Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

RED INALÁMBRICA DE SENSORES POR RADIOFRECUENCIA

DOCUMENTO 3 - CÁLCULOS DE SOFTWARE

Alumno/Alumna: Dieguez Martín Alexander

Director/Directora: Oleagordia Aguirre Iñigo Javier

Curso:2018-2019

Fecha: 15/ 07/ 2019

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El entorno de desarrollo Arduino ha ido evolucionando constantemente a lo largo del tiempo, incluyendo numerosas librerías para facilitar el manejo de los componentes electrónicos. El IDE de Arduino es un programa informático compuesto por un conjunto de herramientas de programación. Esto hace que se pueda realizar casi cualquier código para poder controlar cualquier elemento electrónico hasta ciertos niveles. Gracias a sus numerosas actualizaciones de la interface IDE de Arduino se van añadiendo cada vez más placas, subsanando los errores y añadiendo más comandos de programación.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Además, en el caso de Arduino, incorpora las herramientas para cargar el programa ya compilado en la memoria flash del hardware a través del puerto serie.

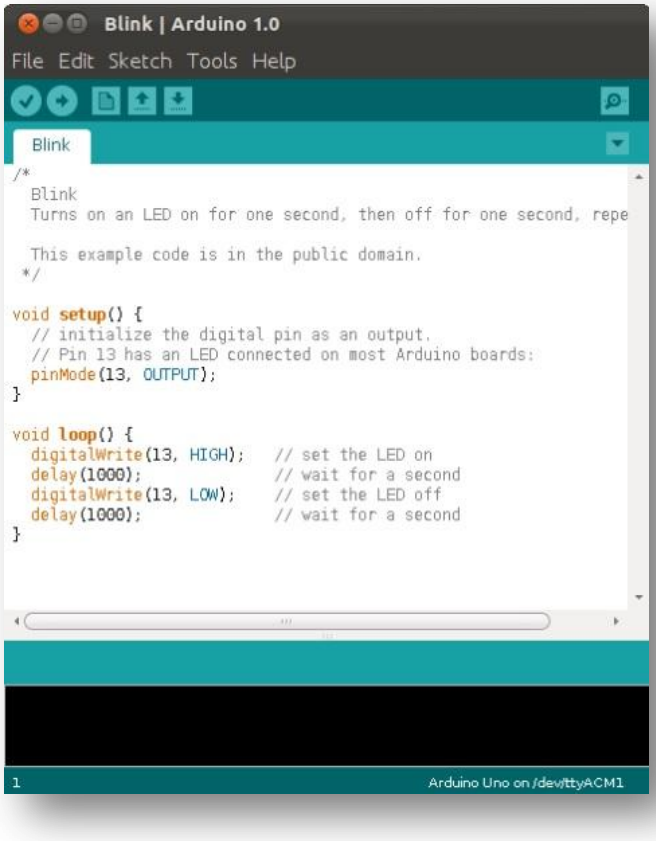

Figura 1.1 Entorno IDE de programación de Arduino

Los programas de Arduino están compuestos por un solo fichero con extensión *".ino"*, aunque es posible organizarlo en varios ficheros. El fichero principal siempre debe estar en una carpeta con el mismo nombre que el fichero.

La estructura básica de un sketch de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes son obligatorias y encierran bloques que contienen declaraciones e instrucciones como se puede apreciar en la fig. 1.1

- Setup()

Es la parte encargada de recoger la configuración. La función de configuración setup() debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para configurar o definir los pines a utilizar, e incorporar los diferentes "*includes*" en la programación. Esto es según el número de librerías que se vayan a utilizar.

- Loop()

Es la que contiene el programa que se ejecutará cíclicamente. La función setup() se invoca una sola vez, que es cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pines o el puerto serie. Debe ser incluido en un programa, aunque no haya declaración que ejecutar. Así mismo se puede utilizar para establecer el estado inicial de las salidas de la placa.

Por la facilidad que presenta esta plataforma a la hora de inicializar variables, configurar *timers*, configurar convertidores D/A o A/D, entradas digitales, entre muchas cosas más, se ha elegido el Arduino para programar el sensor DHT11 y la pantalla LCD I2C.

El lenguaje de programación Arduino se deriva del lenguaje Processing, el cual a su vez surgió de Java. De hecho, el Arduino IDE es un software desarrollado en Java. Por lo tanto, el lenguaje que se utilizará es el C.

2. PROGRAMACIÓN EN C

Debido a que se trabaja con Arduino, se va a utilizar el lenguaje en C. Como se ha mencionado anteriormente, el entorno de desarrollo utiliza 2 funciones (setup y loop) que se tienen que configurar. Las constantes, se pueden programar fuera de las funciones, si se quiere que sean globales.

Lo primero que se hará es inicializar todas las librerías que se van a utilizar. A continuación, se definirán las constantes, la función void setup(), se crearán nuevas funciones en el programa y por último, se programará la función main void loop().

2.1 INICIALIZACIÓN DE LIBRERÍAS

Las librerías son programas que se desarrollan con el fin de crear una interfaz que facilitan mucho las operaciones y simplifica las líneas de código. De una forma más completa, las librerías en Java permiten reutilizar código, es decir, que se puede hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar nosotros mismos esas funcionalidades.

A continuación, se van a incluir las librerías utilizadas sobre los tres programas de Arduino programados. Estos incluyen la programación del emisor, receptor y el módulo Wifi.

- #include <ESP8266WiFi.h>: Esta librería proporciona las rutinas específicas Wifi de ESP8266 a las que se llaman para conectar a la red. El módulo ESP8266 puede funcionar como una estación, por lo que se puede conectar a la red Wifi. Y también puede funcionar como un punto de acceso wireless (SoftAP), para establecer su propia red Wifi. Por lo tanto, se puede conectar otras estaciones a dicho módulo ESP. Teniendo esto en cuenta es posible que el módulo ESP pueda operar tanto en modo estación, como en modo punto de acceso. La biblioteca ESP8266WiFi proporciona una amplia colección de métodos C++ y propiedades o atributos para configurar y operar un módulo ESP en modo estación y/o punto de acceso. Esta librería se divide en varías clases. En la mayoría de los casos, al escribir el código, el usuario no está interesado en esta clasificación. Se utiliza para dividir la descripción de esta librería en piezas más manejables.
- #include <DHT11.h>: La codificación de datos está basada en un esquema de ancho de pulso. Un ancho de pulso representa un 1 lógico, un pulso corto representa un 0 lógico. Todos los bits comienzan con un pulso bajo de 50uS. Las librerías de comunicación con el DHT11 aprovechan este pulso para la sincronización. Luego viene un pulso alto que varía según el estado lógico o el valor del bit que el DHT11 desea transmitir. Esta librería implementa el protocolo para la comunicación con el DHT11 y DHT22.
- #include <VirtualWire.h>: Esta librería se encarga de gestionar las funciones de los módulos RF como envío y recepción de paquetes de datos, comprobación de errores, etc... Se utiliza la librería para mejorar la robustez de la comunicación. Las transmisiones ASK requieren una serie de impulsos de "entrenamiento" para sincronizar el receptor y el transmisor. También necesitan de un buen balance entre 1s y 0s para mantener el balance DC del mensaje. Con la librería Virtual Wire cada transmisión consta de un código de entrenamiento, el mensaje, la longitud del mensaje, y el checksum. Los mensajes son enviados con codificación 4 a 6 bit, para mejorar el balance DC

