15.2 Transparencia de datos

La transparencia de datos es la capacidad de empaquetar información con un bloque de inicio o encabezado y un bloque final del paquete de información, el éxito de la transparencia esta en no repetir los datos de encabezado o final dentro de la información. De esta manera se puede ver de forma transparente la información dentro de los datos de control. Para encabezar la información se implementa el carácter DLE o marcador de transparencia equivalente al código 0x10, STX comienzo de texto que equivale al código 0x02, y para finalizar el bloque de datos se usa el carácter ETX que es equivalente a 0x03.

Para demostrar el funcionamiento de la estructura de transparencia se implementan a continuación dos programas uno que transmite información y otro que la recibe, los programas esta fundamentados en un microcontrolador PIC 16F628A, y un PIC 16F877A ambos con oscilador de 4MHz, a continuación se puede apreciar el código fuente del modulo receptor que es similar al ejemplo del control de flujo:

```
// Declaración de constantes del LCD
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D7_Direction at TRISB7_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB4_bit;

//Declaración de variables de trabajo.
char BUFER[20], Dato, Bandera=0;
unsigned short Estado=0, N=0;

//Declaración de constantes.
const char ACK=0x06, NAK=0x15, ENQ=0x05, EOT=0x04, STX=0x02, ETX=0x03,
DLE=0x10,
NULL=0x00;

//Vector de interrupciones.
void interrupt()
{
   //Se evalúa la interrupción por recepción serial.
   if( PIR1.F5 )
   {
      //Se lee el Dato de entrada.
      Dato = UART1_Read();
      //Se evalúa el estado del tráfico de información.
      switch( Estado )
```

```
{
    //En espera...
    case  0: //Se evalúa si llega una solicitud..
        if( Dato == ENQ )
        {
          //Se cambia al siguiente Estado y se
          //envía el reconocimiento positivo.
          Estado = 1;
          UART1_Write(ACK);
        }
        else
        {
          //Se envía reconocimiento negativo.
          UART1_Write(NAK);
        }
        break;
    //Delimitador de transparencia...
    case  1: // Se evalúa si lo que llega es DLE.
        if( Dato == DLE )
        {
          //Se cambia al siguiente Estado...
          Estado = 2;
        }
        else
        {
          //Se identifica un error de orden en los datos
          //y se transmite un reconocimiento negativo y se
          //reanuda al Estado inicial.
          //Se reinicia de nuevo al Estado inicial.
          Estado = 0;
          UART1_Write(NAK);
        }
        break;
    // Recibiendo información...
    case 2:  // Se evalúa si lo que llega es STX.
        if( Dato == STX )
        {
          //Se inicia la cadena de información y se
          //pasa al Estado 3
          N=0;
          BUFER[N]=NULL;
          Estado = 3;
        }
        else
        {
          //Se identifica un error de orden en los datos
          //y se transmite un reconocimiento negativo y se
          //reanuda al Estado inicial.
          //Se reinicia de nuevo al Estado inicial.
```

```
            Estado = 0;
            UART1_Write(NAK);
        }
        break;
// Esperando datos y delimitador de transparencia.
case 3:  //Se evalúa si el Dato que llega es un DLE.
        if( Dato == DLE )
        {
            //Se cambia al Estado siguiente.
            Estado = 4;
            //Se finaliza el BUFER de datos;
            BUFER[N]=NULL;
        }
        else
        {
            //Se anexa un nuevo Dato al BUFER.
            BUFER[N]=Dato;
            //Se incrementa el índice de los datos.
            N++;
        }
        break;
// Esperando fin de cadena...
case 4: // Se evalúa si llega un ETX
       if( Dato == ETX )
       {
            //Se cambia al Estado siguiente.
            Estado = 5;
       }
       else
       {
            //Se cambia de nuevo al Estado inicial.
            Estado = 0;
            //Se transmite reconocimiento negativo.
            UART1_Write(NAK);
       }
       break;
//Esperando fin de transmisión...
case 5: //Se evalúa si llega el EOT...
       if( Dato == EOT )
       {
           //Se activa la Bandera de llegada de información.
           Bandera = 1;
           //Se transmite reconocimiento positivo.
           UART1_Write(ACK);
       }
       else
       {
           //Se transmite reconocimiento negativo.
           UART1_Write(NAK);
```

```
                }
                Estado = 0;
                break;

            default: break;
        }
    }
}

void main( void )
{
    //Se activan las interrupciones periféricas.
    INTCON = 0b11000000;
    //Se activa la interrupción por recepción serial.
    PIE1   = 0b00100000;
    //Configuración del modulo USART a 9600 bps.
    UART1_Init(9600);
    //Configuración del LCD.
    Lcd_Init();
    //Texto de inicio.
    Lcd_Out(1,1,"Transparencia");
    while(1) //Bucle infinito.
    {
        //Se evalúa si la Bandera esta activa.
        if( Bandera == 1 )
        {
            //Se imprime la información en el LCD.
            Lcd_Out(2,1,"            ");
            Lcd_Out(2,1,BUFER);
            Bandera = 0;
        }
    }
}
```

