Universidad Politécnica de Madrid

Grado en Ingeniería Electrónica y Automática

Asignatura

SISTEMAS ELECTRÓNICOS DIGITALES

TRABAJO MICROCONTROLADOR

Control de Riego

Curso 2021-2022

Julio Augusto Arbizu García 52070

Adrián Rodríguez Fernández 54829

ÍNDICE

Contenido

INTRODUCCIÓN

En este documento se expone el proceso que hemos seguido para desarrollar el trabajo de microcontroladores. Nuestro equipo ha elegido como tema el control de riego de un invernadero. Hemos hecho algunas modificaciones a la propuesta inicial. Por ejemplo, para intentar acercarnos a la realidad, hemos tenido en cuenta diversos factores ambientales como la temperatura, humedad, luminosidad, etc.

Para ejecutar el programa utilizaremos la placa *STM32F407* y el entorno de trabajo *STM32CubeIDE*, así como distintos periféricos como sensores y actuadores, que se detallarán más adelante.

Respecto a la división del trabajo entre los distintos componentes del grupo, al ser solo dos hemos trabajado conjuntamente la mayor del tiempo, tanto de forma presencial, para el montaje de la maqueta, como de forma online para la programación.

A continuación, se expone el link al repositorio *git* utilizado:
https://github.com/Adrofe/Trabajo-MICROS

DISEÑO DEL SISTEMA

Lo primero que hicimos fue definir las funcionalidades que queríamos en nuestro sistema. Como se trata de un control de riego, tendríamos un modo manual que se activaría pulsando un botón, y un modo automático, que decidimos que sería en función de la luminosidad ambiental, para simular que se regaría de noche automáticamente el invernadero.

También pensamos que sería buena idea controlar el nivel de agua del recipiente donde se encontraría la bomba, para impedir el riego si no hay agua suficiente. Por otra parte, pensamos utilizar un sensor de humedad para saber si el ambiente es húmedo e impedir el riego, para simular que si ha llovido no se riegue la planta. El sensor que conseguimos es el *DHT11*, que además de ser sensor de humedad también lo es de temperatura, por lo se nos ocurrió poner un motor *DC* con una hélice que se activa si aumenta la temperatura por encima de cierto umbral, recreando un control de temperatura dentro del invernadero. Por último, queríamos representar los datos de este último sensor en algún *display*, por lo que conseguimos una *LCD* para representar la humedad y temperatura.

En el apartado de Montaje se explicará cada uno de los sensores y actuadores utilizados más en profundidad.

CASOS DE USO

A continuación, se expondrán todas las posibles situaciones que pueden darse sobre el sistema y como el control de este responderá a ellas:

- Si la temperatura o la humedad varían, estas se muestran en la pantalla *LCD*.
- Si hay agua en el tanque se enciende el *led* azul de la placa.
- Si no hay agua en el tanque se enciende el *led* rojo de la placa.
- Si se aprieta el botón de riego manual, hay agua en el tanque y la humedad está dentro de los parámetros de entorno seco, se inicia el riego.

- Si se aprieta el botón de riego manual, hay agua en el tanque y la humedad no está dentro de los parámetros de entorno seco, no se inicia el riego.
- Si se aprieta el botón de riego manual y no hay agua en el tanque, no se inicia el riego y se mostrará en la pantalla el mensaje "No hay agua".
- Si se aprieta el botón de riego manual y la bomba ya está en funcionamiento, no se reinicia el riego y se mostrará en la pantalla el mensaje "La bomba ya está *ON*".
- Si los valores de luminosidad indican que es de noche se encenderá el *led* de noche.
- Si los valores de luminosidad no indican que es de noche no se encenderá el *led* de noche.
- Si los valores de luminosidad indican que es de noche, la bomba no está ya en funcionamiento, los valores de humedad indican entorno seco y hay agua en el tanque, se inicia el riego automático.
- Si los valores de luminosidad indican que es de noche, la bomba no está ya en funcionamiento, los valores de humedad indican entorno no seco y hay agua en el tanque, no se inicia el riego automático.
- Si los valores de luminosidad indican que es de noche, la bomba no está ya en funcionamiento, los valores de humedad indican entorno seco y no hay agua en el tanque, no se inicia el riego automático y se mostrará en la pantalla en mensaje "No hay agua".
- Si los valores de luminosidad no indican que es de noche, no se inicia el riego automático.
- Si los valores de humedad no indican entorno seco y la bomba está activa, esta se parará.
- Si la temperatura sube de los valores deseados, el ventilador se encenderá.
- Si la temperatura no sube de los valores deseados, el ventilador no se encenderá.
- Si la temperatura baja dentro de los valores deseados y el ventilador está en funcionamiento, este se detendrá.
- Si la temperatura no baja dentro de los valores deseados y el ventilador está en funcionamiento, este no se detendrá.