- #include "LowPower.h": Ofrece una sencilla API desde la que manejar el modo sleep del Arduino. Concretamente, se deben especificar los siguientes valores:
  - SLEEP_xD: Fija el tiempo que estará "dormido" usando para ello el Watchdog
  - ADC_OFF: Apaga los convertidores analógico-digital.
  - BOD_OFF: Apaga el circuito de Brown Out Detection, que es un circuito que sirve como detector de niveles de tensión peligrosamente bajos que podrían incluso llegar a dañar los circuitos.
- #include <Wire.h>: Para usar el bus I2C en Arduino, el IDE Standard proporciona la librería "Wire.h", que contiene las funciones necesarias para controlar el hardware integrado. Permite comunicar con I2C/TWI Arduino con otros dispositivos. La comunicación se establece a través de los pines SDA(línea de datos, ubicada en pun analógico 4 en la placa Arduino UNO) y SCL(línea de reloj, ubicada en el pin analógico 5).
- #include <LiquidCrystal_I2C.h>: Esta librería contiene todo lo necesario para gestionar un display LCD con Arduino. Permite crear un objeto que representa al display LCD y que contiene todas las operaciones "debajo nivel" para que resulte fácil la programación de este dispositivo. Como argumentos recibe una serie de números que se refieren a pines concretos de la palca Arduino, conectados a diferentes pines del display.

2.2 DECLARACIÓN DE CONSTANTES

Después de declarar todas las librerías se procede a la declaración de constantes. La principal ventaja de utilizar constante es que una vez asignado un valor, esta constante puede ser utilizada dentro de cualquier expresión o condición dentro del programa. De este modo se evita el tener que rescribir un mismo valor; o bien; si un valor ha de ser cambiado, no es necesario buscar dicho valor a lo largo de todo el programa, ya que basta con cambiar el valor de la constante. Este es un método eficaz para desarrollar software reutilizable.

Las constantes que se han declarado tipo "define" son las siguientes:

- DHT11 dht11(D4): Define el pin digital 4 del sensor DHT11.
- DHTPIN 8: Pin al que se tiene que conectar del circuito.
- DHTTYPE DHT11: Define el tipo de sensor utilizado. Tiene como elección tanto el DHT11 como el DHT22.

2.3 DECLARACIÓN DE VARIABLES

A continuación, se detallan las variables globales que se han utilizado en el sistema:

- const char* ssid: Nombre id para la red wifi a vincular.
- const char* password: Contraseña de la red wifi a la que se vincula.
- const char* host: Nombre del servidor programado de la Raspberry pi.
- int err: Variable entera del error de conexión.
- float temp, hum: Variables decimales donde se guardarán los valores de temperatura y humedad recogidos por el módulo ESP8266.
- const int httpPort = 80: Configuración del puerto 80 en adelante.
- String url = "/dht11.php": Valor de la url donde ver los resultados en el servidor.
- String key = "pass=1234": Contraseña de acceso al servidor.
- String dato1 = "&Temperatura=": Configuración de la columna temperatura en el servidor, donde llegaran los valores registrados por el DHT11.
- String dato2 = "&Humedad=":Configuración de la columna humedad en el servidor, donde llegaran los valores registrados por el DHT11.
- int humidity = dht.readHumidity(): Variable entera, donde se guardara la lectura de la humedad leída por el sensor DHT11.
- int temp = dht.readTemperature(): Variable entera, donde se guardara la lectura en grados Celsius de la temperatura leída por el sensor DHT11.
- int f = dht.readTemperature(true): Variable de temperatura en Fahrenheit.
- int hi_f = dht.computeHeatIndex(f,humidity): Variable entera sobre el índice de calor en Fahrenheit recogido por el sensor DHT11.
- int heat_index =(hi_f-32)*5/9: Fórmula para pasar el valor del índice de calor en grados Celsius.
- char Msg[30]: Variable para guardar un mensaje no mayor a 30 caracteres, donde se va a configurar para mostrar los valores de temperatura y humedad recogidos por el sensor DHT11.

2.4 DECLARACIÓN DE FUNCIONES

Recordar que la estructura general de cualquier función en lenguaje C consta de tres partes:

1. Tipo: Toda función que posea valor de retorno debe declararse del mismo tipo que dicho valor. En caso de no devolver ningún valor, se declarará del tipo void.
2. Nombre: Conjunto de caracteres alfanuméricos que identifican a la función.
3. Argumentos: Conjunto de datos que acepta la función. Se deberá especificar el número de argumentos, así como el tipo de estos. En caso de no aceptar datos, el argumento deberá definirse como void.

Las funciones que se han utilizado han sido las siguientes:

- void setup()
- void loop()
- void lcd_display()

2.4.1 FUNCIÓN VOID SETUP()

Esta función se encarga de configurar los puertos del microcontrolador, puertos de entrada y salida, timers, etc... Al tratarse de una función exclusivamente de configuración, no precisa de ningún tipo de argumento, ni devuelve dato alguno una vez finalizada su ejecución.

Como la programación en el IDE Arduino se ha divido en tres programas llamados: ESP8266, Rx_receptor y Tx_emisor se va colocar los void setup() correspondientes a los tres programas.

MÓDULO WIFI ESP8266 SERVIDOR LAMP

```
Voidsetup(){
Serial.begin(9600);
//EMPIEZA A CONECTAR CON LA RED WIFI
Serial.println();
Serial.println();
Serial.print("Conectando a ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        }
Serial.println("");
Serial.println("WiFi conectado");
Serial.println("Direccion IP: ");
Serial.println(WiFi.localIP());
}
```

Rx_receptor

```
Voidsetup() {
Serial.begin(9600);
lcd.begin(20,4); //CONFIGURACION DEL NUMERO DE LINEAS Y COLUMNAS
lcd.backlight(); //LUZ DE LA LCD ON
pinMode(led, OUTPUT);
//BITS POR SEGUNDO
vw_setup(2000);
//PIN AL QUE CONECTAR EL MODULO RECEPTOR
vw_set_rx_pin(2);
vw_rx_start();
lcd.begin(20,4);
lcd.backlight();
lcd.createChar(1, thermometer);
lcd.createChar(2, droplet);
lcd.createChar(3,hi);
lcd.clear(); //LIMPIAMOS LA PANTALLA
}
```

Tx emisor

```
Voidsetup() {
Serial.begin(9600);
dht.begin();  //INICIALIZAMOS EL SENSOR
//pinMode(ledPin,OUTPUT);
//CONFIGRUACION DEL VIRTUALWIRE
vw_setup(2000); //NUMERO DE BITS POR SEGUNDO
vw_set_tx_pin(2);//ELEGIMOS EL PIN TX, QUE POR DEFECTO ES EL 12
}
```

2.4.2 FUNCIÓN VOID LOOP()

Este es el programa main o principal de la función. Esta función es un ciclo while infinito, es decir, se reproducirá una y otra vez.

