Codificación Digital

Luís Miguel Capacho V. capacho4@hotmail.com, Nelson Antonio Becerra C. nelsonabc25@hotmail.com, Jaime Alberto López R. jaimealopezr@yahoo.com, Diego Felipe García. felipeg_84@hotmail.com.
Programa de Ingeniería Electrónica, Universidad del Quindío

Resumen— En este documento se pretende profundizar en el análisis de los métodos de codificación de señales digitales, es decir, en la codificación de línea y de bloques; para ello se implementó un software en Builder C++, que permite generar un dato aleatorio de 8 bits, el cual es transmitido serialmente a un microcontrolador para su correspondiente codificación de línea o de bloque. El código de línea implementado fue el NRZI (no retorno a cero invertido) y el código de bloque implementado fue el de Hamming.

III. INTRODUCCIÓN

Una señal digital es una secuencia de pulsos de tensión discretos y discontinuos. Cada pulso es un elemento de la señal. Los datos binarios se transmiten codificando cada bit en los elementos de señal. En el caso más sencillo, habrá una correspondencia uno a uno entre los bits y dichos elementos.

Un factor importante que se utiliza para mejorar las prestaciones de un sistema es el esquema de codificación, el cual es simplemente la correspondencia que se establece entre los bits de los datos con los elementos de la señal.

Se han intentado una gran variedad de aproximaciones para la codificación de señales digitales, estas aproximaciones se agrupan en códigos de línea y códigos de bloque.

Los códigos de líneas más utilizados son: Los no retorno a cero, entre los cuales tenemos NRZ, NRZ-L, NRZI, Los binario multinivel, entre los cuales tenemos el bipolar-AMI y el Pseudoternario y los Bifase: Manchester y Manchester diferencial.

Los códigos de bloque más utilizados son: El Hamming, los códigos cíclicos, el código Reed-Solomon, los códigos convolucionales entre otros.

Codificación NRZI

El NRZI mantiene constante el nivel de tensión la duración del bit. Los datos se codifican mediante la presencia o ausencia de transmisión de la señal. Un 1 se codifica mediante la transición (bajo a alto o alto a bajo) al principio del intervalo de señalización, mientras que un cero se representa por la ausencia de transmisión.

NRZI es un ejemplo de codificación diferencial. La codificación diferencial, en lugar de determinar el valor absoluto, la señal se codifica en función de los cambios entre los elementos de señal adyacentes. En términos generales, la codificación de cada bit se hace de la siguiente manera: si se trata del valor binario 0, se codifica con la misma señal que el bit anterior; si se trata de un valor binario 1, entonces se codifica con una señal diferente que la utilizada para el bit precedente. En la Figura 1 se muestra el formato de codificación para el NRZI.

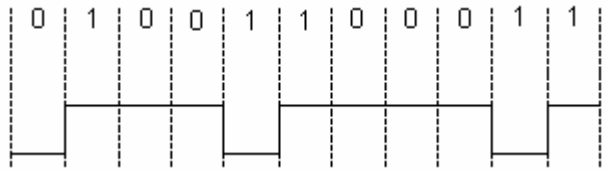

Figura 1. Señal con codificación digital NRZI.

Codificación de Hamming

Este código fue propuesto por R. W. Hamming en 1950 y permite a través de un subconjunto de códigos de control de paridad localizar la presencia de errores dentro del mensaje. Estos códigos tienen como muy poco distancia mínima 3.

El código Hamming es clasificado como un código de bloque por que tiene como entrada un grupo (bloque) de *m* bits de datos a los cuales se le agregan un grupo de *r* bits de parida de acuerdo a reglas preestablecidas, dando como salida un grupo de *n* bits comúnmente llamado palabra-código. Así mismo se dice que es lineal porque satisface la condición de que cualquier par de

palabras-código al ser sumadas en modulo 2 producen otra palabra existente dentro del conjunto empleado para codificar los posibles mensajes.

Un código de Hamming se puede denotar mediante un par (n,m). Sin embargo los valores de n y m deberán verificar una serie de condiciones:

- n es la longitud de la palabra de código
- m es el número de bits de datos de la palabra original sin codificar
- el número de bits de paridad será r=n-m, pero deberá cumplirse la siguiente relación entre la longitud de la palabra de código y el número de bits de paridad: $n=2r-1$ con $r>=3$
- según esto también se cumplirá la siguiente relación entre el número de bits de datos y el número de bits de paridad: $m=2r-r-1$

Por lo tanto, a cada palabra original se le añade unos bits de paridad para obtener la palabra de código, de forma que estos bits de paridad sirvan posteriormente para encontrar y corregir errores que se produzcan en la transmisión.