MONTAJE

Hemos montado una maqueta para simular el comportamiento de nuestro sistema. Junto a la maceta y al tanque de agua (un tarro de cristal) hemos utilizado los siguientes componentes:

1. Salidas y Actuadores

Bomba de agua

Para la bomba de agua hemos comprado una pequeña, utilizada para peceras, que trabaja con un rango de voltaje de 3 a 5 V y consume una corriente entre 100 mA y 200 mA. Primeramente, pusimos su entrada positiva conectada directamente al micro, pero nos dimos cuenta de que no funcionaba así que la controlamos con un transistor, que el micro encendía o apagaba a modo de interruptor. Sin embargo, cuando conectamos la *LCD* vimos que introducía mucho ruido y corrompía la señal, por lo que optamos por controlarla con el chip *L293D*, que se explicará más adelante en esta sección.

Figura 1. Bomba de agua

Motor DC (hélice)

El motor de *DC*, tiene un rango de voltaje de 3 a 6 V y consume también mucha corriente, por lo que al igual que con la bomba de agua, la controlamos con el *L293D*.

Figura 2. Motor DC con hélice

LCD 1602

Utilizamos la *LCD* para mostrar por pantalla diversos mensajes, así como la temperatura y la humedad. Tiene los siguientes pines:

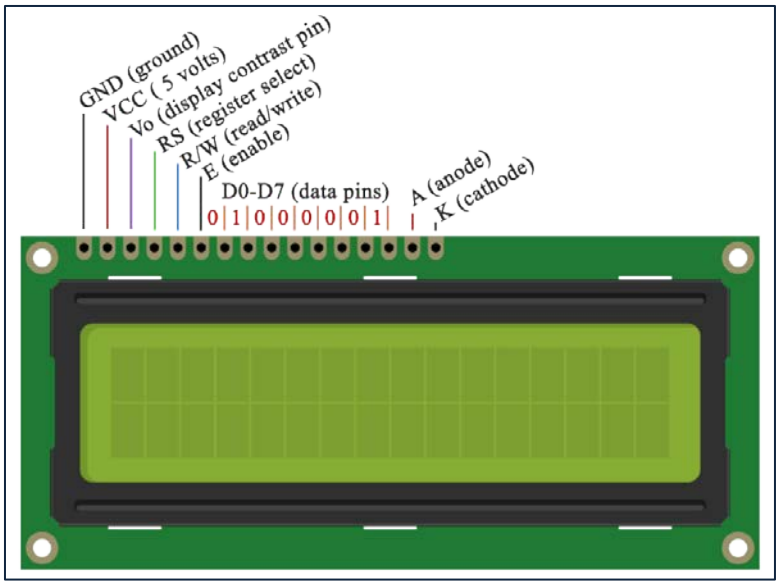

Figura 3. Pantalla LCD

El primer pin va a tierra y el segundo a V_{cc}, en nuestro caso 5 voltios. El siguiente pin es V_o, que va a un potenciómetro para ajustar el contraste. Después está *RS*, que selecciona el registro a leer o escribir, que está conectado como una salida del micro. El siguiente es el pin *R/W*, que lo conectamos a tierra ya que solo queremos escribir en la pantalla. El siguiente pin es el *Enable*, que al igual que el *RS* va al micro. A continuación, están 8 pines de datos, que van al micro y son los que utilizamos para mandar la información a mostrar en la pantalla. Finalmente están los ánodos y cátodos de los *LEDs* internos que van a 5 V y tierra respectivamente.

LEDs microcontrolador

Por último, respecto a las salidas, utilizamos los *LEDs* incorporados en el microcontrolador para monitorizar diversos estados, como el nivel de agua o de luz. En nuestro caso utilizaremos el *led* azul, rojo y naranja.

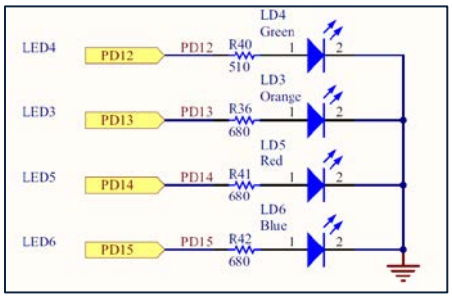

Figura 4. LEDs internos

2. Entradas y Sensores

LDR

La *LDR* es una fotorresistencia, que varía su resistencia en función de la luz. Por ello, medimos utilizando un divisor de tensión, conectando el voltaje intermedio al microcontrolador, donde será digitalizada mediante un conversor analógico-digital. Como V_{cc} utilizamos 3.3 V mediante el módulo de alimentación, que será tratado posteriormente. Este sensor lo utilizamos para que se riegue automáticamente si la luminosidad baja de cierto umbral.