A continuación, se expondrán las distintas líneas de código que conforman el programa principal, agrupadas en bloques dependiendo de su funcionalidad. En ellas se harán constantes llamadas a las funciones descritas anteriormente.

Lo mismo que ocurre con la programación del void setup() que se divide en tres programas, se tiene con void loop(). Es por ello que se divide de la misma forma.

MÓDULO WIFI ESP8266 SERVIDOR LAMP

```
Void loop(){
//INCIALIZAMOS LAS VARIABLES DE TEMPERATURA, HUMEDAD Y ERROR
int err;
float temp, hum;
if ((err = dht11.read(hum, temp)) == 0){
        Serial.print("Temperatura: ");
        Serial.print(temp);
        Serial.print("Humedad: ");
        Serial.print(hum);
        Serial.println();
        }
else{
        Serial.println();
        Serial.print("Error Num :");
        Serial.print(err);
        Serial.println();
        }
Serial.print("Conectando a ");
Serial.println(host);
//USAMOS WIFICLIENT PARA CREAR UNA CONEXION TCP
WiFiClient client;
const int httpPort = 80;
```

```
if (!client.connect(host, httpPort)) {
Serial.println("Conexion fallida");
return;
}
//CREAMOS UNA URL PARA LAS PETICIONES
String url = "/dht11.php";
String key = "?pass=1234";
String dato1 = "&Temperatura=";
String dato2 = "&Humedad=";
Serial.print("Requesting URL: ");
Serial.println(url);
//ENVIAMOS LAS PETICIONES AL SERVIDOR RASPBERRY PI
client.print(String("GET ") + url + key + dato1 + temp + dato2 + hum + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" +
"Connection: close\r\n\r\n");
  unsigned long timeout = millis();
while (client.available() == 0){
        if (millis() - timeout > 5000) {
                Serial.println(">>> Client Timeout !");
                client.stop();
                return;
                }
        }
//LEE TODAS LAS LINEAS DEL SERVIDOR E IMPRIME EN EL PUERTO SERIE
while (client.available()){
        String line = client.readStringUntil('\r');
        Serial.print(line);
        }
Serial.println();
Serial.println("closing connection");
```

```
delay(60000);
}
```

Rx receptor

```
Voidloop(){
uint8_t buf[VW_MAX_MESSAGE_LEN];
uint8_t buflen = VW_MAX_MESSAGE_LEN;
if (vw_get_message(buf, &buflen)){
        digitalWrite(led, HIGH);
        delay(100);
        int i;
        for (i = 0; i < buflen; i++){
                Serial.print((char)buf[i]);
                MsgReceived[i] = char(buf[i]);
                //Serial.print(MsgReceived[i]);
                }
        sscanf(MsgReceived, "%d,%d,%d",&humidity, &temp,&heat_index);
        digitalWrite(led, LOW);
        lcd_display();
        memset( MsgReceived, 0, sizeof(MsgReceived));
        }
}
```

Tx emisor

```
voidloop() {
delay(5000);
//LEEMOS Y ALMACENAMOS LOS DATOS DEL SENSOR
int humidity = dht.readHumidity();
int temp = dht.readTemperature();
int f = dht.readTemperature(true);
```

```
int hi_f = dht.computeHeatIndex(f,humidity); //INDICE DE CALOR EN FAHRENHEIT
int heat_index =(hi_f-32)*5/9; //CONVERTIMOS EL INDICE DE CALOR A GRADOS CELSIUS
char Msg[30];
sprintf(Msg, "%d,%d,%d", humidity,temp ,heat_index);
//ENCENDEMOS EL LED PARA SABER QUE ESTÁ TRANSMITIENDO
digitalWrite(13, true);
//LowPower.powerDown(SLEEP_250MS, ADC_OFF, BOD_OFF);
Serial.print("Humedad: ");
Serial.print(humidity);
Serial.print(" %\t");
Serial.print("Temperatura: ");
Serial.print(temp);
Serial.print(" *C ");
Serial.print(f);
Serial.print(" *F\t");
Serial.print("Índice de calor: ");
Serial.print(heat_index);
Serial.print(" *C ");
Serial.print("mensaje: ");
Serial.print(Msg);
vw_send((uint8_t *)Msg, strlen(Msg));
// ESPERAMOS HASTA QUE EL MENSAJE COMPLETO SE ENVÍE.
vw_wait_tx();//APAGAMOS LA LUZ UNA VEZ SE HA TRANSMITIDO EL MENSAJE
digitalWrite(13, false);
delay(200);
//PONEMOS 5 MINUTOS EL MODO SLEEP
//COMO LA LIBRERIA LOWPOWER SOPORTA COMO MAXIMO 8S, USAMOS COMO BUCLE EL MAXIMO TIEMPO PERMITIDO EN MODO SLEEP QUE SON 5 MINS
```

```
// 5x60=300
//300/4=75
for(int i=0;i<75;i++){
        LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_OFF);
        //delay(4000);
        }
}
```

2.4.3 FUNCIÓN VOID LCD_DISPLAY()

```
void lcd_display(){
 lcd.setCursor(1,0);
lcd.print(" WEATHER STATION ");
lcd.setCursor(4,1);
lcd.print("TEMP");
lcd.setCursor(9, 1);
lcd.write(1);
lcd.setCursor(11, 1);
lcd.print(temp);
lcd.write(0b11011111);
lcd.print("C");
lcd.setCursor(4,2);
lcd.print("HUM");
lcd.setCursor(9, 2);
lcd.write(2);
lcd.setCursor(11, 2);
lcd.print(humidity);
lcd.print("%");
lcd.setCursor(4,3);
lcd.print("HI");
lcd.setCursor(9, 3);
```

```
lcd.write(3);
lcd.setCursor(11, 3);
lcd.print(heat_index);
lcd.write(0b11011111);
lcd.print("C");
}
```

3. PROGRAMACIÓN EN G

Labview contiene una extensa variedad de herramientas para adquirir, analizar, visualizar y almacenar datos, así como herramientas para ayudar a solucionar problemas en el código que se escriba. Es por ello que se ha desarrollado una pequeña programación en G para la visualización y almacenaje de los datos obtenidos sobre la temperatura, humedad e índice de calor aportados por el sensor DHT11. De una manera sencilla se obtendrá un control del sistema en la que el operario es capaz de controlar únicamente con la ayuda del ordenador.

Esta programación está dividida en dos ventanas: la ventana del panel frontal y el diagrama de bloques. A continuación, se presentará una descripción detallada sobre el contenido de cada una, así como la explicación y el funcionamiento de cada uno de sus componentes.

También se detallarán los bloques utilizados para la vinculación de las placas Arduino con Labview, ya que es una manera sencilla de poder enlazar ambos sistemas utilizando únicamente el sistema de bloques llamado Lynx.