Para obtener la codificación de cada dato se tiene que cumplir que $\boldsymbol{A}.\boldsymbol{T} = 0$, donde $\boldsymbol{A}$ corresponde a la matriz del código de dimensiones rxn y $\boldsymbol{T}$ a un vector columna del dato codificado de longitud n. Las columnas de la matriz $\boldsymbol{A}$ deben ser todas distintas y no contener el vector 0.
Tomando como un ejemplo un dato de 4 bits (m=4), 3 bits de error (r=3) se tienen la siguiente matriz A:

$$A = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}}_{\text{n columnas = 7}} \} \text{ r filas = 3}$$

Nótese que los valores de cada columna se construyen asignando los números en binario desde el 1 hasta el 7.
El siguiente paso consiste en nombrar cada una de estas columnas, asignado c_0, c_1 y c_2 para aquellas que sólo tienen un 1 y m_1, m_2, m_3 y m_4 para las demás, como se muestra a continuación:

$$A = \begin{bmatrix} \overset{c_0}{0} & \overset{c_1}{0} & \overset{m_1}{0} & \overset{c_2}{1} & \overset{m_2}{1} & \overset{m_3}{1} & \overset{m_4}{1} \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

A partir de esta última se construye el sistema lineal de ecuaciones para cada una de las filas de acuerdo a aquellos coeficientes que son 1:

$$c_2 \oplus m_2 \oplus m_3 \oplus m_4 = 0$$
$$c_1 \oplus m_1 \oplus m_3 \oplus m_4 = 0$$
$$c_0 \oplus m_1 \oplus m_2 \oplus m_4 = 0$$

Como los valores de m corresponden al mensaje, se despeja cada uno de los bits de comprobación de error. Esto da como resultado el siguiente conjunto de ecuaciones:

$$c_2 = m_2 \oplus m_3 \oplus m_4$$
$$c_1 = m_1 \oplus m_3 \oplus m_4$$
$$c_0 = m_1 \oplus m_2 \oplus m_4$$

Aplicando estas ecuaciones a cada uno de los 16 datos de 4 bits se obtiene la siguiente tabla, la cual es utilizada en el Microcontrolador para generar los datos codificados:

Número	Bits de Error (c_2,c_1,c_0)	Dato Codificado en binario ($c_2,c_1,c_0,m_4,m_3,m_2,m_1$)
0	000	0000000
1	011	0110001
2	101	1010010
3	110	1100011
4	110	1100100
5	101	1010101
6	011	0110110
7	000	0000111
8	111	1111000
9	100	1001001
A	010	0101010
B	001	0011011
C	001	0111100
D	010	0101101
E	100	1001110
F	111	0111111

IV. SISTEMA PROPUESTO

Con el fin de realizar la codificación de un dato aleatorio, por medio del código de línea NRZI y el código de bloque Hamming, proponemos el sistema en diagramas de bloques mostrado en la Figura 2.

Figura 2. Diagrama de bloques codificación digital.

Por medio de un PC vamos a generar el dato aleatorio de 8 bits que deseamos codificar, el

lenguaje de programación y la plataforma que se utilizó es Builder C++ 6.0. Este dato es transmitido serialmente a un microcontrolador que realizará la codificación requerida, ya sea de línea o de bloque, y este entregará simultáneamente el dato y el código por uno de sus puertos para de esta forma lograr visualizar la codificación en un osciloscopio.

La Figura 3 muestra el diagrama electrónico del sistema propuesto, el cual fue diseñado con un microcontrolador Motorota *MC68HC908GP32* para la codificación de las señales binarias que se originan desde el computador. Esta señal binaria debe someterse a una conversión de niveles de voltaje, ya que los niveles con los que transmite el computador no pueden ser asimilados correctamente por el microcontrolador, es decir, estos niveles de voltaje lógicos son diferentes. Por tal motivo es preciso contar con un protocolo que defina los niveles de transmisión de datos, este protocolo es el RS232. El dispositivo que cuenta con este protocolo es el MAX232 como se muestra en la Figura 3.