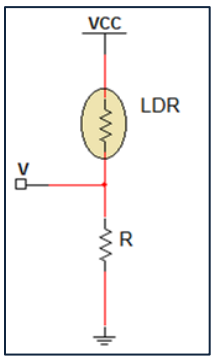

Figura 5. Divisor de tensión LDR

Sensor Nivel Agua

El sensor de nivel de agua tiene una línea de conductores que varían su resistencia en función del nivel de agua. Tiene tres patillas, dos para la alimentación y la señal, que se conecta al micro para convertir mediante un *ADC* esta señal. El sensor también puede funcionar de modo digital (1 hay agua y 0 no), pero decidimos que su modo analógico es mejor para poder medir el nivel de agua que consume la bomba con mejor precisión.

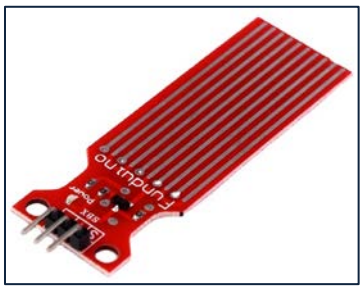

Figura 6. Sensor de nivel de agua

Pulsador microcontrolador

Utilizamos el pulsador incorporado en la placa para activar el riego en modo manual por medio de interrupciones.

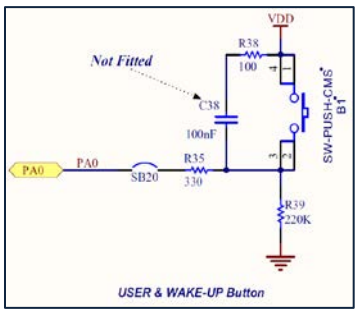

Figura 7. Pulsador microcontrolador

DHT11

El sensor de humedad y temperatura tiene tres pines, dos para la alimentación (5 V y tierra) y otro para los datos. Tiene una resolución de ± 1 ºC y ± 4% de *RH (relative humidity)*. Este sensor funciona de forma peculiar, y hemos tenido que crear una serie de funciones especiales. Mirando la hoja de características vemos que una transmisión consta de 40 bits, que significan lo siguiente:

- 8 *bits* para la parte entera de la humedad
- 8 *bits* para la parte decimal de la humedad (en nuestro caso 0, ya que no es tan preciso el sensor, pero hay otros modelos superiores DHT22 que si lo utilizan)
- 8 *bits* para la parte entera de la temperatura
- 8 *bits* para la parte decimal de la temperatura (pasa lo mismo que con la parte decimal de la humedad)
- 8 *bits* para la suma de todos, esto se usa para comprobar que la transmisión es correcta

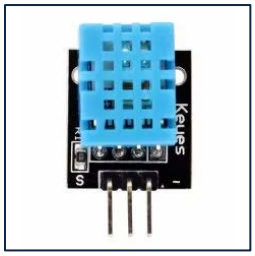

Figura 8. Sensor de humedad y temperatura

El sensor funciona como una comunicación serie, pero solo utilizando un solo canal que se intercambia entre *input* y *outp*ut según la siguiente secuencia:

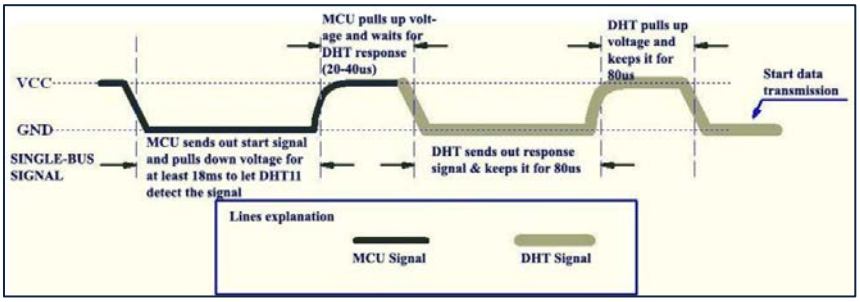

Figura 9. Comienzo de la comunicación

Primero desde el micro ponemos la línea a 0 durante al menos 18 ms, para que el sensor detecte la señal. Después volvemos a poner la línea a 1 y esperamos la respuesta del sensor que tarda entre 20 y 40 microsegundos, tras lo cual el sensor vuelve a poner la línea a 0 durante 80 microsegundos, a 1 tras otros 80 microsegundos y después comienza la transmisión de datos.

Antes de transmitir un dato la línea se pone a 0 durante 50 microsegundos, y después se transmite el valor de ese bit que depende de la duración del pulso alto. Por ejemplo:

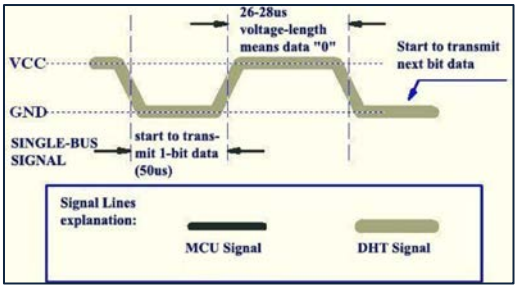

Figura 10. Onda 0 lógico

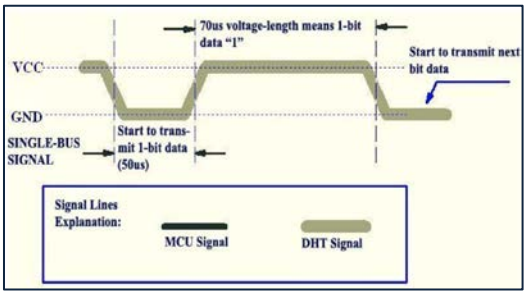

Figura 11. Onda 1 lógico

Si el pulso alto dura entre 26 y 28 microsegundos significa que el dato es un 0 lógico, y si dura 70 microsegundos es un 1 lógico. De esta forma se transmiten los 40 *bits*, lo que tarda aproximadamente unos 4 milisegundos. En la *datasheet* el fabricante nos recomienda no hacer más de una muestra por segundo.

3. Otros chips y módulos

L293D

Al principio conectamos los motores directamente al micro y pronto vimos que esto no funcionaba, por lo que se nos ocurrió controlar el apagado y apagado de los mismos a través de un transistor funcionando como interruptor. Sin embargo, comprobamos que de esta manera se introducía mucho ruido en el micro y dejaban de funcionar otros elementos como la *LCD* o los sensores. Por lo que estuvimos buscando una solución y encontramos este chip.

Figura 12. Chip L293D

El *L293D* es un dispositivo diseñado para controlar motores de hasta 36 V, con alta inmunidad al ruido y una corriente de hasta 600 mA, más que de sobra para nuestros actuadores. El chip es simétrico y nos permite controlar dos motores con la misma fuente de tensión, que nosotros controlamos la bomba y la hélice. Tiene la siguiente interfaz:

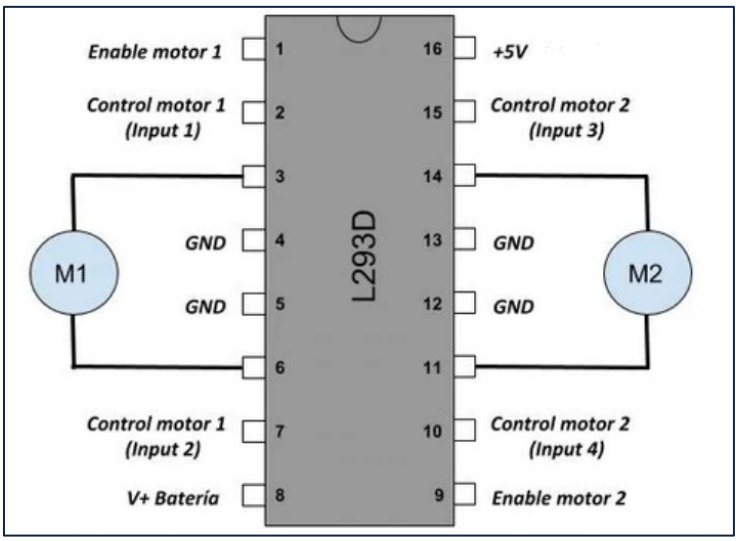

Figura 13. Esquemático L293D

Como podemos observar, tiene dos entradas de alimentación, una de 5 V, para alimentar internamente al dispositivo, y otra llamada V+ Batería que es la que alimenta a los motores. Posee cuatro entradas que se tienen que conectar a tierra. Asimismo, tiene dos entradas de *Enable*, para poder parar los motores, que nosotros conectamos a un pulsador para tener un sistema de parada de emergencia. La patilla 3 y 6 se conectan al primer motor y la 11 y 14 al segundo.

Por último, quedan cuatro patillas llamadas *Input* 1, 2, 3 y 4, que se utilizan para gobernar al motor. Cada motor tiene dos, que nos permiten indicar el sentido de giro del mismo. Como en nuestro caso solo girará en un sentido, solo una entrada por cada motor se conecta al micro, que es el que indica cuando tiene que encenderse.

Pila 4,5 V

Debido a los requerimientos especiales de los motores respecto a la corriente, hemos optado por separar la alimentación de los sensores y la *LCD* de estos. Por ello hemos comprado una pila de 4,5 V para alimentar el *L293D*.

Figura 14. Pila 4,5 V

Módulo de alimentación

Gracias a este módulo que enchufamos a la red eléctrica podemos tener por separado una tensión de 5 y 3.3 V, que utilizamos para alimentar tanto a la LCD como a los sensores (agua, luz, *DHT11*) según sus necesidades.