3.1 PANEL FRONTAL Y PROGRAMACIÓN

En el panel frontal se puede encontrar todos los indicadores que hacen referencia al comportamiento del sensor DHT11. También se han añadido una serie de opciones más en las que se pueden realizar bases de datos, guardando todos los datos recogidos en tablas Excel para su posterior visualización o almacenamiento. Es posible hacer pantallazos sobre el propio panel frontal y guardarlo en varios formatos, como pueden ser *.jpeg*, *.bmp* y *.png*.

En caso de que el sensor DHT11 estuviese dando lecturas potencialmente peligrosas, se han realizado en el panel frontal una serie de indicadores luminosos sobre la temperatura y humedad. Cuando el sensor sobrepase los valores límite establecidos por el operador. Los indicadores luminosos se encenderían avisando así de que el sensor estaría trabajando sobre valores peligrosos. Es posible modificar estos valores límite para ir adaptando el sensor para cualquier tipo de utilidad en los que se desee controlar los valores de temperatura y humedad.

Por último, se ha añadido una gráfica donde se puede apreciar el recorrido que realizan los valores de temperatura y humedad, así como los diferentes picos que produciría el sistema en la recogida de lecturas. Se trata de una parte importante del sistema porque permite un control más preciso del comportamiento de ambas lecturas.

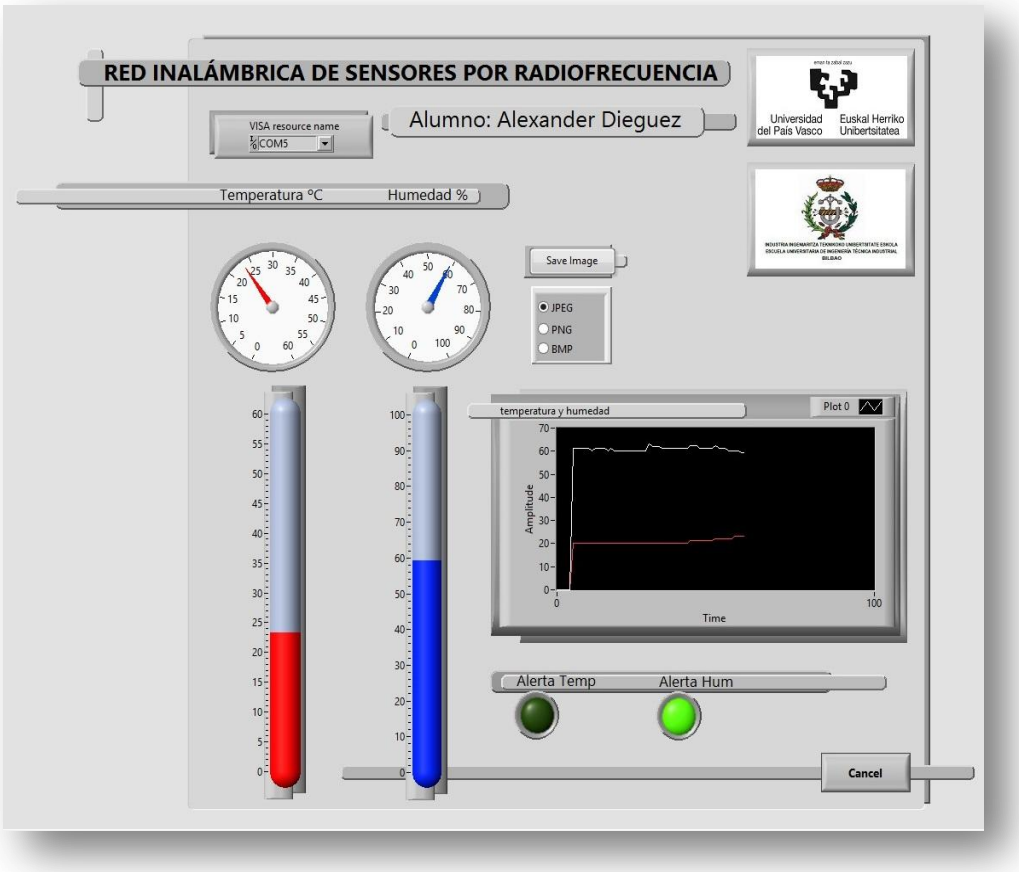

Figura 3.1 Panel frontal del sistema.

En el apartado de programación es donde se encuentra toda la configuración de los bloques. Empezando por uno de los más importantes del sistema, el cual es el encargado de realizar la sincronización con la tarjeta de Arduino. Para ello se han utilizado los bloques llamados *VISA*. Donde se pueden encontrar tanto las lecturas analógicas como digitales, así como la sincronización con el puerto serial del Arduino.

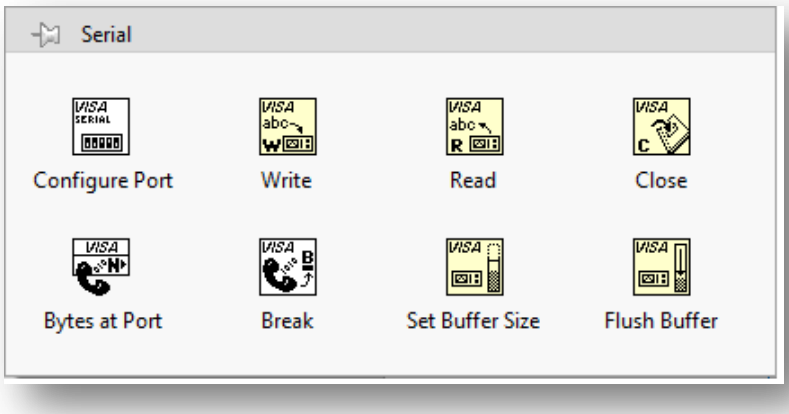

Figura 3.2 Bloques apartado VISA en LabVIEW.

Para la recogida y guardado en diferentes formatos del pantallazo del panel frontal, se ha realizado un bucle con los diferentes estados u opciones que se pueden encontrar en cuanto a la elección del formato de imagen. También se han añadido el botón de guardado de imagen y el selector de formatos.

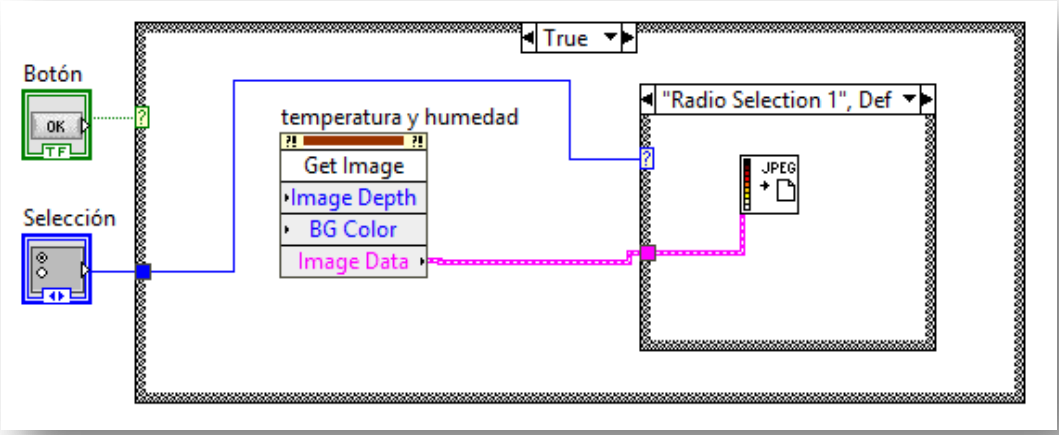

Figura 3.3 Bloques encargados del guardado de imagen en diferentes formatos.

