PROGRAMA PRINCIPAL

```
/* INTERCAMBIO DE DATOS ARDUINO (actuando como servidor) y LABVIEW
(actuando como cliente).

   Se leen i=8 señales analógicas (0-5V) conectadas a los pins A0-
A7 de Arduino y se transmiten mediante un transmisor/receptor
bluetooth conectado a los pins TX14 y RX15.

   Los datos se transmiten en serie a través del puerto Serial
(correspondiente al transmisor/receptor Bluetooth) de Arduino que
funciona a 19200 bps sin salto de linea.

   Dichos datos se estructuran conforme al protocolo establecido de
la siguiente manera y son transmitidos y recibidos como caracteres
ASCII:

        <L| N_SECUENCIA | CODIGO_DE_SEGURIDAD |L | COMANDO_DATOS |
Li | DATOSi (2bytes/dato)| L| CONTROL DE ERROR | >

        Donde: < = Inicio de trama (1byte).
               L = Longitud (en bytes) del carácter proximo (1byte
                    equivalente a un caracter ASCII).
               N_SECUENCIA = Número de la secuencia (1byte).
               CODIGO_DE_SEGURIDAD = Código de seguridad (1byte =
                                     206)
               COMANDO_DATOS = Selección de modo de funcionamiento
                               (1 byte):
                                COMANDO: El cliente manda comandos
que ejecuta Arduino, tales como seleccionar que señales se
transmiten, encender una alarma...

                                DATOS: Se transmiten todas las
señales de forma normal.

              DATOSi = Datos de la variable i (valores entre 0-1023
correspondientes a una variable de 2bytes).
              CONTROL DE ERROR: Suma de los valores de todo lo
enviado anteriormente a excepción de los simbolos
              de inicio y fin de trama y el valor del propio error.

   El envio se realiza mediante un proceso de pregunta-respuesta,
por lo que hasta obtener una trama procedente de LabView válida,
Aruino permanece a la espera recibiendo.
*/


////////////////////////////////
//VARIABLES GLOBALES          //
////////////////////////////////

  #define PowerLED 23   //seleción del PIN en el que se encuentra
conectado el LED ROJO;
  #define LED_Amarillo 22 ///seleción del PIN en el que se
encuentra conectado el LED AMARILLO (ALARMA);
  #define c_seg 206    //codigo de seguridad = 206;
```

```
  unsigned int n_seq = 0;        // variable para almacenar el n°
de secuencia
  byte c_d = 0;        // Comandos o datos a ejecutar o continuar

  byte Sensor_PIN[] = {A7, A6, A5, A4, A3, A2, A1, A0}; //ENTRADAS
ANALÓGICAS
  unsigned int Sensor[8];  //Almacenamiento de los datos
procedentes de las variables analógicas
  byte l_S[8];             //Longitud de los datos procedentes de
las variables analógicas
  unsigned int retardo = 0;

  unsigned int dat_rec[8]; //Almacenamiento de los datos recibidos
para analizarlo y ejecutar los comandos que portan
  byte l_dat_rec[8];

  String datos_recibidos = "";    //String en el que se almacenan
datos recibidos.
   boolean recepcion = false;



////////////////////////////////
//DECLARACION DE FUNCIONES    //
////////////////////////////////

byte cuentaChars(unsigned long N);
byte l_error (int d_error);
unsigned int string2int (String objeto, int inicio, int fin);
String CAPTURA_DATOS (unsigned int c_d, unsigned int D1);
int CALCULA_ERROR(unsigned int n_seq, int c_d);
String CONSTRUIR_TRAMA(unsigned int n_seq, int c_d, String datos,
int error);
void ENVIA_DATOS();
void ANALIZA_DATOS();
void RECIBE_DATOS();




////////////////////////////////
//CONFIGURACIÓN               //
////////////////////////////////

void setup(){
   Serial3.begin(19200); //Inicialización del puerto Serie3 en el
que se encuentra conectado el módulo LM780 a Arduino
   Serial3.setTimeout(60000); //Timeout para la recepción de datos
  datos_recibidos.reserve(100); //Datos reservados para la
recepcion de datos
  pinMode(PowerLED, OUTPUT);  //Pin PowerLED como salida
  pinMode(LED_Amarillo, OUTPUT); //Pin LED_Amarillo como salida
}
```

```
////////////////////////////////
//PROGRAMA PRINCIPAL          //
////////////////////////////////

void loop(){

   if (Serial3.available()){  //RECIBE MIENTRAS EL PUERTO DE SERIE
ESTE DISPONIBLE
   RECIBE_DATOS();
   }
   if (recepcion){  //SI HA RECIBIDO CORRECTAMENTE DATOS
     ANALIZA_DATOS();  //ANALIZA
     ENVIA_DATOS();   //ENVIA
     datos_recibidos="";  //RESET DE LOS DATOS RECIBIDOS
     recepcion=false;  //RESET DEL FLAG DE RECEPCION
   }
}
```