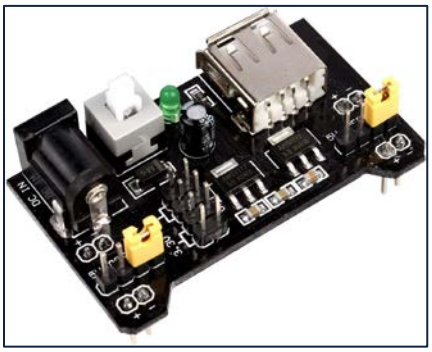

Figura 15. Fuente de alimentación externa

4. Montaje final

Una vez obtenidos todos los dispositivos necesarios, el montaje ha sido sencillo, aunque un poco tedioso ya que son muchos cables y hay que prestar mucha atención a donde se conectan, para que no haya errores. Para pegar las cosas hemos utilizado tanto celo como silicona y un soldador de estaño para cables y uniones. A continuación, se adjunta una foto del resultado final:

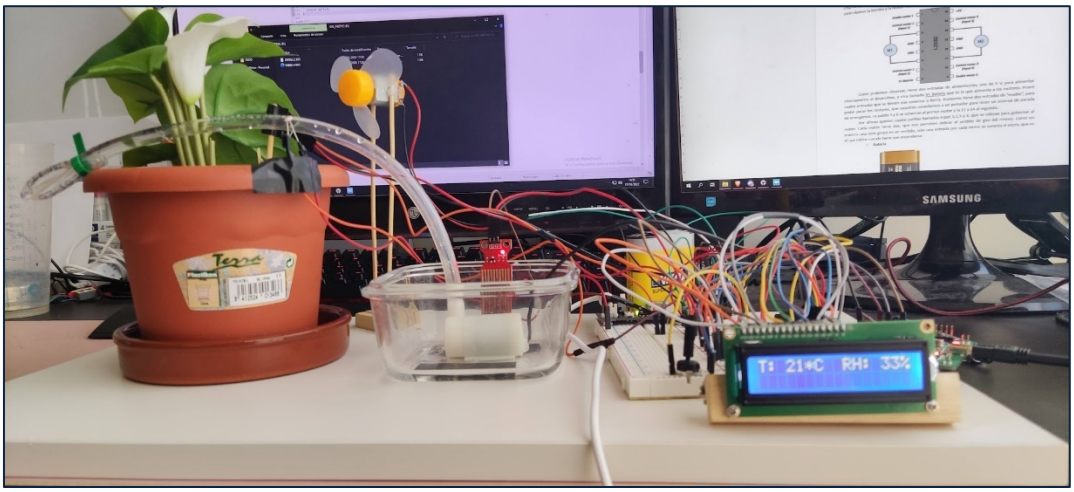

Ilustración 16 Montaje Final

PROGRAMACIÓN

En este punto se explicará el funcionamiento de cada una de las partes del código, haciendo hincapié en las funcionalidades más importantes, las distintas interconexiones entre las mismas y las decisiones tomadas a la hora de solventar los problemas que han ido surgiendo.

1. Bibliotecas

stdbool.h

Biblioteca necesaria en para poder usar variables del tipo *bool*.

STM_MY_LCD16X2.h

Esta biblioteca gestiona las funciones internas necesarias para controlar el funcionamiento y las comunicaciones de la pantalla *LCD*. Entre las múltiples funciones que posee son de especial importancia:

- **void LCD1602_clear(void)**

Esta función se encarga de limpiar los datos de la pantalla para poder enviar otros nuevos.

- **void LCD1602_print(char string[])**

Esta función recibe una cadena de caracteres que imprime por la pantalla *LCD*.

- **void LCD1602_PrintInt(int number)**

Esta función recibe un número entero que imprime por la pantalla *LCD*.

- LCD1602_PrintFloat(float number)

Esta función recibe un número con dos decimales que multiplica por cien, convierte las componentes de este número en enteros y va guardándolos en un vector el cual envía digito por digito a la función *LCD1602_PrintInt(number).* Se desarrollo está función para imprimir *float*, pero después nos dimos cuenta de que la resolución del sensor no era tan alta y finalmente no se utilizó.

```
void LCD1602_PrintFloat(float number)
{
	int num[5];
	number = number * 100;
	num[0] = (int)number / 10000;
	number = (int)number % 10000;
	num[1] = (int)number / 1000;
	number = (int)number % 1000;
	num[2] = (int)number / 100;
	number = (int)number % 100;
	num[3] = (int)number / 10;
	number = (int)number % 10;
	num[4] = (int)number;

	if (num[0] == 0) {
		if (num[1] == 0) {
			if (num[2] == 0) {
				LCD1602_PrintInt(0);
				LCD1602_print(".");
				for (int i = 3; i < 5; i++) {
					LCD1602_PrintInt(num[i]);
				}
			}
			else {
				for (int i = 2; i < 5; i++) {
					LCD1602_PrintInt(num[i]);
					if (i == 2) LCD1602_print(".");
				}
			}
		}
		else {
			for (int i = 1; i < 5; i++) {
				LCD1602_PrintInt(num[i]);
				if (i == 2) LCD1602_print(".");
			}
		}
	}
	else {
		for (int i = 0; i < 5; i++) {
			printf(num[i]);
			if (i == 2) LCD1602_print(".");
		}
	}
	}
```

- void LCD1602_Begin4BIT(GPIO_TypeDef* PORT_RS_E, uint16_t RS, uint16_t E, GPIO_TypeDef* PORT_MSBs4to7, uint16_t D4, uint16_t D5, uint16_t D6, uint16_t D7)