Para realizar las lecturas de la tarjeta Arduino se han realizado con los bloques VISA. Concretamente utilizando los bloques "*VISA Read Function*" y "*Fract/Exp string to Number Function*". Básicamente uno de ellos realiza la lectura analógica y este se divide en 3 variables: temperatura, humedad e índice de calor. Para poder visualizar esta información se tiene que realizar una conversión de *string* a número para poder llevar esta variable a los diferentes indicadores.

Donde se puede apreciar como se recoge la información directamente del bloque de lectura del Arduino es en el bloque llamado *Lectura buffer*, el cual está compuesto por un *string* donde aparecen las lecturas de la temperatura, humedad e índice de calor.

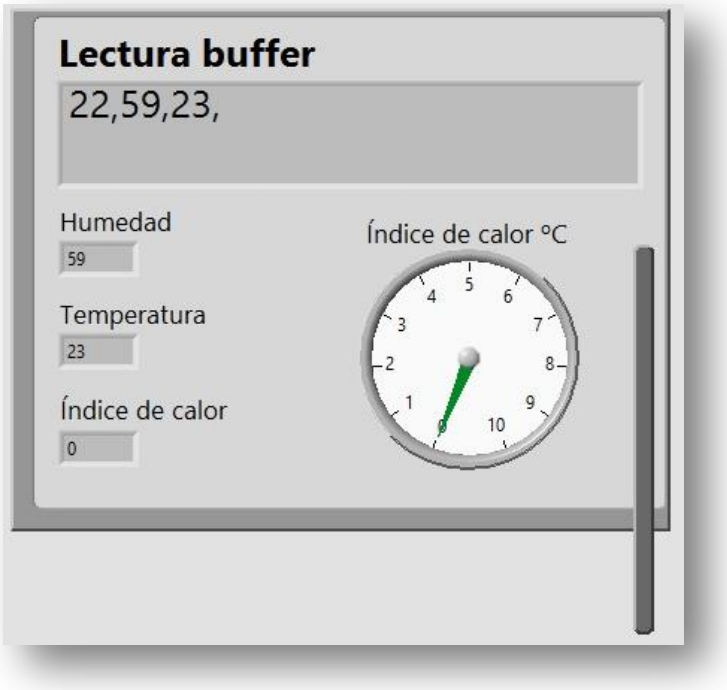

Figura 3.4 Lecturas del buffer

Para poder manipular esa información, hay que indicarle donde empieza y donde acaba cada número recogido. Para ello se utilizan constantes numéricas indicando la posición en la que empieza y cuantos dígitos se quieren recoger. A continuación, una vez realizada la conversión, se llevan a los diferentes indicadores.

Las lecturas del buffer se van refrescando cada cierto periodo de tiempo, el cual hay que configurar para obtener una fluidez en las gráficas y en los indicadores en los que se puedan recoger la información.

También hay que tener en cuenta que cada vez que refresca el buffer las lecturas en la gráfica sobre la humedad y temperatura bajan a cero y vuelven a su magnitud real en un periodo de tiempo muy corto, es decir, se crea un pico cada vez que se refresca. Eso es a causa del delay ocasionado entre las lecturas. Es por ello que para evitar que se produzcan ese tipo de situaciones se tiene que calibrar tanto el tiempo estipulado en el programa de LabVIEW, como los *delays* colocados en el código de programación del IDE Arduino. Es una tarea que requiere tiempo, ya que cada vez que se modifica un parámetro se tiene que comprobar cómo mejora o empeora el sistema, y corregir en función de este.

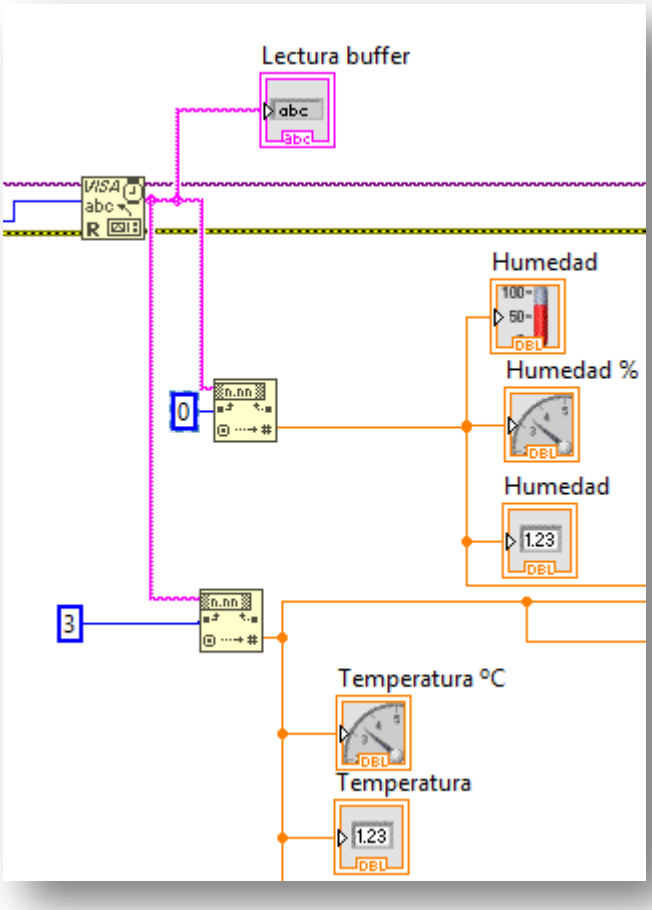

Figura 3.5 Bloques encargados de recoger y transf. las lecturas del sensor.

Uno de los sistemas de bloques importantes en cuanto a la seguridad del sistema, son los encargados de realizar los indicadores luminosos de los límites del sistema, es decir, que cuando el sistema sobrepase los límites indicadores, este devuelva un aviso luminoso al operario. Para ello se ha utilizado un bloque comparador y una constante numérica para indicar el límite que se quiere establecer.

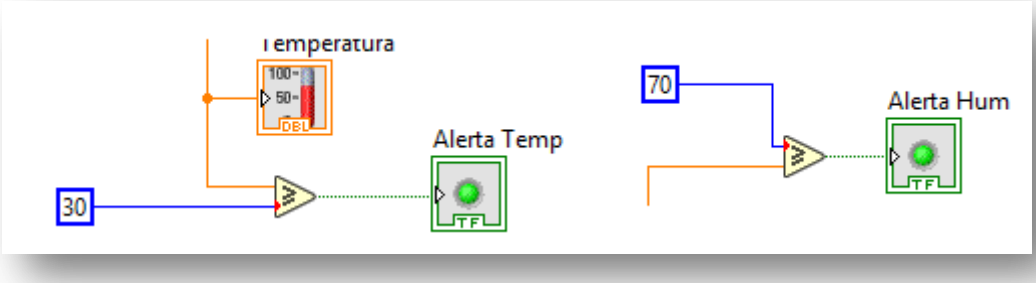

Figura 3.6 Bloques encargados de realizar las alertas del sistema.

