# Linear Block Code Error Correction

Devin Trejo
devin.trejo@temple.edu

March 25, 2016

# 1 Summary

# 2 Introduction

## 2.1 Hamming Linear Block Encoding/Decoding Theory

The Hamming linear block code is a forward error detection and correction method where the original message is encoded to include more symbols to add redundancy to the original message. The process allows the receiver of the message to check and attempt to correct errors introduced by a noisy communications channel. In this lab we will use a (6,3) Hamming code generator matrix (G) which we will transform a 3-bit messages ($\vec{m}$) into 6-bit codewords ($\vec{x}$).

$$\vec{x} = \vec{p}G \text{ where,} \tag{1}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

On the receiving side we first take our received codeword ($\vec{r}$) and detect any errors in the received message. We do so by taking our codeword and checking it against a parity matrix (H). The resulting syndrome ($\vec{z}$) will point to the location of any errors in our transmitted data. Note that our linear block code produces a non-systematic codeword; meaning our message is not separated into k message bits and (n-k) parity bits. [1] Instead our message is embedded within our error-correcting code to produce a unique codeword.

$$\vec{z} = \vec{r}H \text{ where,} \tag{2}$$

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

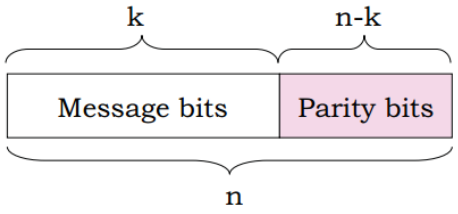


Figure 1: Example of Systematic Encoding Formating [1]

The generator matrix (G) is constructed from an identity matrix of size $k \times k$ concatenated with a matrix A of size $k \times (n-k)$ to the left. If we take matrix A we can create our parity matrix (H) by concatenating it with a identity matrix of size $(n-k) \times (n-k)$ on the bottom.

$$G_{kxn} = \begin{bmatrix} I_{k\times k} & | & A_{k\times(n-k)} \end{bmatrix} \quad H_{nxk} = \begin{bmatrix} A_{k\times(n-k)} \\ I_{(n-k)\times(n-k)} \end{bmatrix}$$

The error detector referenced by equation 2 is said to detect an error when the syndrome ($\vec{z}$) in non-zero. To correct any 1-bit errors seen in the 6-bit received message requires we flip the bit seen in the position referenced by the non-zero syndrome. For example, if $\vec{z} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$ then we say corresponding bit 2 in $\vec{r}$ needs to be flipped.

The error correction works by looking at the **Hamming Distance** between two vectors $\vec{v_1}$ and $\vec{v_2}$. From our encoding technique we have a finite set of acceptable codewords possible. If the received codeword ($\vec{r}$) is not in the set of possible codewords we know we have an error. To correct the received code word we look at the Hamming Distance between the received codeword ($\vec{r}$) and all codewords in the set of possible codewords. We correct the received codeword ($\vec{r}$ to a valid codeword which has a minimum Hamming distance. [2]

After we have our codeword corrected ($\vec{r'}$), we can retrieve the original message ($\vec{p_r}$) by using a (3,6) decoding matrix (R). The decoding matrix reverses the steps taken by the generator matrix (G).

$$\vec{p_r} = \vec{r'}R \text{ where,} \tag{3}$$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Finally we wish to look at the performance of this Hamming Linear Block code encoder/decoder. The correction capacity for any given codeword is given by equation 4 where t denotes the number of errors that are detectable and correctable. In some instances the receiver may not be able to correct a received codeword but it is still detectable. The error detection (e) is given by equation 5.

$$t = \lfloor \frac{d_{min} - 1}{2} \rfloor \tag{4}$$

$$e = d_{min} - 1 \tag{5}$$

## 2.2 Hamming Linear Block Encoding/Decoding Implementation

For this lab we will transmit the message *"EE is my avocation"* and set up a communication channel between a PIC32 Microchip MCU and Windows desktop over Ethernet. The Windows machine will run a Visual Basic application which will receive a message from the PIC32 server and introduce random errors to simluate errors that may be introduced when transmitting over a noisy communications channel. VB client *Client2 v16B* will introduce at most 1-bit in error within a given message. VB Client *Client3 v16* will introduce up to 10-bits in error. After we will setup our PIC32 MCU to receive a message from the VB client in order to perform error detection and correction using the Hamming linear block scheme.

To use a (6,3) Hamming code requires our individual messages to be 3-bits in size. Therefore, we format our transmission string into messages 3-bits in size. We then will encode our message to produce 6-bit codewords. Since our message is using characters within the standard ASCII table, the MSB in all characters will always be zero. Therefore, ignoring the MSB can now take our 7-bit ASCII character and break it up 3 individual bits. A combination of 3 characters will end up creating a perfect set to complete 7 packets we can send.

$$\begin{array}{rcccccc} 'E' & = & [0100 & 0101] & \rightarrow & [100 & 0101] \\ 'E' & = & [0100 & 0101] & \rightarrow & [100 & 0101] \\ '\text{␣}' & = & [0100 & 0000] & \rightarrow & [100 & 0000] \end{array}$$

| 100 | 010 | 110 | 001 | 011 | 000 | 000 |
|---|---|---|---|---|---|---|
| $\vec{p_0}$ | $\vec{p_1}$ | $\vec{p_2}$ | $\vec{p_3}$ | $\vec{p_4}$ | $\vec{p_5}$ | $\vec{p_6}$ |

Table 1: 3-Bit Message Composition for First 3 Characters

Our overall message is 18 characters in length. Distributing our 18 characters into 7 packets of length 3-bits produces 42 packets. Those 42 packets will next be encoded using our (6,3) Hamming linear block encoder to produce packets of 6-bits. Finally, we will pad each 6-bit packet with two leading zeros and send it as a 8-bit ASCII character. The final transmitted packet sequence for the first 3 characters are shown in table 2 by applying equation 1.

| 0010 0110 | 0001 0111 | 0011 0001 | 0000 1101 | 0001 1010 | 0010 0110 | 0000 0000 |
|---|---|---|---|---|---|---|
| $\vec{x_0}$ | $\vec{x_1}$ | $\vec{x_2}$ | $\vec{x_3}$ | $\vec{x_4}$ | $\vec{x_5}$ | $\vec{x_6}$ |

Table 2: 6-Bit Codewords Composition for First 3 Characters

# 3 Discussion

# References


[1] Hari Balakrishnan and George Verghese. Introduction to EECS II: Digital Communication Systems, 2010.

[2] William Stallings. *Data and Computer Communications*. Pearson Education, Saddle River, tenth edition, 2014.


# Appendix

Main program source code available on my GitHub:
`https://github.com/dtrejod/myece4532/blob/master/lab4/ECE4532%20PIC32%20BSD%20Server/source/main.c`