FUNCIÓN RECIBE_DATOS.

```
extern String datos_recibidos;
extern boolean recepcion;

boolean continuar;
static char inChar;

void RECIBE_DATOS(){

  /* FUNCIÓN QUE REALIZA LA LECTURA DE LOS DATOS QUE SE RECIBEN EN
EL BUFFER
  DEL PUERTO SERIAL3 */
      char inChar = (char)Serial3.read();

    if (inChar == '<') {
      continuar = true;
      }

    if (continuar == true){
    datos_recibidos += inChar;
    }

    if (inChar == '>' && continuar == true){
      continuar = false;
      recepcion = true; //Se confirma que se ha recibido
correctamente
      }
}
```

FUNCIÓN ANALIZA_DATOS:

```
     //VARIABLES GLOBALES DE DENTRO DEL PROCESO DE ANÁLISIS DE
DATOS
      extern unsigned int n_seq;
      extern byte c_d;
      extern String datos_recibidos;

       //VARIABLES TEMPORALES PARA PROCESAR LOS DATOS RECIBIDOS
      static unsigned long suma_recibida = 0;
      static byte ESTADO = 0;
      static byte l_nseq;
      static unsigned int n_seq_recibida;
      static byte cseg_r;
      static byte l_cdre;
      static byte cd_r;
      static byte l_error_r;
      static unsigned long error_r;

      void ANALIZA_DATOS(){


     //COMPROBACIÓN DE QUE LA TRAMA RECIBIDA ES CORRECTA MEDIANTE
      //UNA MAQUINA DE ESTADOS FINITOS


      int indice = 1;

      //ESTADO 0 (n_seq)
      if (isDigit(datos_recibidos[1]) && ESTADO == 0){
        l_nseq =  datos_recibidos[indice] - '0';   //LECTURA DE LA
     LONGITUD DEL NÚMERO DE SECUENCIA
        indice++;  //Incremento del indice
        suma_recibida = l_nseq;  //Acomulación de valores para
comprobación de errores
        n_seq_recibida = string2int (datos_recibidos, indice,
indice+l_nseq); //Lectura del NUMERO DE SECUENCIA
        suma_recibida = suma_recibida + n_seq_recibida;
//Acomulación de valores para comprobación de errores
        indice = indice + l_nseq; //Incremento del indice
        ESTADO = 1;  //CAMBIO DE ESTADO.


        cseg_r = string2int(datos_recibidos, indice, indice+3);
//SE LEE EL CODIGO DE SEGURIDAD PARA COMPROBARLO EN EL SIGUIENTE
ESTADO
        indice=indice+3;
        suma_recibida = suma_recibida + cseg_r;
        } else { ESTADO = 0; }

     if (ESTADO == 1 && cseg_r == c_seg){
        //CODIGO DE SEGURIDAD CORRECTO
        l_cdre = datos_recibidos[indice] - '0'; //LECTURA DE la
longitud de C-D
        indice++;
        suma_recibida = suma_recibida + l_cdre;
        ESTADO = 2;
     } else
```

```
        { //CODIGO DE SEGURIDAD ERRONEO
            ESTADO = 0;
         }

     if (ESTADO == 2){
       cd_r = string2int(datos_recibidos, indice, indice+l_cdre);
//LECTURA DE C_D
       indice = indice + l_cdre;
       suma_recibida = suma_recibida + cd_r;
       ESTADO = 3;
     } else { ESTADO = 0; }

     if (ESTADO == 3){ //LECTURA DE LOS DATOS CORRESPONDIENTES A
COMANDOS
       for (int i=0; i<=7; i++){
         l_dat_rec[i] = datos_recibidos[indice] - '0';
         indice++;
         suma_recibida = suma_recibida + l_dat_rec[i];
         dat_rec[i] = string2int (datos_recibidos, indice,
indice+l_dat_rec[i]);
         indice = indice + l_dat_rec[i];
         suma_recibida = suma_recibida + dat_rec[i];
       }
       ESTADO = 4;
     } else { ESTADO = 0; }


     if (ESTADO == 4){ //LECTURA DEL ERROR
       l_error_r = datos_recibidos[indice] - '0';
       indice++;
       suma_recibida = suma_recibida + l_error_r;
       error_r=string2int(datos_recibidos,indice,indice+l_error_r);
       indice = indice + l_error_r;
       ESTADO = 5;
     } else { ESTADO = 0; }


     if (ESTADO == 5 && suma_recibida == error_r){ //COMPROBACIÓN
DE QUE LA TRAMA ES CORRECTA


       ///VALIDAR DATOS y EJECUTAR ACCIONES O COMANDOS.

       n_seq = n_seq+1;
       c_d = cd_r;
       ESTADO = 0;
     } else { ESTADO = 0; }
     }
```