Uno de los procesos en cuanto a la recogida de información, es la de realizar una adquisición de datos y poder guardar toda esa información en una tabla de Excel. Todo ello para poder realizar un historial de medidas del sensor y poder realizar un estudio de comportamiento de este. También poder realizar cualquier tipo de consulta de algún día en especial. Se trata de un apartado muy sencillo en la programación de los bloques en LabVIEW, ya que utilizando un único bloque es capaz de realizar esta tarea. El bloque en cuestión se llama "*Write to measurement File*".

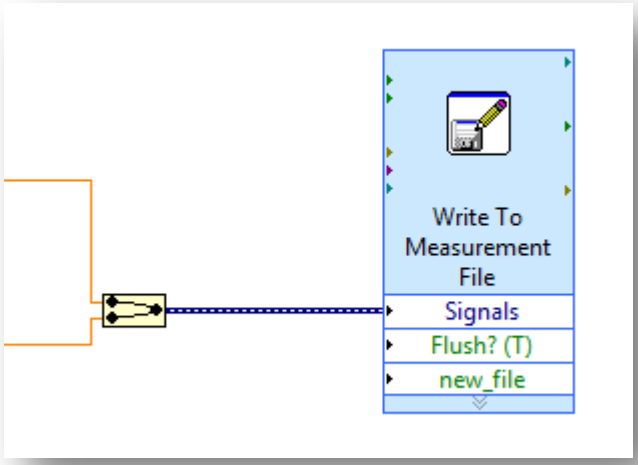

Figura 3.7 Bloque encargado de recoger la información y almacenarla.

Por último, los bloques encargados de visualizar el comportamiento de los datos de temperatura y humedad en una gráfica. Para poder visualizar dos datos en una misma gráfica se ha utilizado el bloque llamado "*Bundle Function*".

Figura 3.8 Bloques encargados de realizar la visualización de la temp. y humedad.

Finalmente se muestra a continuación la imagen de todo el conjunto de bloques programados en LabVIEW. Donde se puede apreciar abajo a la derecha el botón de Stop. Encargado de finalizar de forma instantánea toda comunicación con la tarjeta de Arduino.

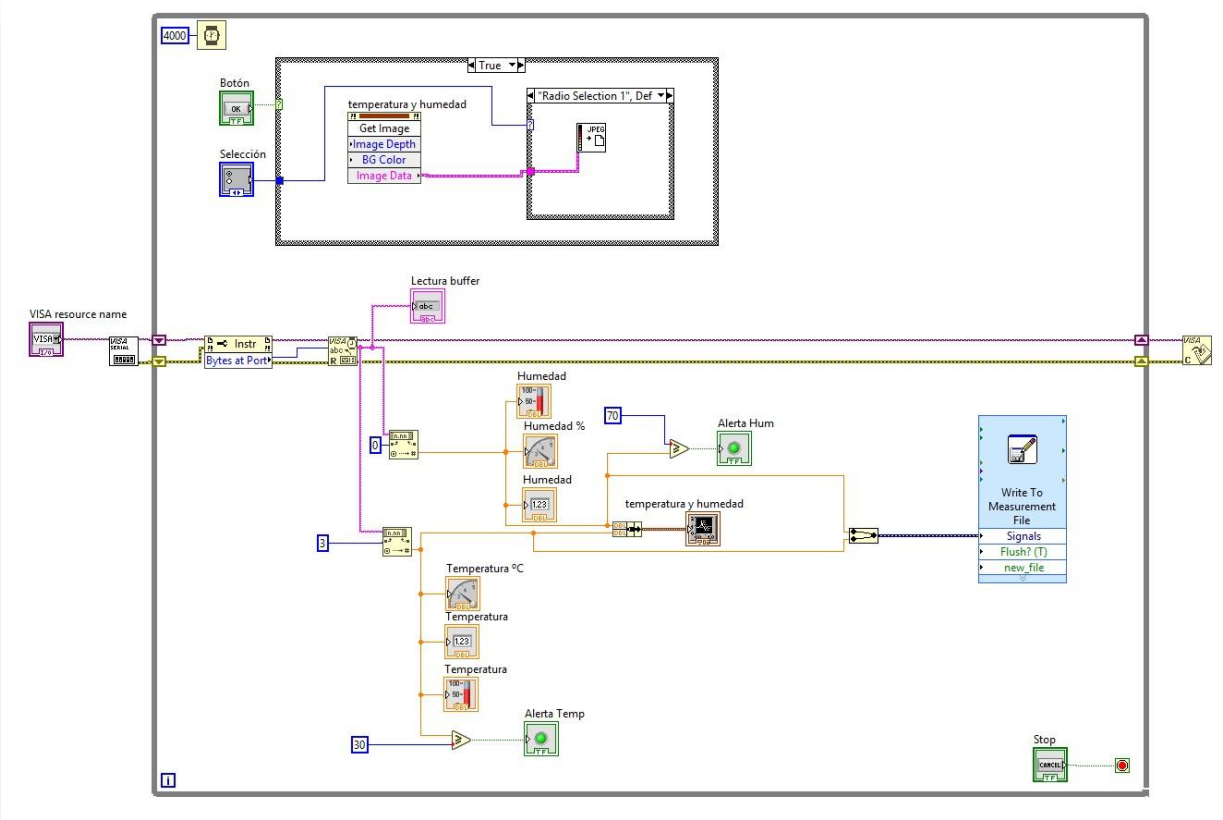

Figura 3.9 Conjunto total de los bloques que componen el sistema.

Hay ocasión de poder mejorar o incorporar nuevos bloques al sistema en función de las necesidades del cliente, como puede añadir un nuevo transductor para realizar un control más completo del entorno donde se coloque el prototipo. También es posible añadir un bloque para transformar la información recogida del excel a otro formato más común, o incluso mandarlo por correo cada cierto periodo de tiempo.

Se trata de un programa base donde los límites de mejora los establece el mercado y las necesidades de los clientes.

3.2 PALETA DE CONTROLES

Se utiliza únicamente en el apartado del panel frontal. Contiene todos los controles e indicadores que se emplearán para crear la interfaz del VI con el usuario.

