


b) Design and build Error Corrector module

The following step is to design and build the Error Corrector module which corrects an error in the received word based on the output from the Error Position Indicator module. The Error Corrector module enables the correct data to be output only if there is no error or a single error in the received data.

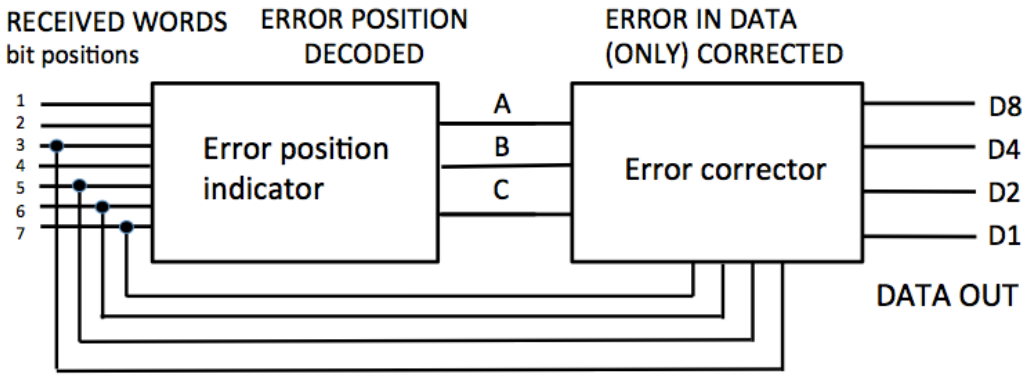
The inputs to this module are the check codes (A, B, and C) generated from your Error Position Indicator module, and the data bits located at positions 3, 5, 6, and 7 in the received word. The four outputs of this module are D1, D2, D4, and D8 which represent the 4 data bits with no errors. A block diagram of this module is given below.

You will have noticed that the same logic circuitry is used for both code encoding as error correcting.



c) Link modules together

Connect your two modules together and test them to see if they are working correctly. A block diagram of the two joined modules is given below:



Testing

You should test your overall system by introducing an error in each of the bit positions (1 to 7), one at a time, and checking that the data out (D8 to D1) is correct.

Deliverables

You need to have your circuit thoroughly checked by your TA. You should be able to demonstrate that an error in any of the received data bits can be corrected and that an error in any of the parity bits does not affect the data bits. You need to submit an image of your circuit and a short report answering the following questions.

Remember that what you have designed only works correctly for a single error. More redundant bits need to be added to, at least, detect a double error.

Your short report should only include

1. a graph of the number of parity bits required for the range of data bits from 4

to 67. (It will not be linear).
2. What logic needs to be added to detect double errors.
3. What do we mean by Hamming distance?