Esta función se encarga de asignar internamente todos los puertos de salida necesarios para el correcto funcionamiento de la pantalla *LCD*.

```
void LCD1602_Begin4BIT(GPIO_TypeDef* PORT_RS_E, uint16_t RS, uint16_t E,
GPIO_TypeDef* PORT_MSBs4to7, uint16_t D4, uint16_t D5, uint16_t D6, uint16_t D7)
{
	//Set GPIO Ports and Pins data
	PORT_RS_and_E = PORT_RS_E;
	PIN_RS = RS;
	PIN_E = E;
	PORT_MSB = PORT_MSBs4to7;
	D4_PIN = D4;
	D5_PIN = D5;
	D6_PIN = D6;
	D7_PIN = D7;
	//Initialise microsecond timer
	LCD1602_TIM_Config();
	//Set the mode to 4 bits
	mode_8_4_I2C = 2;
	//Function set variable to 4 bits mode
	FunctionSet = 0x28;

	//Initialise LCD
	//1. Wait at least 15ms
	HAL_Delay(20);
	//2. Attentions sequence
	LCD1602_write4bitCommand(0x3);
	HAL_Delay(5);
	LCD1602_write4bitCommand(0x3);
	HAL_Delay(1);
	LCD1602_write4bitCommand(0x3);
	HAL_Delay(1);
	LCD1602_write4bitCommand(0x2);  //4 bit mode
	HAL_Delay(1);
	//3. Display control (Display ON, Cursor ON, blink cursor)
	LCD1602_writeCommand(LCD_DISPLAYCONTROL | LCD_DISPLAY_B | LCD_DISPLAY_C |
LCD_DISPLAY_D);
	//4. Clear LCD and return home
	LCD1602_writeCommand(LCD_CLEARDISPLAY);
	HAL_Delay(3);
	//4. Function set; Enable 2 lines, Data length to 8 bits
	LCD1602_writeCommand(LCD_FUNCTIONSET | LCD_FUNCTION_N);
	HAL_Delay(3);
}
```

- void LCD1602_2ndLine(void)

Esta función hace que los datos recibidos a continuación de su sentencia se envíen a la segunda línea de la pantalla *LCD*.

2. Defines

- #define DHT11_PORT DHT11_GPIO_Port y #define DHT11_PIN DHT11_Pin

Definen los puertos necesarios para el funcionamiento del sensor *DHT11*.

3. Funciones

- void Set_Pin_Output (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Esta función pone los puertos como salidas.

- void Set_Pin_Input (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

Esta función pone los puertos como entradas.

- void delay (uint16_t time)

Esta función genera un *delay* en microsegundos. Fue creada para el sensor DHT11 ya que trabaja en una escala de microsegundos y con el HAL_Delay solo tenemos una escala de milisegundos.

- void DHT11_Start (void)

Esta función comienza la comunicación con el sensor *DHT11*. Para ello llama a las funciones *Set_Pin_Output (GPIOx, GPIO_Pin)* y *Set_Pin_Input (GPIOx, GPIO_Pin)* que inicialmente ponen el pin como salida y la escribe en 1, para después ponerla a 0 durante 18 ms y finalmente poner el pin como entrada para recibir la información.

```
void DHT11_Start (void)
{
	HAL_GPIO_WritePin(DHT11_PORT, DHT11_PIN, 1);	//Inicia el pin como HIGH
	HAL_Delay(300);
	Set_Pin_Output (DHT11_PORT, DHT11_PIN);		//pin como out para
transmitir
	HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 0);	//pin en LOW para iniciar
la comunicacion
	delay (18000);							//18ms
	Set_Pin_Input(DHT11_PORT, DHT11_PIN);		//pin como in para recibir
}
```

- uint8_t DHT11_Check_Response (void)

Esta función comprueba que el sensor *DHT11* funciona correctamente antes de leer la información que este recibe del exterior a través de la variable *Response* (0/1).

```
uint8_t DHT11_Check_Response (void)
{
	uint8_t Response = 0;
	delay (40);
	if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
	{
		delay (80);
		if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN))) Response = 1;	//El
pin responde
		else Response = -1;
	}
	while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)));
	//Espera a que el pin se ponga LOW

	return Response;
}
```

- uint8_t DHT11_Read (void)

Esta función lee los 8 *bits* de la información recibida del sensor y según el ancho del pulso la identifica como 1 o 0.

```
uint8_t DHT11_Read (void)
{
	uint8_t i,j;
	for (j=0;j<8;j++)
	{
		while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)));	//Espera a
que el pin se ponga HIGH
		delay (40);
		 //40 us
		if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))		//Si el
pin esta en LOW
		{
```

```
                    i&= ~(1<<(7-j));
                // write 0
               }
               else i|= (1<<(7-j));
                //Si el pin esta en HIHG, write 1
               while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)));   //Espera a
que el pin se ponga LOW
        }
        return i;
}
```