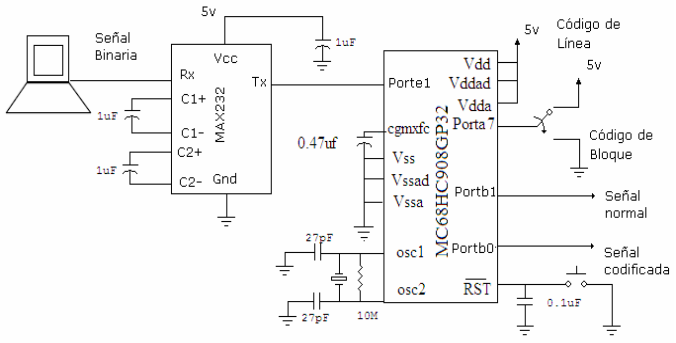

Figura 3. Sistema propuesto para el codificar de línea y bloque.

El microcontrolador es el encargado de codificar la señal proveniente del computador; además este cuenta con un interruptor para seleccionar el tipo de código a realizar. Por otra parte se pueden observar simultáneamente en el osciloscopio la señal aleatoria procedente del computador y la señal codificada por línea o bloque dependiendo de la opción escogida en el interruptor.

Como se mencionó anteriormente este dispositivo puede realizar dos tipos de codificación: *NRZI* y *Hamming*. En cuanto a la codificación de línea *NRZI* se realiza sobre todo el byte transmitido por el computador, ya que esta se hace bit por bit; en cambio el código de bloque da *Hamming* se efectúa sobre el nible menos significativo del byte recibido, debido a que en este tipo de codificación por cada cuatro bits se adicionan tres bits para corrección y detección de errores en el proceso de transmisión.

La ventana principal del software que nos permite generar el dato aleatorio se muestra en la Figura 4.

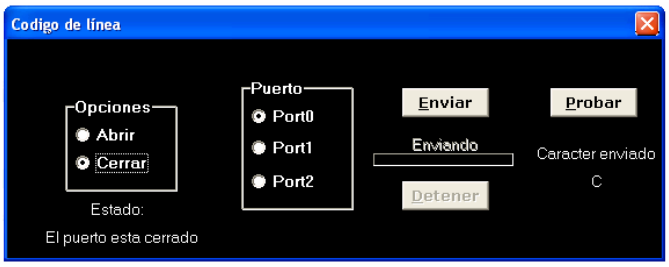

Figura 4. Ventana principal.

Este software además de generar aleatoriamente el dato que se desea codificar, lo envía por el puerto serial al microcontrolador para su codificación. En la Figura 4 se observan las opciones y posibilidades que ofrece dicho software, entre las cuales se encuentran abrir o cerrar el puerto serial, elegir entre el puerto 0, 1 y 2, enviar el dato, detener la transmisión y observar el carácter enviado.

La función que permite generar un dato aleatorio se muestra a continuación:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
 char dato;
 randomize();
 dato=random(255);
 enviar_dato(dato);
 randomize();
 dato=random(255);
 enviar_dato(dato);
}
```

La función enviar_dato envía un dato de 8 bits por el puerto serial a una velocidad de 4800 baudios sin bit de paridad.

El código completo que genera y transmite los datos por el puerto serial es el siguiente:

```
#include <vcl.h>
#pragma hdrstop
#include "main.h"
#include "serialw32.h"
```

```
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
SerialCommDriver Serial;
int Puerto=0;
bool prueba=false;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
 Serial.Close();
 Port=false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AbrirClick(TObject *Sender)
{
 if(Serial.Open(Puerto,4800,8,NOPARITY,0,0))
  {
   Label2->Caption="El puerto esta abierto";
   Port=true;
  }
 else
  {
   Application->MessageBox("No se puede abrir el Puerto!!","Error",
MB_OK);
   Cerrar->Checked=true;
   Port=false;
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::CerrarClick(TObject *Sender)
{
 if(Serial.Close())
  {
   Label2->Caption="El puerto esta cerrado";
   Port=false;
  }
}
//---------------------------------------------------------------------------
void TForm1::enviar_dato(char dato)
{
 Serial.WriteChar(dato);
}