El siguiente programa está diseñado para un microcontrolador PIC 16F877A, el cual toma la lectura de un sensor LM35, y la transmite con el respectivo control de flujo y transparencia:

```
//Declaración de constantes.
const char ACK=0x06, NAK=0x15, ENQ=0x05, EOT=0x04, STX=0x02, ETX=0x03, DLE=0x10, NULL=0x00;

void main( void )
{
    char BUFER[10];
    int  adc;
    float Temp;
    //Se configura el modulo ADC, con el pin
    //AN3 como voltaje de referencia.
    ADCON1 = 0b11000001;
```

```
    //Se inicia el modulo USART a 9600 bps.
    UART1_Init(9600);
    delay_ms(500);
    while(1)//Bucle infinito.
    {
        OTRO:
        //Lectura del canal 0
        adc = ADC_Read(0);
        //Se calcula el valor de la temperatura.
        Temp = 0.244*adc;
        //Se escribe la temperatura en el BUFER.
        FloatToStr( Temp, BUFER );
        //Retardo de lectura de 400m segundos
        delay_ms(100);
        //Se transmite el ENQ...
        UART1_Write(ENQ);
        //Se espera el ACK...
        while( !UART1_Data_Ready() );
        //Se verifica si llega un ACK...
        if( UART1_Read() != ACK )goto OTRO;
        //Se transmite un DLE...
        UART1_Write(DLE);
        //Se transmite un STX...
        UART1_Write(STX);
        //Se transmite la información del BUFER...
        UART1_Write_Text(BUFER);
        //Se transmite un DLE...
        UART1_Write(DLE);
        //Se transmite el ETX...
        UART1_Write(ETX);
        //Se transmite el fin de transmisión...
        UART1_Write(EOT);
        //Se espera el ACK
        while( !UART1_Data_Ready() );
        //Se verifica si llega un ACK...
        if( UART1_Read() != ACK )goto OTRO;
    }
}
```

La estructura para la transparencia de datos obedece a la siguiente forma:

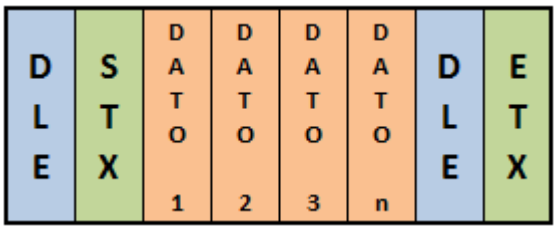
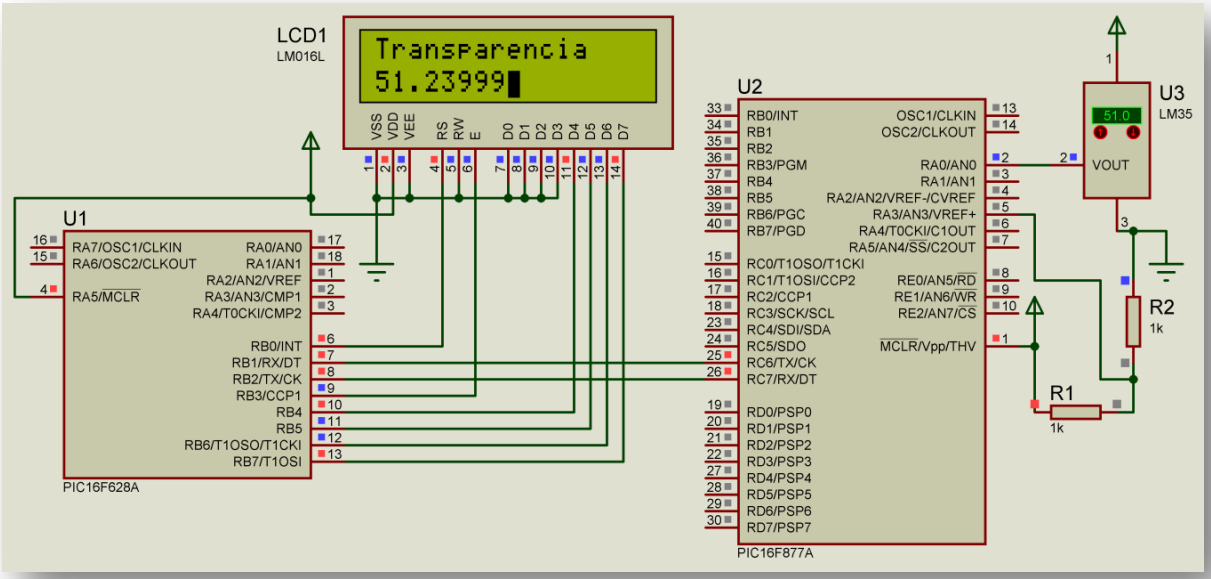


Figura 15-3

Para realizar la respectiva simulación se realiza el siguiente circuito que tiene los dispositivos, PIC 16F628A, PIC 16F877A, RES, LM016L, LM35:



Circuito 15-2

15.3 Control de errores CRC

El control de errores en una transmisión es de suma importancia y más cuando la calidad del medio de transmisión no es confiable. Para detectar los errores de transmisión existen diferentes técnicas entre las cuales se pueden citar, VRC o verificación de redundancia vertical, esta estrategia anexa un bit más a cada paquete de información el cual representa el número par o impar de unos en el paquete. LRC o Verificación de redundancia longitudinal la cual verifica el número de bits pares o impares en un grupo de bytes, para este fin se realiza la operación lógica o exclusiva, entre ellos. Por último se puede hablar del CRC, o verificación de redundancia cíclica, que es la estrategia de mayor efectividad, y que es la técnica en la cual se centra el ejemplo de este capítulo. Para llegar al resultado del CRC, se efectúa una división binaria entre el paquete de datos que se desean transmitir y un polinomio fijo que debe respetar algunas condiciones. Después de hacer la división el residuo de la misma es anexado al paquete de información y este es el CRC. El ente receptor realiza la misma división con el mismo polinomio y después de culminarla el resultado consignado en el residuo debe ser 0 si los datos han llegado sin ningún error, de lo contrario los datos del paquete están deteriorados.