- void leerDatosSensorDHT11()

Esta función inicializa el sensor, detecta si hay respuesta a partir de la función *DHT11_Check_Response ()* y procede a leer los 40 *bits* de datos recibidos de la función *DHT11_Read ()*, dividiéndolos en *bytes* y almacenándolos en sus respectivas variables. Cabe de destacar que los *bytes* 2 tanto de la temperatura como de la humedad están vacíos debido a la precisión del sensor, aun así, hay que añadirlos para respetar la secuencia de comunicación del sensor. Finalmente, estos datos se convierten a tipo *float*.

```
void leerDatosSensorDHT11(){
        DHT11_Start();                          //se inicializa el sensor
        presencia = DHT11_Check_Response(); //esperamos a su respuesta
        Rh_byte1 = DHT11_Read();                //
        Rh_byte2 = DHT11_Read();                // Se leen los 5 siguientes
bytes de datos correspondientes a la humedad y temperatura
        Temp_byte1 = DHT11_Read();          //
        Temp_byte2 = DHT11_Read();          //
        SUM = DHT11_Read();

        TEMP = Temp_byte1;                  //valor de temperatura
        RH = Rh_byte1;                              //valor de humedad

        temperatura = (int) TEMP - 3;
        humedad = (int) RH;
}
```

- void escribirDatosDHT11aLCD()

Esta función imprime en la pantalla LCD los datos de tipo float obtenidos en la función *leerDatosSensorDHT11()* a partir de distintas funciones de la biblioteca *STM_MY_LCD16X2.h* .

```
void  escribirDatosDHT11aLCD(){
       LCD1602_clear();
       LCD1602_print("T: ");            //valores mostrados en la LCD
       LCD1602_PrintInt(temperatura);
       LCD1602_print("*C");
       LCD1602_print("  RH: ");
       LCD1602_PrintInt(humedad);
       LCD1602_print("%");
}
```

- void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)

Esta función gestiona la interrupción del riego manual mediante un pulsador, así como su funcionamiento o no según del nivel del agua del tanque, si se cumplen las condiciones de riego la

bomba se activará durante un período controlado por el *TIM2*. Además, muestra un mensaje en la pantalla *LCD* indicando los motivos del no riego.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	if((GPIO_Pin == Boton_Pin) && (bomba==0)){
		if(valorAgua>2520){
			if(humedad<60){
				HAL_TIM_Base_Start_IT(&htim2);
				HAL_GPIO_WritePin(GPIOD,BombaAgua_Pin,1);
				HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,1);
				bomba = 1;
			}
			else{
				LCD1602_2ndLine();
				LCD1602_print("Entorno humedo");
			}
		}
		else{
			LCD1602_2ndLine();
			LCD1602_print("  No hay agua  ");
		}

	}
	else{
		LCD1602_2ndLine();
		LCD1602_print("   Bomba ON    ");

	}
}
```

- void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)

Esta función gestiona el funcionamiento de los conversores analógicos digitales. Se ha añadido un temporizador de bloqueo debido a la problemática del número de datos generados, los cuales saturan el procesador por lo que no ejecuta otras funciones (se bloquea). Esto se ha solucionado de esta forma debido a las necesidades de frecuencia de 50MHz del sensor *DHT11*, la cual imposibilita el cambio de tiempo de muestreo por el método tradicional de reducir la frecuencia de reloj.

```
void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc){ //Conversores

	if (hadc->Instance == ADC1){
		valorAgua = HAL_ADC_GetValue(&hadc1);
		HAL_TIM_Base_Start_IT(&htim4);
	}

	if (hadc->Instance == ADC2){
		valorLDR = HAL_ADC_GetValue(&hadc2);
		HAL_TIM_Base_Start_IT(&htim4);
	}

}
```

- void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)

Esta función gestiona las interrupciones que ejecutan el código cuando finaliza la cuenta de cualquier temporizador. El temporizador 2 gestiona el tiempo de funcionamiento de la bomba una vez

encendida, cuando este acaba la *flag* bomba se pone a cero lo que permite que se pueda volver a encender en caso de otra interrupción. Por último el temporizador 4 se encarga de dividir la frecuencia de muestreo de los sensores.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM1) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */
  if (htim->Instance == TIM2){  //temporizador del regado
        bomba = 0;
        HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,0);
        HAL_GPIO_WritePin(GPIOD,BombaAgua_Pin,0);
        HAL_TIM_Base_Stop_IT(&htim2);
  }

  if (htim->Instance == TIM4){  //temporizador sensores
        HAL_TIM_Base_Stop_IT(&htim4);
        HAL_ADC_Start_IT(&hadc1);
        HAL_ADC_Start_IT(&hadc2);
    }
  /* USER CODE END Callback 1 */
}
```

4. Main

A continuación, se mostrarán distintos segmentos del código y se explicarán sus funcionalidades necesarias para el cumplimiento de los casos de uso mencionados anteriormente.

- Segmento 1

Este segmento se encarga de la inicialización del sistema.

```
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_ADC1_Init();
  MX_TIM6_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  MX_ADC2_Init();
  /* USER CODE BEGIN 2 */

  //Inicializacion LCD
```

```
  LCD1602_Begin8BIT(RS_GPIO_Port, RS_Pin, E_Pin, D0_GPIO_Port, D0_Pin, D1_Pin,
D2_Pin, D3_Pin, D4_GPIO_Port, D4_Pin, D5_Pin, D6_Pin, D7_Pin);
  LCD1602_clear();
  LCD1602_print("   Iniciando   ");
  LCD1602_2ndLine();
  LCD1602_print("    sistema    ");
  //LCD1602_PrintFloat(temperatura);

  HAL_TIM_Base_Start(&htim6); //Se inicia el temporizador del sensor de humedad

  HAL_Delay(2000);            //2s

  //Inicializamos los ADC
  HAL_ADC_Start_IT(&hadc1);
  HAL_ADC_Start_IT(&hadc2);
```

- Segmento 2

Este segmento se encarga de los indicadores del nivel del agua, en función de la variable *valorAgua*, a partir de los *LEDs* internos de la placa.

```
        if (valorAgua<2520){
              HAL_GPIO_WritePin(GPIOD,GPIO_PIN_14,1);
              HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,0);
        }
        else {
              HAL_GPIO_WritePin(GPIOD,GPIO_PIN_14,0);
              HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,1);
        }
```

- Segmento 3

Controla el riego automático del agua en función de la variable *valorLDR* para provocar que el riego de nuestra planta se produzca sólo por la noche y así evitar la evaporación de esta. Además, influyen otros valores como la humedad que evita que el sistema riegue en caso de lluvia o un regado manual anterior, también como es lógico el sistema no empezará el proceso de riego si ya está regando. En caso de que no haya agua no se producirá el riego, por ser un gasto de energía inútil y el sistema mostrará un mensaje en la pantalla *LCD* indicándolo.

```
if (valorLDR<320){
              HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, 1);
                    if(bomba==0) {
                          if(valorAgua>2520){
                                      if(humedad<60){
                                            HAL_TIM_Base_Start_IT(&htim2);

HAL_GPIO_WritePin(GPIOD,BombaAgua_Pin,1);

HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,1);
                                                  bomba = 1;
                                      }
                                      else{
                                           LCD1602_2ndLine();
                                           LCD1602_print("Entorno humedo");
                                      }
                          }
                          else{
                                LCD1602_2ndLine();
                                LCD1602_print("  No hay agua  ");
                          }
                    }
```

```
                    else{
                          LCD1602_2ndLine();
                          LCD1602_print("   Bomba ON   ");
                     }
        }
        else{
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, 0);
               }
```

- Segmento 4

Este segmento se encarga de cortar el riego tanto automático como manual en caso repentino de aumento de la humedad (lluvia).

```
        if(humedad>60){
              HAL_GPIO_WritePin(GPIOD,BombaAgua_Pin,0);
              bomba = 0;
        }
```

- Segmento 5

Este segmento se encarga de encender el motor del ventilador en caso de un aumento crítico de la temperatura con la finalidad de reducir esta. Por lo tanto, mantendrá su funcionamiento hasta que esta vuelva a valores razonables.

```
        //VENTILADOR
        if (temperatura > 25.0){ //Ventilacion en caso de alta
temperatura
              HAL_GPIO_WritePin(GPIOD,Ventilador_Pin,1);
        }
        else {
              HAL_GPIO_WritePin(GPIOD,Ventilador_Pin,0);
        }
```

*Otra futura funcionalidad que se podría añadir a este segmento sería encender el ventilador en función de un determinado *valorLDR* y un *valorCO2* generado por un sensor de CO_2 con la finalidad de que, por la noche, cuando las plantas no hacen la fotosíntesis, hubiera un sistema de ventilación para evitar la acumulación de CO_2. Finalmente, no fue implementado por la falta de este sensor.

- Segmento 6

Este segmento gestiona los tiempos de espera en el sensor *DHT11* ya que el fabricante recomienda unos tiempos de muestreo superiores a 1 s. Una vez pasado un segundo se ejecutan las funciones relacionadas con el sensor.

```
        //SENSOR HUMEDAD
        uint32_t tiempo_espera = 1000;
        uint32_t tickstart = HAL_GetTick(); //se crea un delay de 2s sin
bloquear el micro
        while((HAL_GetTick() - tickstart) < tiempo_espera){

        }
```

```
        leerDatosSensorDHT11();

        escribirDatosDHT11aLCD();

}
```

BIBLIOGRAFIA

- Material de clase
- Material del laboratorio de la asignatura
- Youtube.com
- Deepbluembedded.com
- Controllerstech.com
- Electronics.stackexchange.com
- Datasheet STM32F407
- Datasheet L293D
- Datasheet DHT11