FUNCIÓN ENVIA_DATOS

```
  extern unsigned int n_seq;        // variable para almacenar el
n° de secuencia
  extern byte c_d;        // Comandos o datos a ejecutar o
continuar
  extern unsigned int retardo;
  extern unsigned int dat_rec[8]; //Almacenamiento de los datos
recibidos para analizalo y ejecutar los comandos que portan




void ENVIA_DATOS(){

  /* FUNCIÓN QUE CONTRUYE UNA TRAMA DE DATOS Y LA ENVIA A TRAVÉS
DEL PUERTO SERIAL3 */

//delay(retardo);
    //TRANSMISIÓN DE UNA SECUENCIA DE DATOS

    String datos = CAPTURA_DATOS (c_d, dat_rec[0]); //Obtención de
variables analógicas en función de c_d recibido
    int error =  CALCULA_ERROR(n_seq, c_d);
    String trama = CONSTRUIR_TRAMA(n_seq, c_d, datos, error);
    Serial3.print(trama);


    //RESET DE LOS FLAGS
}
```

FUNCIÓN CAPTURA_DATOS

```
extern byte c_d;
extern unsigned int Sensor[8];
extern byte l_S[8];
extern unsigned int dat_rec[8];
extern byte l_dat_rec[8];
extern unsigned int retardo;

String CAPTURA_DATOS (unsigned int c_d, unsigned int seleccion){
    /*FUNCION QUE REALIZA LA LECTURA DE LAS VARIABLES ANALÓGICAS
      EN FUNCIÓN DEL CAMPO C_D RECIBIDO y DATOS 1 (donde cada bit
    corresponde a el encendido o apagado de una de las variable) y
    LO ALMACENA EN EL ARRAY Sensor[i]*/

  String enviar_sensor = "";

    if ((bitRead(c_d, 7) == 1) && (bitRead(c_d, 0) == 1)){
      //SELECCIÓN DE VARIABLES
      for (int i = 7; i>=0; i--){
        if(bitRead(seleccion, i) == 1){
          Sensor[i] = analogRead(Sensor_PIN[i]);delay(1);
          l_S[i]=cuentaChars(Sensor[i]);
          enviar_sensor += String(l_S[i]);
```

```
          enviar_sensor += String(Sensor[i]);
        } else {Sensor[i] = 0;
              l_S[i]=cuentaChars(Sensor[i]);
              enviar_sensor += String(l_S[i]);
              enviar_sensor += String(Sensor[i]);
         }
      }
    }
    else{
      for (int i = 7; i>=0; i--){
        Sensor[i] = analogRead(Sensor_PIN[i]);delay(1);
        l_S[i]=cuentaChars(Sensor[i]);
          enviar_sensor += String(l_S[i]);
          enviar_sensor += String(Sensor[i]);
      }
   }

  //if ((bitRead(c_d, 7) == 1) && (bitRead(c_d, 1) == 1)){
  //    // RETARDO
  //    retardo = dat_rec[1] * 1000;
  //} else{ retardo = 0; }

  if ((bitRead(c_d, 7) == 1) && (bitRead(c_d, 2) == 1)){
      //ALARMA 2
        digitalWrite(PowerLED, HIGH);
      }
      else{
        digitalWrite(PowerLED, LOW);
      }


    if ((bitRead(c_d, 7) == 1) && (bitRead(c_d, 3) == 1)){
      //ALARMA
        digitalWrite(LED_Amarillo, HIGH);
      }
      else{
        digitalWrite(LED_Amarillo, LOW);
      }

  return(enviar_sensor);
}
```