La estructura de información con el CRC incorporado se implementa en la forma que se puede apreciar en la siguiente figura:

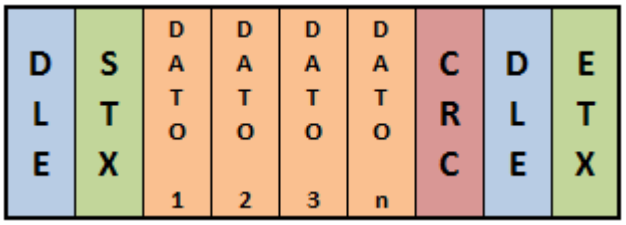


Figura 15-4

Los polinomios que se implementan en el cálculo del CRC se denotan en términos de X, en el cual cada X, es un 1 en el polinomio, para ilustrar esta situación se puede apreciar el siguiente ejemplo:

$$X^7 + X^5 + X^2 + X + 1$$

Este polinomio equivale al siguiente número binario:

10100111

Para que los polinomios funciones de forma adecuada deben cumplir con las siguientes condiciones:

- Debe empezar y terminar con un 1.
- No debe ser divisible por X, es decir por 10.
- Debe ser divisible por X+1, es decir por 11.

Los siguientes son polinomios estándar para diferente número de bits:

CRC-4

$$X^4 + X + 1$$

CRC-8

$$X^8 + X^2 + X + 1$$
$$X^8 + X^5 + X^4 + 1$$

CRC-12

$$X^{12} + X^{11} + X^3 + X + 1$$

CRC-16

$$X^{16} + X^{15} + X^2 + 1$$
$$X^{16} + X^{12} + X^5 + 1$$

CRC-32

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

El siguiente ejemplo aritmético muestra una división para generar el CRC, y su respectiva verificación:

Información 1 0 1 1 0 1 0 1 1

Polinomio 1 0 0 1 1

Calculo del CRC

```
1 0 1 1 0 1 0 1 1 0 0 0 0
1 0 0 1 1 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
  0 1 0 1 1 ↓ ↓ ↓ ↓ ↓ ↓ ↓
  0 0 0 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓
    1 0 1 1 0 ↓ ↓ ↓ ↓ ↓ ↓
    1 0 0 1 1 ↓ ↓ ↓ ↓ ↓ ↓
      0 1 0 1 1 ↓ ↓ ↓ ↓ ↓
      0 0 0 0 0 ↓ ↓ ↓ ↓ ↓
        1 0 1 1 1 ↓ ↓ ↓ ↓
        1 0 0 1 1 ↓ ↓ ↓ ↓
          0 1 0 0 0 ↓ ↓ ↓
          0 0 0 0 0 ↓ ↓ ↓
            1 0 0 0 0 ↓ ↓
            1 0 0 1 1 ↓ ↓
              0 0 1 1 0 ↓
              0 0 0 0 0 ↓
                0 1 1 0 0
                0 0 0 0 0
                  1 1 0 0
```

← CRC

Se cambia el polinomio por 0 → dado que el bit mas significativo es cero

Chequeo del CRC

```
1 0 1 1 0 1 0 1 1 1 1 0 0
1 0 0 1 1 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
  0 1 0 1 1 ↓ ↓ ↓ ↓ ↓ ↓ ↓
  0 0 0 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓
    1 0 1 1 0 ↓ ↓ ↓ ↓ ↓ ↓
    1 0 0 1 1 ↓ ↓ ↓ ↓ ↓ ↓
      0 1 0 1 1 ↓ ↓ ↓ ↓ ↓
      0 0 0 0 0 ↓ ↓ ↓ ↓ ↓
        1 0 1 1 1 ↓ ↓ ↓ ↓
        1 0 0 1 1 ↓ ↓ ↓ ↓
          0 1 0 0 1 ↓ ↓ ↓
          0 0 0 0 0 ↓ ↓ ↓
            1 0 0 1 1 ↓ ↓
            1 0 0 1 1 ↓ ↓
              0 0 0 0 0 ↓
              0 0 0 0 0 ↓
                0 0 0 0 0
                0 0 0 0 0
                  0 0 0 0
```

Chequeo →

La implementación de una división de este modo es demasiado compleja. Cuanto más larga sea la trama más difícil es realizarla, por eso es más simple realizar una operación lógica con registros de desplazamiento como estrategia para determinar el residuo de la división. El siguiente diagrama muestra un circuito para el polinomio:

$$X^{16}+X^{15}+X^{2}+1$$

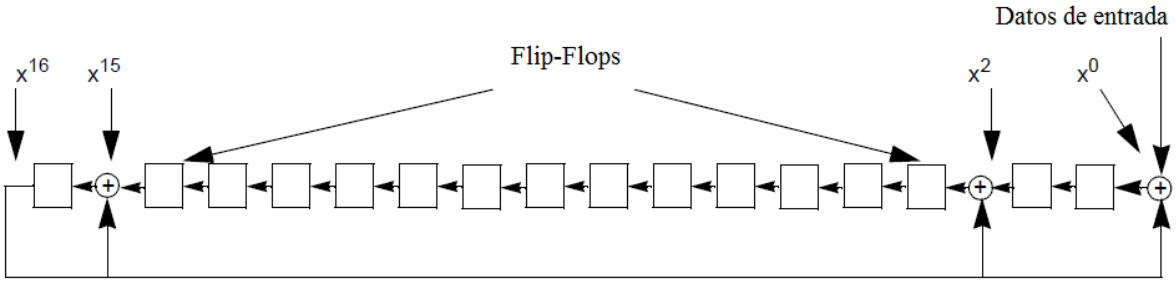


Figura 15-5

Usando como estrategia de programación el corrimiento de bits como lo muestra la figura se puede hacer un conjunto de funciones que calculen el CRC, sobre una trama de longitud indefinida teniendo como base mínima un byte y polinomios con orden múltiplo de 8, tales como 8, 16, 24, 32. Para comprender con claridad este concepto se puede observar el siguiente ejemplo que usa un polinomio de orden 8:

Polinomio:

$$X^{8}+X^{5}+X^{4}+1$$

100110001

```
 unsigned short CRC=0;

//Función para insertar in bit en el cálculo del CRC.
void CorrerBitCRC( char _bit )
{
    unsigned short CRC2=0;
    CRC2 = CRC2 | (CRC<<1)&128; //Set X7
    CRC2 = CRC2 | (CRC<<1)&64; //Set X6
    CRC2 = CRC2 | ((CRC<<1)^(CRC>>2))&32; //Set X5
    CRC2 = CRC2 | ((CRC<<1)^(CRC>>3))&16; //Set X4
    CRC2 = CRC2 | (CRC<<1)&8; //Set X3
    CRC2 = CRC2 | (CRC<<1)&4; //Set X2
    CRC2 = CRC2 | (CRC<<1)&2; //Set X1
    CRC2 = CRC2 | ((CRC>>7)^_bit)&1; //Set X0
    CRC  = CRC2;
}
//Función para insertar un Byte al cálculo del CRC.
void CorrerByteCRC( char Byte )
{
    short B;
    for( B=7; B>=0; B-- )
     CorrerBitCRC( (Byte>>B)&1 );
}

//Función para calcular el CRC de una trama de texto.
unsigned short CalcularRCR( char *trama, unsigned char FINAL )
{
    unsigned short N=0;
    CRC = 0;
    while( trama[N]!=0 )
    {
        CorrerByteCRC( trama[N] );
        N++;
    }
    CorrerByteCRC( FINAL );
    return CRC;
}
```

Para realizar la implementación de este ejemplo, se usa el mismo hardware del ejemplo de transparencia, el código fuente para el microcontrolador transmisor implementando el CRC es el siguiente:

```
//Función para insertar un Byte al cálculo del CRC.
void CorrerByteCRC( char Byte )
{
    short B;
    for( B=7; B>=0; B-- )
     CorrerBitCRC( (Byte>>B)&1 );
}
```

```
//Función para calcular el CRC de una trama de texto.
unsigned short CalcularRCR( char *trama, unsigned char FINAL )
{
    unsigned short N=0;
    CRC = 0;
    while( trama[N]!=0 )
    {
        CorrerByteCRC( trama[N] );
        N++;
    }
    CorrerByteCRC( FINAL );
    return CRC;
}

void main( void )
{
    char BUFER[10];
    int  adc;
    float Temp;
    //Se configura el modulo ADC, con el pin
    //AN3 como voltaje de referencia.
    ADCON1 = 0b11000001;
    //Se inicia el modulo USART a 9600 bps.
    UART1_Init(9600);
    delay_ms(500);
    while(1)//Bucle infinito.
    {
        OTRO:
        //Lectura del canal 0
        adc = ADC_Read(0);
        //Se calcula el valor de la temperatura.
        Temp = 0.244*adc;
        //Se escribe la temperatura en el BUFER.
        FloatToStr( Temp, BUFER );
        //Calculo del CRC.
        CalcularRCR( BUFER, 0 );
        //Retardo de lectura de 400m segundos
        delay_ms(100);
        //Se transmite el ENQ...
        UART1_Write(ENQ);
        //Se espera el ACK...
        while( !UART1_Data_Ready() );
        //Se verifica si llega un ACK...
        if( UART1_Read() != ACK )goto OTRO;
        //Se transmite un DLE...
        UART1_Write(DLE);
        //Se transmite un STX...
        UART1_Write(STX);
        //Se transmite la información del BUFER...
```

```
        UART1_Write_Text(BUFER);
        //Se transmite el CRC...
        UART1_Write(CRC);
        //Se transmite un DLE...
        UART1_Write(DLE);
        //Se transmite el ETX...
        UART1_Write(ETX);
        //Se transmite el fin de transmisión...
        UART1_Write(EOT);
        //Se espera el ACK...
        while( !UART1_Data_Ready() );
        //Se verifica si llega un ACK...
        if( UART1_Read() != ACK )goto OTRO;
    }
}
```

El código fuente para el microcontrolador receptor con la implementación del CRC es el siguiente:

```
// Declaración de constantes del LCD
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D7_Direction at TRISB7_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB4_bit;

//Declaración de variables de trabajo.
char BUFER[20], Dato, Bandera=0;
unsigned short Estado=0, N=0;

//Declaración de constantes.
const char ACK=0x06, NAK=0x15, ENQ=0x05, EOT=0x04, STX=0x02, ETX=0x03,
DLE=0x10,
NULL=0x00;

unsigned short CRC=0, CRC_2, Lon;

//Función para insertar in bit en el cálculo del CRC.
void CorrerBitCRC( char _bit )
{
    unsigned short CRC2=0;
```

```
    CRC2 = CRC2 | (CRC<<1)&128; //Set X7
    CRC2 = CRC2 | (CRC<<1)&64; //Set X6
    CRC2 = CRC2 | ((CRC<<1)^(CRC>>2))&32; //Set X5
    CRC2 = CRC2 | ((CRC<<1)^(CRC>>3))&16; //Set X4
    CRC2 = CRC2 | (CRC<<1)&8; //Set X3
    CRC2 = CRC2 | (CRC<<1)&4; //Set X2
    CRC2 = CRC2 | (CRC<<1)&2; //Set X1
    CRC2 = CRC2 | ((CRC>>7)^_bit)&1; //Set X0
    CRC  = CRC2;
}

//Función para insertar un Byte al cálculo del CRC.
void CorrerByteCRC( char Byte )
{
    short B;
    for( B=7; B>=0; B-- )
     CorrerBitCRC( (Byte>>B)&1 );
}

//Función para calcular el CRC de una trama de texto.
unsigned short CalcularRCR( char *trama, unsigned char FINAL )
{
    unsigned short N=0;
    CRC = 0;
    while( trama[N]!=0 )
    {
        CorrerByteCRC( trama[N] );
        N++;
    }
    CorrerByteCRC( FINAL );
    return CRC;
}

//Vector de interrupciones.
void interrupt()
{
    //Se evalúa la interrupción por recepción serial.
    if( PIR1.F5 )
    {
       //Se lee el Dato de entrada.
       Dato = UART1_Read();
       //Se evalúa el estado del tráfico de información.
       switch( Estado )
       {
            //En espera...
            case  0: //Se evalúa si llega una solicitud..
                 if( Dato == ENQ )
                 {
                    //Se cambia al siguiente Estado y se
```

```
            //envía el reconocimiento positivo.
            Estado = 1;
            UART1_Write(ACK);
        }
        else
        {
            //Se envía reconocimiento negativo.
            UART1_Write(NAK);
        }
        break;
//Delimitador de transparencia...
case  1: // Se evalúa si lo que llega es DLE.
        if( Dato == DLE )
        {
            //Se cambia al siguiente Estado...
            Estado = 2;
        }
        else
        {
            //Se identifica un error de orden en los datos
            //y se transmite un reconocimiento negativo y se
            //reanuda al Estado inicial.
            //Se reinicia de nuevo al Estado inicial.
            Estado = 0;
            UART1_Write(NAK);
        }
        break;
// Recibiendo información...
case 2:  // Se evalúa si lo que llega es STX.
        if( Dato == STX )
        {
            //Se inicia la cadena de información y se
            //pasa al Estado 3
            N=0;
            BUFER[N]=NULL;
            Estado = 3;
        }
        else
        {
            //Se identifica un error de orden en los datos
            //y se transmite un reconocimiento negativo y se
            //reanuda al Estado inicial.
            //Se reinicia de nuevo al Estado inicial.
            Estado = 0;
            UART1_Write(NAK);
        }
        break;
// Esperando datos y delimitador de transparencia.
case 3:  //Se evalúa si el Dato que llega es un DLE.
```

```
        if( Dato == DLE )
        {
           //Se cambia al Estado siguiente.
           Estado = 4;
           //Se finaliza el BUFER de datos;
           BUFER[N]=NULL;
        }
        else
        {
           //Se anexa un nuevo Dato al BUFER.
           BUFER[N]=Dato;
           //Se incrementa el índice de los datos.
           N++;
        }
        break;
// Esperando fin de cadena...
case 4: // Se evalua si llega un ETX
      if( Dato == ETX )
      {
           //Se cambia al Estado siguiente.
           Estado = 5;
      }
      else
      {
           //Se cambia de nuevo al Estado inicial.
           Estado = 0;
           //Se transmite reconocimiento negativo.
           UART1_Write(NAK);
      }
      break;
//Esperando fin de transmisión...
case 5: //Se evalúa si llega el EOT...
      if( Dato == EOT )
      {
          //Se mide la longitud de la trama.
          Lon = strlen(BUFER);
          //Se filtra el CRC, que está en la última posición.
          CRC_2 = BUFER[Lon-1];
          BUFER[Lon-1]=0;
          //Se calcula el CRC...
          CalcularRCR( BUFER, CRC_2 );
          // Se verifica que el chequeo sea cero...
          if( CRC==0 )
          {
            //Se activa la Bandera de llegada de información.
            Bandera = 1;
            //Se transmite reconocimiento positivo.
            UART1_Write(ACK);
          }
```

```
                    else
                    {
                      //Se transmite reconocimiento negativo.
                      UART1_Write(NAK);
                    }
                  }
                  else
                  {
                    //Se transmite reconocimiento negativo.
                    UART1_Write(NAK);
                  }
                  Estado = 0;
                  break;
              default: break;
        }
    }
}

void main( void )
{
    //Se activan las interrupciones periféricas.
    INTCON = 0b11000000;
    //Se activa la interrupción por recepción serial.
    PIE1   = 0b00100000;
    //Configuración del modulo USART a 9600 bps.
    UART1_Init(9600);
    //Configuración del LCD.
    Lcd_Init();
    //Texto de inicio.
    Lcd_Out(1,1,"Transparencia");
    while(1) //Bucle infinito.
    {
        //Se evalúa si la Bandera esta activa.
        if( Bandera == 1 )
        {
            //Se imprime la información en el LCD.
            Lcd_Out(2,1,"               ");
            Lcd_Out(2,1,BUFER);
            Bandera = 0;
        }
    }
}
```