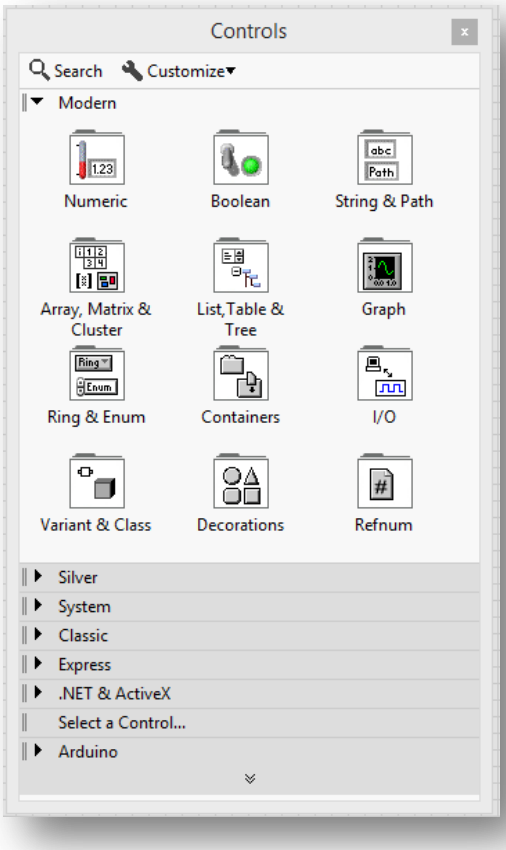

Figura 3.10 Funciones que se encuentran en la paleta de controles.

Dicho menú contiene las siguientes opciones:

- *Numeric:* Para la introducción y visualización de cantidades numéricas.
- *Boolean:* Para la entrada y visualización de valores booleanos.
- *String & Table:* Para la entrada y visualización de texto.
- *List & Ring:* Para la visualizar y/o seleccionar una lista de opciones.
- *Array & Cluster:* Para agrupar elementos.
- *Graph:* Para representar gráficamente los datos.
- *Path & RefNum:* Para gestión de archivos.
- *Decorations:* Para introducir decoraciones en el panel frontal. No visualizan datos.
- *User Controls:* Para elegir un control creado por el propio usuario.

- *ActiveX:* Para transferir datos y programas de unas aplicaciones a otras dentro de Windows.
- *Select a Control:* Para seleccionar cualquier control.

Al seleccionar objetos desde el menú *Controls* estos aparecen sobre el panel frontal, pueden colocarse donde convenga, y además tienen su propio menú desplegable que permite la configuración de algunos parámetros específicos de cada tipo de control.

3.3 PALETA DE FUNCIONES

Se emplea en el diseño del diagrama de bloques. La paleta de funciones contiene todos los objetos que se emplean en la implementación del programa del VI, ya sean funciones aritméticas, de entrada/salida de señales, entrada/salida de datos a fichero, adquisición de señales, temporización de la ejecución del programa.

Para seleccionar una función o estructura concretas, se debe desplegar el menú *Functions* y elegir entre las opciones que aparecen. A continuación, se enumeran todas ellas, junto con una pequeña definición.

- *Structures:* Muestra las estructuras de control del programa, junto con las variables locales y globales.
- *Numeric:* Muestra funciones aritméticas y constantes numéricas.
- *Boolean:* Muestra funciones y constantes lógicas.
- *String:* Muestra funciones para manipular cadenas de caracteres, así como constantes de caracteres.
- *Array:* Contiene funciones útiles para procesar datos en forma de vectores, así como constantes de vectores.
- *Cluster:* Contiene funciones útiles para procesar datos procedentes de gráficas y destinados a ser representados en ellas, así como las correspondientes constantes.
- *Comparison:* Muestra funciones que sirven para comparar números, valores booleanos o cadenas de caracteres.
- *Time & Dialog:* Contiene funciones para trabajar con cuadros de diálogo, introducir contadores y retardos, etc…
- *File I/O:* Muestra funciones para operar con ficheros.
- *Communication:* Muestra diversas funciones que sirven para comunicar varios ordenadores entre sí, o para permitir la comunicación entre distintos programas.
- *Instrument I/O:* Muestra un submenú de VIs, que facilita la comunicación con instrumentos periféricos que siguen la norma ANSI/IEEE 488.2-1987, y el control del puerto serie.
- *Data adquisition:*Contiene a su vez un submenú donde puede elegirse entre distintas librerías referentes a la adquisición de datos.

- *Analysis:* Contiene un submenú en el que se puede elegir entre una amplia gama de funciones matemáticas de análisis.
- *Tutorial:* Incluye un menú de VIs que se utilizan en el manual LabVIEW Tutorial.
- *Advanced:* Contiene diversos submenús que permiten el control de la ayuda, de los VIs, manipulación de datos, procesado de eventos, control de la memoria, empleo de programas ejecutables o incluidos en librerías DLL, etc…
- *Instrument drivers:* En él se muestran los drivers disponibles de distintos instrumentos.
- *User libraries:* Muestra las librerías definidas por el usuario.
- *Aplication control:* Contiene varias funciones que regulan el funcionamiento de la propia aplicación en ejecución.
- *Select a VI:* Permite seleccionar cualquier VI para emplearlo como subVI.

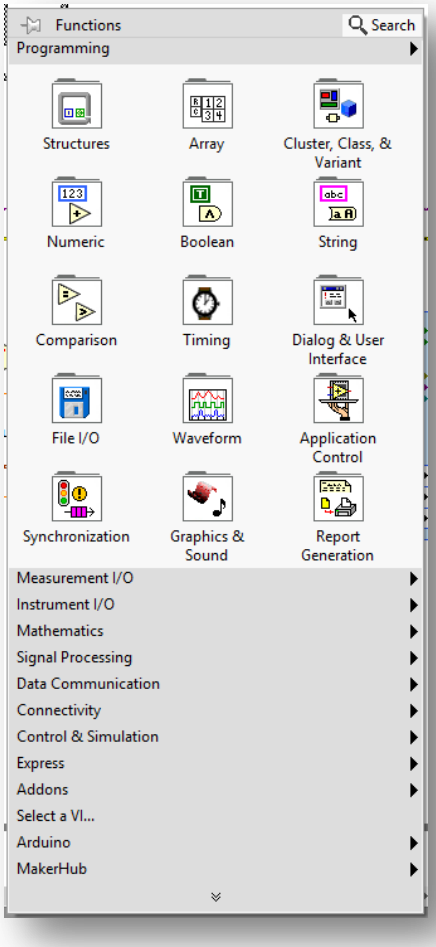

Figura 3.11 Funciones que se encuentran en la paleta de funciones.

4.PROGRAMACIÓN DE BLYNK

Para realizar la adaptación y configuración del sistema en una aplicación móvil, se ha utilizado un programa llamado Blynk. El cual facilita mucho las cosas en cuanto a la comunicación entre el Arduino y el móvil.

Blynk es una aplicación de iOS y Android para controlar Arduino, Raspberry Pi, Módulos Wifi ESP8266 y demás tarjetas a través de Internet. Este último dato es muy importante, ya que el proyecto se basa principalmente en una red de sensores en la que el traspaso de información se realiza de forma inalámbrica.