FUNCIÓN CONSTRUIR_TRAMA:

```
extern unsigned int n_seq;
extern byte c_d;
extern String datos;
extern int error;

String CONSTRUIR_TRAMA(unsigned int n_seq, int c_d, String datos,
int error){

    /*  FUNCIÓN QUE CONSTRUYE EL STRING DE DATOS A ENVIAR */

  String TEMP = "<";
```

```
  TEMP += String(cuentaChars(n_seq)) += String (n_seq);
  TEMP += String(206);
  TEMP += String(cuentaChars(c_d)) += String (c_d);
  TEMP += datos;
  TEMP += String(l_error(error)) += String (error);
  TEMP += '>';
  return (TEMP);
}
```

FUNCIÓN CALCULA_ERROR.

```
extern unsigned int n_seq;
extern byte c_d;
extern unsigned int Sensor[8];
extern byte l_S[8];

int CALCULA_ERROR(unsigned int n_seq, byte c_d){
  /*FUNCIÓN QUE CALCULA EL NÚMERO DE CONTROL DE ERRORES
  QUE CORRESPONDE A LA SUMA DE TODOS LOS VALORES ENVIADOS
  ANTERIORMENTE*/

 int TEMP = 0;
 TEMP = cuentaChars(n_seq) + n_seq + 206;
 TEMP = TEMP + cuentaChars(c_d) + c_d;
 for (int i = 7; i>=0; i--){
   TEMP = TEMP + Sensor[i] + l_S[i];
 }
 TEMP = TEMP + l_error(TEMP);
 return (TEMP);
}
```

FUNCIÓN cuentaChars

```
extern unsigned long N;

 byte cuentaChars(unsigned long N){
   /*FUNCIÓN QUE CUENTAL EL NÚMERO DE CARACTERES QUE TIENE UN
NÚMERO entero de 2 bytes
   cuyo valor máximo es 2^16=65536, y por tanto 5
cárácteres=1byte*/
   if (N>=0 && N<10){ return(1); }
   if (N>=10 && N<100) { return (2); }
   if (N>=100 && N<1000) { return (3) ; }
   if (N>=1000 && N<10000) { return (4) ; }
   if (N>=10000 && N<100000) { return (5); }
 }
```

FUNCION L_ERROR

```
 extern int d_error;

 byte l_error (int d_error){
   /*FUNCIÓN que corrige la longitud del error, ya que esta también
se suma
   al control de errores*/

   if (d_error<9){return(1);}
   if (d_error>=9 && d_error<98){return (2); }
   if (d_error>=98 && d_error<997) {return(3); }
   if (d_error>=997 && d_error<9996) {return(4); }
   if (d_error>=9996) { return(5); }
 }
```

FUNCION string2int

```
extern String objeto;
extern int inicio;
extern int fin;

 unsigned int string2int (String objeto, int inicio, int fin){
    /* FUNCION string2int:
       ENTRADAS: 1. String en el que realizar la conversión
                 2. Indice del sting en el que empieza el número.
                 3. Indice del string en el que termina el número.

       SALIDAS: 1. Número en formato unsigned int.
                   En caso de no ser un número no funciona.
       */

  String TEMP;
  for (int i=inicio; i < fin; i++){
    TEMP += objeto[i];
  }
  return(TEMP.toInt());
}
```