void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
 Puerto=0;
 Port=false;
 Cerrar->Checked=true;
 Label2->Caption="El puerto esta cerrado";
 Application->MessageBox("Abra el puerto!!","Información",
MB_OK);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
 Puerto=1;
 Port=false;
 Cerrar->Checked=true;
 Label2->Caption="El puerto esta cerrado";
 Application->MessageBox("Abra el puerto!!","Información",
MB_OK);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RadioButton3Click(TObject *Sender)
{
 Puerto=2;
 Port=false;
 Cerrar->Checked=true;
 Label2->Caption="El puerto esta cerrado";
 Application->MessageBox("Abra el puerto!!","Información",
MB_OK);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AceptarClick(TObject *Sender)
{
 if(Port)
  {
   Timer1->Enabled=true;
   Timer2->Enabled=true;
   Button1->Enabled=true;
   Label3->Enabled=true;
   Barra->Enabled=true;
  }
 else
  Application->MessageBox("No hay puerto abierto!!","Error",
MB_OK);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
 char dato;
 randomize();
 dato=random(255);
 enviar_dato(dato);
 randomize();
 dato=random(255);
 enviar_dato(dato);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 Timer1->Enabled=false;
 Timer2->Enabled=false;
 Button1->Enabled=false;
 Label3->Enabled=false;
 prueba=false;
 Barra->Enabled=false;
 Barra->Position=Barra->Min;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer2Timer(TObject *Sender)
{
 Barra->Position++;
 if(Barra->Position==Barra->Max)
  Barra->Position=Barra->Min;
 if(prueba)
  {
   char dato;
   dato=recibir_dato();
   Label4->Caption=dato;
```

```
 }
}
//---------------------------------------------------------------------------
char TForm1::recibir_dato(void)
{
 char dato;
 dato=Serial.ReadChar();
 return dato;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 if(Port)
  {
   Timer1->Enabled=true;
   Timer2->Enabled=true;
   Button1->Enabled=true;
   Label3->Enabled=true;
   Barra->Enabled=true;
   prueba=true;
  }
 else
  Application->MessageBox("No hay puerto abierto!!","Error",
MB_OK);
}
```

Implementación en lenguaje Ensamblador

El microcontrolador Motorota *MC68HC908GP32* tenía la tarea de recibir los datos generados aleatoriamente por el computador a través del puerto serial utilizando la interfaz RS232 y ponerlos de manera secuencial en dos de sus pines de salida dispuestos para ello. El primero de estos sería la salida no codificada mientras que el segundo mostraría el código de línea o de bloques del anterior. Esto nos permitiría comparar en el osciloscopio ambas señales.

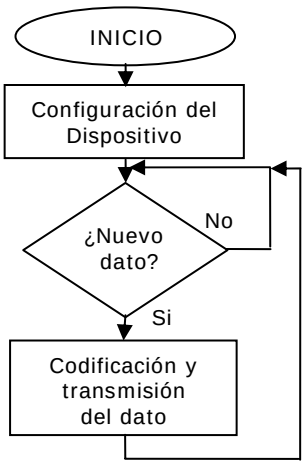

Figura 5. Diagrama de flujo del programa en ensamblador.

La Figura 5 muestra el diagrama de flujo del programa principal. Inicialmente se configura el microcontrolador, esto incluye seleccionar los pines que van a ser utilizados de entrada y salida y la velocidad del puerto serial. El lazo principal espera la transmisión de un dato para codificarlo y transmitirlo. Para determinar si se debe aplicar el código de línea o el código de bloques, se usa un interruptor en el pin 7 del puerto A. El valor de este se verifica en el lazo principal para aplicar la codificación respectiva.

Para cada uno de los métodos de codificación se pone de manera serial el dato transmitido por el PC y el dato codificado. Esto se logra mediante la rotación de bits. La codificación *NRZI* se obtiene al negar el estado anterior del bit 0 del puerto B cada vez que se encuentra un 1 lógico en el dato recibido, mientras que los datos con la codificación de Hamming están guardados en una tabla pre-calculada a la cual se accede según el dato que se vaya a codificar. Cada uno de los bits de la señal de salida tiene una duración de 200ms.

El código implementado en lenguaje ensamblador se muestra a continuación:

```
$include 'gpgtregs.inc'

FLASH      EQU $8000
RAM        EQU $0040
RESET      EQU $FFFE

DATO_RX    EQU $FFE4

************************************
*  Directivas de pre-procesamiento  *
************************************
OUT_N    EQU 0
OUT_C    EQU 1
PORT_LED EQU PORTB
LED      EQU 7
IN_SEL   EQU 7


****************************************
*  Variables utilizadas en el programa  *
****************************************
       ORG RAM
ByteDato  rmb 1
HayByte   rmb 1
ByteRx    rmb 1
Hamming   rmb 1

******************************
*  Memoria Flash del programa  *
******************************
       ORG FLASH
INICIO:
    bset  0,CONFIG1
    ldhx  #$240
    txs

    clra              ; Limpieza de variables
    clrh
```

```
    clrx
    clr   ByteDato
    clr   HayByte
    clr   ByteRx
    clr   Hamming

    bset  LED,DDRB    ; Pines de salida
    bset  LED,PORT_LED
    bset  OUT_N,DDRB
    bset  OUT_C,DDRB