Está compuesto por un tablero de instrumentos digital, donde se puede construir una interfaz gráfica acorde con las funciones requeridas por el sistema, simplemente arrastrando y soltando los diferentes widgets. Es muy fácil de configurar. Lo curioso de esta aplicación es que no está ligado a ninguna tarjeta de desarrollo o *shield* en específico. Compatibilidad de sistemas operativos y placas desde Windows hasta Linux, MacOS, etc…

A continuación se muestra la programación realizada sobre el módulo ESP8266 para poder configurar la aplicación móvil de Blynk.

```
#define BLYNK_PRINT Serial
#include <SPI.h>
#include <ESP8266WiFi.h>
#include <SimpleTimer.h>
#include <BlynkSimpleEsp8266.h>
#include <DHT.h>
char auth[]= "cb3c7bf506c3427aa86436ad2a55c256";
//Wifi
char ssid[]= "Euskaltel-tax4";
char pass[]= "*********";
#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
SimpleTimer timer;
```

```
void sendSensor(){
float h=dht.readHumidity();
float t=dht.readTemperature();
if (isnan(h) || isnan(t)){
        Serial.println("Fallo de lectura del sensor DHT");
        return;
        }
Blynk.virtualWrite(V5, h);
Blynk.virtualWrite(V6, t);
}

void setup(){
Serial.begin(9600);
Blynk.begin(auth, ssid, pass);
dht.begin();
timer.setInterval(1000L, sendSensor);
}

void loop(){
Blynk.run();
timer.run();
}
```

Una vez realizada la programación, se configura desde la aplicación móvil la interface, es decir, los distintos indicadores. Todo ello teniendo en cuenta los valores introducidos en la programación de Arduino, para poder sincronizar los valores a los indicadores correspondientes.

En caso de querer mejorar o configurar de forma más exhaustiva algún tipo de indicador como podría ser la pantalla LCD, la propia aplicación dispone de un pequeño menú de ayuda, donde se compone de ciertos códigos de programación según sea la mejora a aplicar en ella. Obviamente que esas líneas de código se deberían de implementar en la IDE de Arduino y volver a compilar y volcar en el módulo ESP8266.

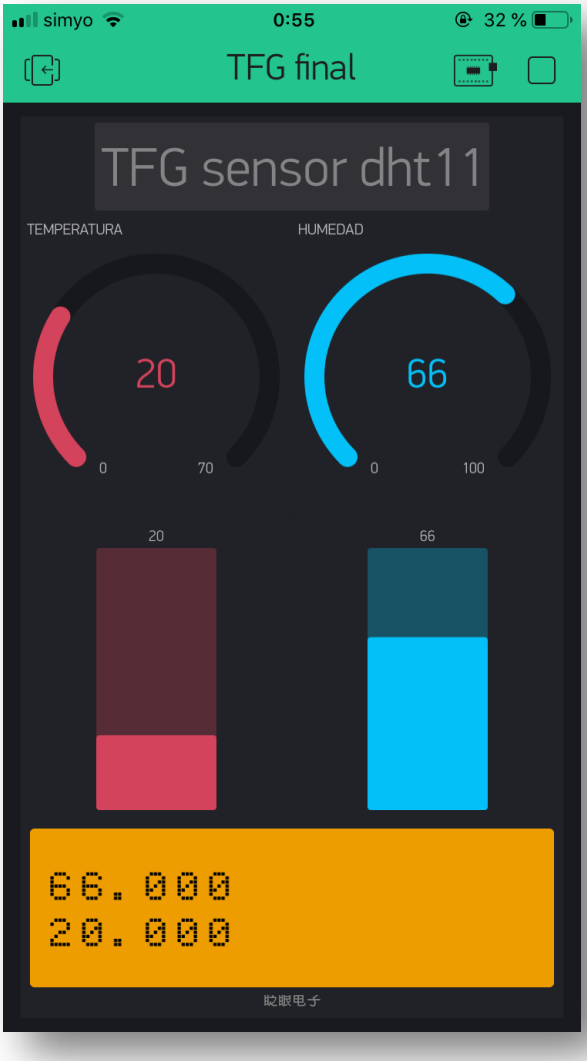

Figura 4.1 Interface de la aplicación Blynk

5. BUS I2C PARA LCD

Para realizar la comunicación de forma que funcionen todos los elementos electrónicos del sistema se propuso una norma de comunicación digital. Una norma que especifica la velocidad, niveles de tensión, y el protocolo a seguir para conseguir esa comunicación. A esa norma se le denominó I2C (*Inter Integrated Circuits bus*) y está basado en las siguientes características:

- Protocolo de dos hilos de control, uno para transmitir los datos, SDA y otro, el reloj asíncrono que indica cuando leer los datos SCL, mas GND y 5V.

- Cada dispositivo conectado al bus I2C y cada uno tiene su dirección exclusiva de 7 bits, así que es posible conectar hasta 2^7=128 dispositivos.
- Uno de estos componentes debe actuar como *master*, es decir el que controla el reloj.
- No se requiere una velocidad de reloj estricta, ya que es el *master* quien controla el *Clock*.
- Es *multi-master*, pero solo uno puede estar activo a la vez, y proporciona un protocolo de arbitraje y detección de colisiones

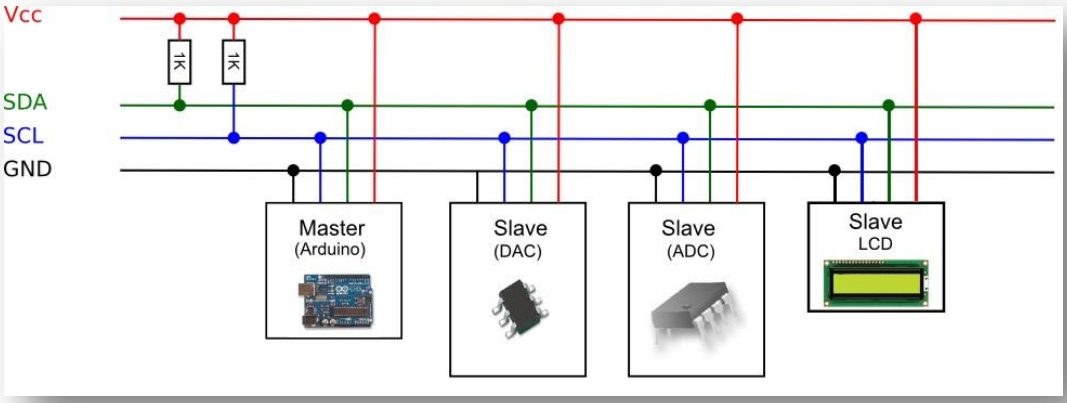

Figura 4.2 Esquema sobre el bus I2C

El propio Arduino es capaz de soportar de fábrica mediante una librería estándar, que utiliza dos de los pines analógicos para las funciones SDA (datos) y SCL (clock). La librería I2C en Arduino se llama *wire*, y gestiona el protocolo de comunicaciones completo.

6. CONCLUSIONES

Es posible obtener una serie de desviaciones entre las medidas obtenidas de los diferentes métodos realizados. Se puede mejorar realizando una serie de calibraciones en todos los sistemas, intentado obtener la medida más precisa posible. Para ello es necesario la implementación física para evaluar el comportamiento de la manera más real posible, y poder así tener en cuenta los factores externos que pueden verse involucrados en el sistema. Una vez compilada toda esa información se procedería a calibrar el sistema de manera mucho más eficiente.