    ; Configuración del SCI
    mov   #$02,CONFIG2
    mov   #%00000101,SCBR     ; Vel = 4800 bps
    mov   #%01000000,SCC1     ;
    mov   #%00100100,SCC2     ; Activa Rx

    lda   SCS1
    clr   SCDR

    cli

**********************
*  Rutina principal  *
**********************
Principal:
    wait
    brclr 0,HayByte,Cod_Bloque
    bclr  0,HayByte   ; Se borra la bandera que
                      ; indica los datos listos

; Selecciona el código
    brset IN_SEL,PORTA,Cod_Linea

****************************************
*  Rutina principal del código de bloque  *
****************************************
Cod_Bloque:
    mov   ByteRx,ByteDato

    lda   ByteDato
    and   #$F         ; Borra la parte alta del dato
    sta   ByteDato

    ldhx  #Tabla      ; Calcula el valor de la tabla
    txa               ; A = X
    add   ByteDato    ; A = X + Muestra
    tax               ; X = ACCA

    pshh
    pula              ; A = H
    adc   #!0         ; A = H + Aux
    psha
    pulh              ; H = A

    mov   x+,Hamming ; Guarda el valor de la tabla

    ldx   #!7

    ; Saca en el puerto el bit codificado
HCiclo:
    brclr 0,Hamming,HCero
    bset  OUT_C,PORTB
    jmp   HFinal
HCero:
    bclr  OUT_C,PORTB
HFinal:
    lsr   Hamming

    ; Saca en el puerto el bit normal
BCiclo:
    brclr 0,ByteDato,BCero
    bset  OUT_N,PORTB
    jmp   BFinal
BCero:
    bclr  OUT_N,PORTB
BFinal:
    lsr   ByteDato

    jsr   Retardo_200
    dbnzx HCiclo

    jmp   Principal

****************************************
*  Rutina principal del código de Línea  *
****************************************
Cod_Linea:
    mov   ByteRx,ByteDato
    ldx   #!8

  ; Saca en el puerto los bits normal y codificado
Ciclo:
    brclr 0,ByteDato,Cero
    bset  OUT_N,PORTB
    lda   PORTB
    eor   #!2
    sta   PORTB
    jmp   Final
Cero:
    bclr  OUT_N,PORTB
Final:
    lsr   ByteDato
    jsr   Retardo_200
    dbnzx Ciclo

    jmp   Principal

**********************
*  Retardo de 200ms  *
**********************
Retardo_200:
     pshh
     pshx
     ldhx  #!56
De_nuevo_200:
     aix   #-1
     cphx  #0
     bne   De_nuevo_200
     pulx
     pulh
     rts

************************************
*  Rutina de servicio de interrupción *
*  por byte recibido por la SCI       *
************************************
ISR_RX:
    pshh

    lda   SCS1
    lda   SCDR

    sta   ByteRx       ; Guarda el byte recibido
    bset  0,HayByte    ; Indica que hay un nuevo
byte

    pulh
    rti

************************************
*  Tabla de datos del código Hamming  *
************************************
Tabla: db  $0,$31,$52,$63,$64,$55,$36,$07,$78,$49,
           $2A,$1B,$1C,$2D,$4E,$7F,

************************************
*  Tabla de vectores de interrupción  *
************************************
    org DATO_RX
    dw  ISR_RX
```

```
org RESET
dw  INICIO
```

IV. CONCLUSIONES

- La selección de un código de línea reside en las diferencias de las características de cada una de las señales, estas son las que hacen que sean ideales para algunas aplicaciones y de poco interés para otras.

- El código de línea *NRZI* presenta una componente de DC considerable, lo cual hace que el sistema que diseñamos no sea acoplable en AC. Esto es de vital importancia si requerimos de acoplamientos o en sistemas de grabación magnética donde la información de muy baja frecuencia puede llegar a perderse.

- Los códigos de bloque incorporan cierta redundancia a las palabras del mensaje. Esa redundancia es un grupo de bits que no portaban información esencial y que su único objetivo es el de detectar y corregir errores que se pudieran producir durante la transmisión.

- La codificación de *Hamming* 7-4 permite detectar dos errores y corregir como máximo uno.

- La efectividad de los códigos de bloque depende de la diferencia entre una palabra de código válida y otra. Cuanto mayor sea esta diferencia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores.