PROYECTO FINAL SENSOR DE TEMPERATURA Y HUMEDAD

Emilio Amir Oros Salazar, Francesc Julio Aquize Flores

emilio.oros@ucb.edu.bo , francesc.aquize@ucb.edu.bo

Resumen

Este informe relata las configuraciones iniciales para la elaboración de un pequeño proyecto que consiste en la medición de temperatura y humedad a través de un sensor dth11, estos datos pasan a través de una tarjeta programable stm32f103c8t6 y son mostrados mediante una pantalla oled I2C.

1. Introducción

Los sensores de humedad son aparatos analógicos o digitales extremadamente útiles. Por ejemplo, en los hogares y en los locales comerciales permiten medir el porcentaje de vapor de agua presente en el aire. En base a ello, envían señales al sistema de climatización para que lo regule y lo sitúe en el rango de confort. También alertan al usuario para que utilice humidificadores y deshumidificadores con los que ajustarlo.

Pero no solo eso. Un sensor de humedad tiene multitud de aplicaciones en el mundo de la jardinería, la agricultura y la construcción. Solo hace falta elegir el adecuado [1].

El agua es una de las sustancias químicas más importantes de la Naturaleza que influye directamente en la existencia de vida en nuestro planeta. El hecho de que la Tierra se encuentre situada a una determinada distancia del Sol permite que el agua esté presente en la atmósfera en sus tres estados: gaseoso, líquido y sólido. De todos ellos el que predomina claramente es el gaseoso. El aire empleado en los procesos de ventilación y acondicionamiento, se le define como una mezcla de aire seco y vapor de agua. A través de la sicrometría la ciencia estudia las propiedades térmicas del aire húmedo, su regulación, su medición y el efecto que la humedad produce en los materiales y en las personas. La humedad contenida en el aire modifica en gran medida sus propiedades e influye enormemente en las sensaciones físicas de las personas. Cuando se habla de humedad en realidad nos estamos refiriendo a aire saturado, es decir aire en el que se mantiene en equilibrio el vapor de agua que contiene, sin que pase a ningún otro estado.

La humedad es por tanto vapor de agua contenido dentro de aire seco. A la presión a la que ese vapor de agua se mantiene estable se le llama presión de saturación. A cada temperatura le corresponde una diferente. La humedad absoluta es el peso de vapor contenido por unidad de volumen de aire (Kg/m3). La humedad relativa (%), es el cociente entre el peso del vapor de agua contenido en un volumen de aire y el peso del vapor saturado en ese mismo volumen [2].

2. Marco teórico

El sensor de humedad

El sensor de humedad es un aparato de lectura utilizado en espacios interiores para controlar la humedad del aire y la temperatura. Las magnitudes medidas por el sensor de humedad se transforman en una señal eléctrica normalizada, cuya intensidad suele estar comprendida entre 4 y 20 mA. Un material semiconductor es el encargado de determinar con precisión los valores de humedad y temperatura que se corresponden con la señal emitida. Este tipo de sensores son especialmente útiles en los sistemas de ventilación mecánica hidrorregulables, ya que permiten regular el caudal de aire renovado en función de la humedad ambiental. Estos sistemas pueden emplearse tanto en viviendas individuales como colectivas. En este último caso se utilizan sistemas de ventilación individualizados que llevan a cabo un barrido y posterior renovación del aire contaminado de los locales. Al ser independiente se consigue disponer de aire interior de calidad y autonomía en el consumo de cada vivienda.

Cuando el sistema hidrorregulable es centralizado se puede utilizar indistintamente en viviendas unifamiliares y plurifamiliares. El sensor de humedad manda una señal a las bocas de extracción hidrorregulables situadas en baños, aseos y cocina por donde el aire viciado se extrae cuando es necesario. El aire limpio entra en el salón y los dormitorios por las entradas de aire hidrorregulables ubicadas en la carpintería de las ventanas, una vez que el sensor ha detectado que los valores de humedad en el interior no son los adecuados [3].

¿Para qué sirve el sensor de humedad?

Como dijimos antes, los sensores de humedad sirven para determinar el porcentaje de vapor presente en el aire seco, así como de agua en determinados materiales. Cuando éste sobrepasa ciertos niveles “de confort”, eleva la sensación de calor o frío. Asimismo, si no los alcanza, provoca sequedad en la garganta y en las vías respiratorias en general. Los sensores de humedad sirven para determinar el porcentaje de vapor presente Ese es el motivo por el que los sistemas de ventilación mecánica hidrorregulables incluidos en los aparatos de climatización los incluyen. Estos sensores envían una señal que indica que el porcentaje de humedad es demasiado alto o excesivamente bajo. A continuación, se encargan de extraer una determinada cantidad de aire y de sustituirla por otra equivalente procedente del exterior para compensarla. Sin embargo, los sensores de humedad tienen otras utilidades más allá de las relacionadas con los sistemas de aire acondicionado y calefacción. Vamos a verlas:

Permiten comprobar el porcentaje de humedad presente en el suelo de jardines y huertos. Así es posible ajustar

eficazmente el riego para favorecer el crecimiento de las plantas.

Brindan la posibilidad de medir la humedad de los materiales de construcción. Esto permite detectar el momento en el que han terminado de fraguar y se puede seguir trabajando con ellos.

Ayudan a prevenir inundaciones y fugas de agua. Solo hay que colocarlos estratégicamente en almacenes, garajes, cocinas o cuartos de baño, por ejemplo [4].

Sensor de temperatura

Los sensores de temperatura son componentes eléctricos y electrónicos que, en calidad de sensores, permiten medir la temperatura mediante una señal eléctrica determinada. Dicha señal puede enviarse directamente o mediante el cambio de la resistencia. También se denominan sensores de calor o termosensores. Un sensor de temperatura se usa, entre otras aplicaciones, para el control de circuitos. Los sensores de temperatura también se llaman sensores de calor, detectores de calor o sondas térmicas [5].

2.1. Características técnicas

Componente	Información relevante	Características técnicas
Tarjeta de desarrollo STM32F103C8T6	Tarjeta STM32 es una placa desarrollo llamada Blue Pill, es de bajo costo que incorpora el núcleo RISC de 32 bits ARM®Cortex-M3 de alto rendimiento, esta tarjeta es adecuada para iniciar proyectos sobre el microcontrolador STM32. Viene con terminales header´s macho – macho que podrás soldar tu mismo. La tarjeta STM32F103C8T6 Blue Pill? STM32F103C8T6 es un microcontrolador muy potente y con CPU de 32 bits, puede superar fácilmente a Arduino UNO en rendimiento. Para programar esta placa tienes que utilizar un programador ST-LINK V2 ya que esta placa no lo incorpora un programador, necesita de un externo, hay distintas formas de programarlo ya sea con IDE de STM32 o incluso con Arduino IDE.	Tipo: Tarjeta STM32F103C Blue Pill • Modelo: STM32F103C • Dimensiones: 23mm x 53mm • Color PCB: Azul • Alimentación de voltaje: • Cualquier pin + 3.3V: (+ 3.3V) • Cualquier pin + 5V: (+ 5V) • Conector MicroUSB: (+ 5V) • Voltaje Lógico IO: 3.3 V • Procesador de núcleo ARM® Cortex®-M3 • Software de programación: STM32CubeProgrammer, STM32CubeIDE, SW4STM32, ARDUINO IDE y Mbed • Firmware precargado: Bootloader para programar con STLINK • Tamaño de núcleo 32-bits • Entradas/salidas Digitales: 37 • Canales PWM:12 • Entradas ADC: 10 canales A/D de 12Bits • Conectividad CAN, I²C, IrDA, LIN, SPI, UART/USART, USB • Periféricos DMA, control de motor por PWM, PDR, POR, PVD, PWM, sensor de temp., WDT • Frecuencia de trabajo de 72MHz. • Memoria: • Flash: 64 KB • SRAM: 20KB • Temperatura de operación -40°C ~ 85°C
Sensor de temperatura DHT11	El DHT11 es un sensor digital de temperatura y humedad relativa de bajo costo y fácil uso. Integra un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos (no posee salida analógica). Utilizado en aplicaciones académicas relacionadas al control automático de temperatura, aire acondicionado, monitoreo ambiental en agricultura y más. Utilizar el sensor DHT11 con las plataformas Arduino/Raspberry Pi/Nodemcu es muy sencillo tanto a nivel de software como hardware. A nivel de software se dispone de librerías para Arduino con soporte para el protocolo "Single bus". En cuanto al hardware, solo es necesario conectar el pin VCC de alimentación a 3-5V, el pin GND a Tierra (0V) y el pin de datos a un pin digital en nuestro Arduino. Si se desea conectar varios sensores DHT11 a un mismo Arduino, cada sensor debe tener su propio pin de datos. Quizá la única desventaja del sensor es que sólo se puede obtener nuevos datos cada 2 segundos. Cada sensor es calibrado en fabrica para obtener unos coeficientes de calibración grabados en su memoria OTP, asegurando alta	• Voltaje de Operación: 3V - 5V DC • Rango de medición de temperatura: 0 a 50 °C • Precisión de medición de temperatura: ±2.0 °C • Resolución Temperatura: 0.1°C • Rango de medición de humedad: 20% a 90% RH. • Precisión de medición de humedad: 5% RH. • Resolución Humedad: 1% RH • Tiempo de sensado: 1 seg. • Interface digital: Single-bus (bidireccional) • Modelo: DHT11 • Dimensiones: 16*12*5 mm • Peso: 1 gr. • Carcasa de plástico celeste

	estabilidad y fiabilidad a lo largo del tiempo. El protocolo de comunicación entre el sensor y el microcontrolador emplea un único hilo o cable, la distancia máxima recomendable de longitud de cable es de 20m., de preferencia utilizar cable apantallado. Proteger el sensor de la luz directa del sol (radiación UV). En comparación con el DHT22 y DHT21, este sensor es menos preciso, menos exacto y funciona en un rango más pequeño de temperatura / humedad, pero su empaque es más pequeño y de menor costo.	
Pantalla oled ssd1306	Las pantallas OLED se destacan por su gran contraste, mínimo consumo de energía y buena calidad de imagen. El display oled 0.96" SPI SSD1306 posee una resolución de 128*64 píxeles, permitiendo controlar cada píxel individualmente y mostrar tanto texto como gráficos. Además por ser de tipo OLED no necesita de retroiluminación (Backlight) como los LCD, lo que hace que su consumo de energía sea mucho menor y aumenta su contraste. El display posee interfaz de comunicación de tipo SPI, pero puede ser cambiada a I2C soldando unos puntos en la placa. Diseñado para trabajar a 5V directamente, gracias a su regulador de voltaje en placa y puede trabajar con sistemas de 3V o 5V sin necesidad de conversores. Debemos tener en cuenta que los pines SPI e I2C son diferentes para cada modelo de Arduino, por lo que debemos revisar el Pinout de nuestro Arduino para saber cuales son los pines SPI, por ejemplo en Arduino Uno son los pines 9, 10, 11, 12, 13. Para manejar la pantalla es necesario utilizar un microcontrolador con al menos 1K de RAM, este espacio cumple la función de buffer para el display. El driver de la pantalla es el SSD1306, con una librería lista para usarse en Arduino. La librería permite mostrar texto, mapas de bits, píxeles, rectángulos, círculos y líneas. A pesar de usar 1K de RAM, el funcionamiento es muy rápido y el código es fácilmente portable a distintas plataformas de microcontroladores.	• Voltaje de operación: 3V – 5.5V DC • Driver: SSD1306 • Interfaz: SPI • Resolución: 128*64 píxeles • Monocromo: píxeles blancos (fondo negro) • Ángulo de visión: 160º • Área visible (display): 23*11.5 mm • Consumo de energía ultra bajo: 0.08W (cuando están encendidos todos los píxeles) • Temperatura de trabajo: -30ºC ~ 70ºC • Dimensiones: 27*27*4.1mm • Peso: 5 gramos

3. Parte práctica

Para comenzar el proyecto debemos abrir el software de desarrollo STM32 Cube IDE

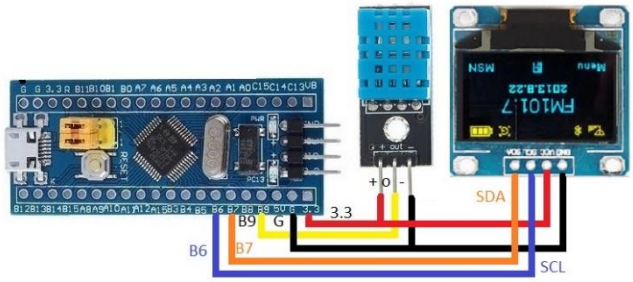


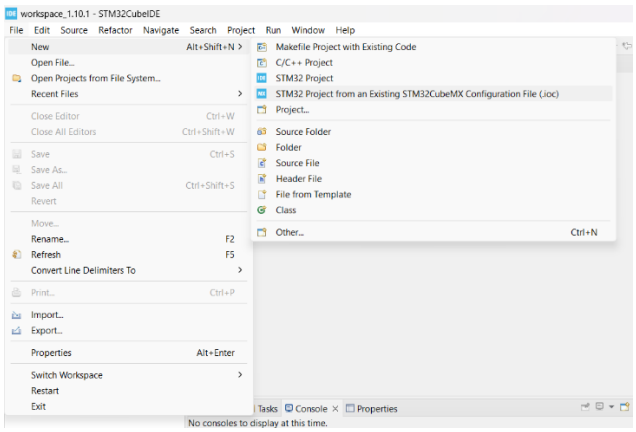
Figura 1. Diagrama de conexiones.

3.1. Configuraciones iniciales

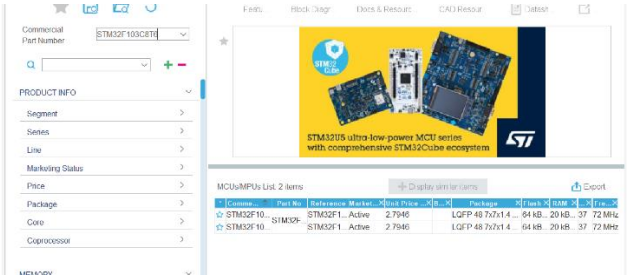
A continuación, anexamos las pruebas de funcionamiento de nuestro laboratorio.

Se procede a detallar los pasos para la programación del proyecto.

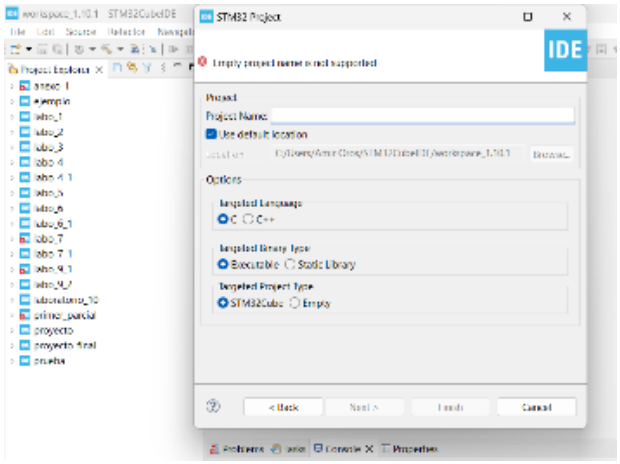
1. Creamos un nuevo proyecto STM32



2. Seleccionamos la tarjeta STM32F103C8T6

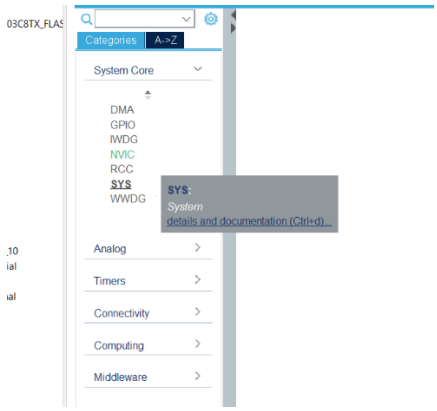


3. Asignamos el nombre del proyecto

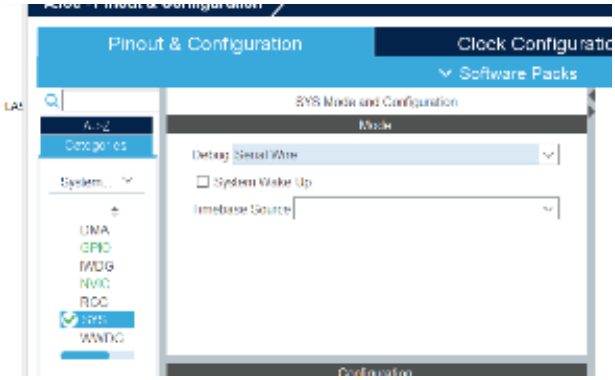


4. Pantalla del IDE

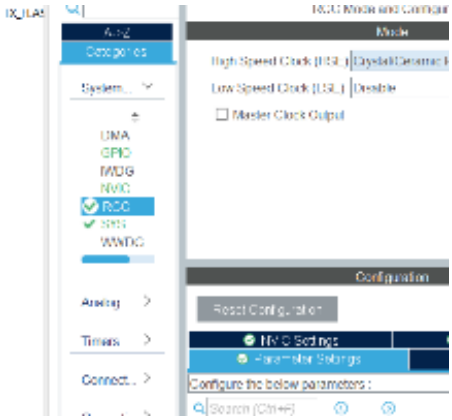
5. Nos dirigimos a la pestaña SYS



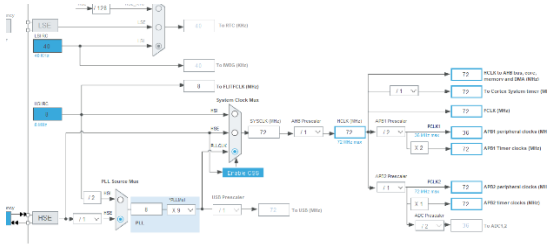
6. Definimos como serial wire



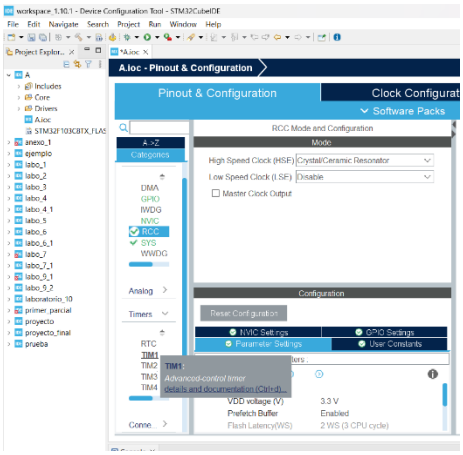
7. Nos dirigimos a la pestaña RCC y se configura como High Speed Clock Crystal Ceramic Resonator



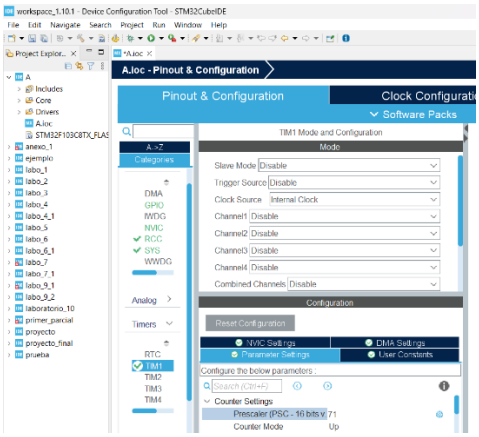
8. Nos dirigimos a la pestaña clock configuration y seteamos la frecuencia en 72 MHZ



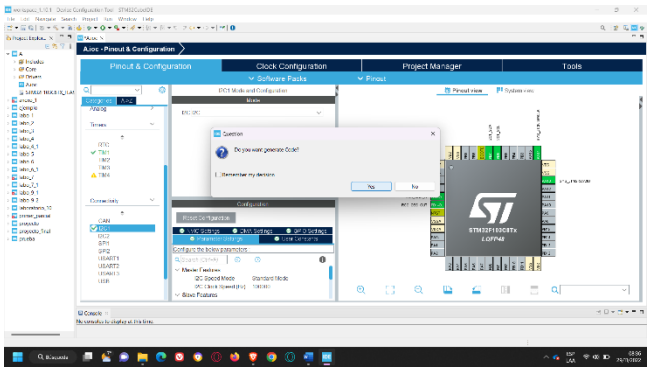
9. Volvemos a la pestaña Pin-out y nos dirigmos a la pestaña TIM1, donde habilitamos la opción use internal clock y configuramos la frecuencia del pre-escalador en 71.



10. Nos dirigimos a la pestaña conectivity y seleccionamos I2C y en la misma habilitamos la opción I2C.



11. Guardamos el proyecto y generamos el código base.



3.2. Código del proyecto

Se presenta el código utilizado en el proyecto donde las partes resaltadas corresponden a la codificación realizada sobre el código base generado.

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "fonts.h"
#include "ssd1306.h"
#include "stdio.h"
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim1;

/* USER CODE BEGIN PV */
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
#define DHT11_PORT GPIOB
#define DHT11_PIN GPIO_PIN_9
uint8_t RHI, RHD, TCI, TCD, SUM;
uint32_t pMillis, cMillis;
float tCelsius = 0;
float tFahrenheit = 0;
float RH = 0;
uint8_t TFI = 0;
uint8_t TFD = 0;
char strCopy[15];
void microDelay (uint16_t delay)
{
 __HAL_TIM_SET_COUNTER(&htim1, 0);
 while (__HAL_TIM_GET_COUNTER(&htim1) < delay);
}
uint8_t DHT11_Start (void)
{
 uint8_t Response = 0;
 GPIO_InitTypeDef GPIO_InitStructPrivate = {0};
 GPIO_InitStructPrivate.Pin = DHT11_PIN;
 GPIO_InitStructPrivate.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
 GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as output
 HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 0);   // pull the pin low
 HAL_Delay(20);   // wait for 20ms
 HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 1);   // pull the pin high
 microDelay (30);   // wait for 30us
 GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
 GPIO_InitStructPrivate.Pull = GPIO_PULLUP;
 HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as input
 microDelay (40);
 if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
 {
```

```
  microDelay (80);
  if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN))) Response = 1;
 }
 pMillis = HAL_GetTick();
 cMillis = HAL_GetTick();
 while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
 {
  cMillis = HAL_GetTick();
 }
 return Response;
}

uint8_t DHT11_Read (void)
{
 uint8_t a,b;
 for (a=0;a<8;a++)
 {
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {  // wait for the pin to go high
   cMillis = HAL_GetTick();
  }
  microDelay (40);   // wait for 40 us
  if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))   // if the pin is low
   b&= ~(1<<(7-a));
  else
   b|= (1<<(7-a));
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {  // wait for the pin to go low
   cMillis = HAL_GetTick();
  }
 }
 return b;
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
 /* USER CODE BEGIN 1 */

 /* USER CODE END 1 */
```

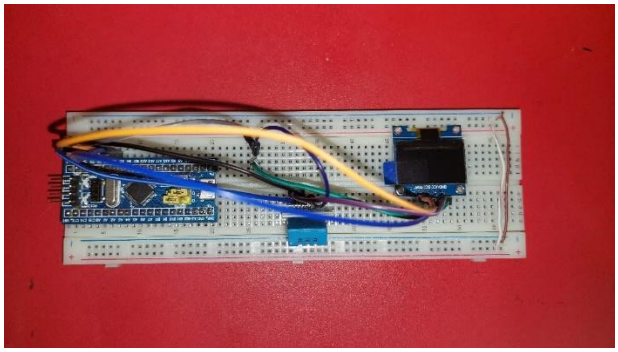
```
 /* MCU Configuration--------------------------------------------------------*/

 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 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_I2C1_Init();
 MX_TIM1_Init();
 /* USER CODE BEGIN 2 */
 HAL_TIM_Base_Start(&htim1);
 SSD1306_Init();
 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
            if(DHT11_Start())
             {
              RHI = DHT11_Read(); // Relative humidity integral
              RHD = DHT11_Read(); // Relative humidity decimal
              TCI = DHT11_Read(); // Celsius integral
              TCD = DHT11_Read(); // Celsius decimal
              SUM = DHT11_Read(); // Check sum
              if (RHI + RHD + TCI + TCD == SUM)
              {
               // Can use RHI and TCI for any purposes if whole number only needed
               tCelsius = (float)TCI + (float)(TCD/10.0);
               tFahrenheit = tCelsius * 9/5 + 32;
               RH = (float)RHI + (float)(RHD/10.0);
               // Can use tCelsius, tFahrenheit and RH for any purposes
               TFI = tFahrenheit;  // Fahrenheit integral
               TFD = tFahrenheit*10-TFI*10; // Fahrenheit decimal
               sprintf(strCopy,"%d.%d C  ", TCI, TCD);
               SSD1306_GotoXY (0, 0);
               SSD1306_Puts (strCopy, &Font_11x18, 1);
```

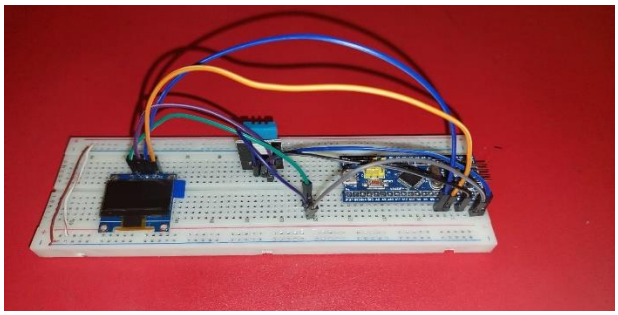
```
                    sprintf(strCopy,"%d.%d F   ", TFI, TFD);
                    SSD1306_GotoXY (0, 20);
                    SSD1306_Puts (strCopy, &Font_11x18, 1);
                    sprintf(strCopy,"%d.%d %%  ", RHI, RHD);
                    SSD1306_GotoXY (0, 40);
                    SSD1306_Puts (strCopy, &Font_11x18, 1);
                    SSD1306_UpdateScreen();
                  }
                }
                HAL_Delay(2000);



    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 400000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */
```

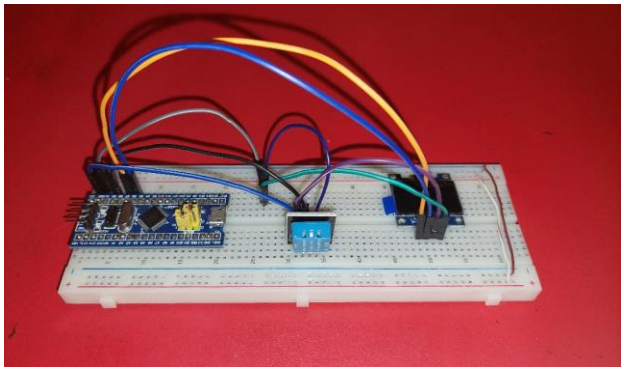
```
  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 71;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_RESET);

  /*Configure GPIO pin : PB9 */
  GPIO_InitStruct.Pin = GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
```

```
}
#endif /* USE_FULL_ASSERT */
```

4. **Pruebas de funcionamiento**







5. **Conclusiones**

A pesar de que este pequeño proyecto es meramente didáctico y demostrativo, no se puede descartar su utilidad en la vida real.

Al extrapolar las técnicas de desarrollo y codificación de este proyecto podemos darle un enfoque totalmente funcional y profesional debido a que El monitoreo de humedad y temperatura son fundamentales para garantizar un entorno óptimo de trabajo. Las personas, las mercancías y las máquinas son sensibles a las variaciones de sus valores.

El monitoreo de humedad es importante si se desea crear ambientes confortables. Además, el aire y su humedad, puede afectar de manera muy distinta a la salud, inmuebles o estructuras.

Los valores de humedad relativa bajos resecan tus objetos, el entorno y a ti. En cambio, valores altos, cercanos al 100%, transferirá humedad a todo lo que toque. Creando, en este último caso, deterioro de madera, oxidación de metales, aparición de moho, etc.

Los valores ideales de monitoreo de humedad son entre 40% y 60%. Por debajo de 40% debes considerarlo bajo y por encima de 60% alto. En ambos casos, según corresponda, deberás tomar medidas para ajustar la humedad del ambiente.

Para corregir la humedad baja puedes utilizar, por ejemplo, humidificadores que incrementan el valor. En caso contrario, que lo que desees sea reducir la humedad, podrías utilizar deshumidificadores.

Ahora, en el caso de la medición de la temperatura ambiental. Resulta beneficioso conocer su valor porque las personas y mercancías son sensibles a sus cambios.

Las personas expuestas a una elevada temperatura pierden agua y electrolitos, necesarios para el funcionamiento de sus órganos. Las bajas temperaturas disminuyen las defensas y favorece la aparición de enfermedades respiratorias. Además, se pueden agravar las enfermedades crónicas cardiovasculares y respiratorias.

Ahora, en cuanto a las mercancías, muchas son sensibles a la temperatura. Por ejemplo, los alimentos se conservan mejor a determinados niveles de temperatura, de igual manera, hay equipos, herramientas, etc, que por encima o debajo de determinada temperatura fallan.

Se puede concluir preliminarmente que el monitoreo de humedad y temperatura permite mantener condiciones ideales de operación de tus máquinas y equipos.

LINK DEL PROYECTO COMPLETO EN EL REPOSITORIO GITHUB	https://github.com/AmirOros/proyecto_final

VIDEO EXPLICATIVO DEL PROYECTO (PRIMERA VERSIÓN)	https://www.youtube.com/watch?v=7oop5I560mo

5.Referencias

- [1] Ó. Rodríguez. "Qué es el sensor de humedad, qué utilidad tiene y aplicaciones". elconfidencial.com. https://www.elconfidencial.com/tecnologia/2021-08-17/que-es-sensor-de-humedad-utilidad-aplicaciones_3220448/ (accedido el 29 de noviembre de 2022).
- [2] "El funcionamiento de un sensor de humedad. Usos frecuentes". El blog de la ventilación inteligente. https://www.siberzone.es/blog-sistemas-ventilacion/el-funcionamiento-de-un-sensor-de-humedad-usos-frecuentes/ (accedido el 29 de noviembre de 2022).

- [3] "El funcionamiento de un sensor de humedad. Usos frecuentes". El blog de la ventilación inteligente. https://www.siberzone.es/blog-sistemas-ventilacion/el-funcionamiento-de-un-sensor-de-humedad-usos-frecuentes/ (accedido el 29 de noviembre de 2022).
- [4] "¿Que es el sensor de temperatura?" https://www.elconfidencial.com/tecnologia/2021-08-17/que-es-sensor-de-humedad-utilidad-aplicaciones_3220448/(accedido el 29 de noviembre de 2022).
- [5] "El sensor de temperatura - Rechner Sensors". Rechner Sensors. https://www.rechner-sensors.com/es/documentacion/knowledge/el-sensor-de-temperatura#:~:text=Los%20sensores%20de%20temperatura%20son,sensores%20de%20calor%20o%20termosensores. (accedido el 29 de noviembre de 2022).
- [6] "STM32F103C8T6 producto de programación "https://uelectronics.com/producto/stm32f103c8t6-tarjeta-de-desarrollo-cortex-m3-blue-pill-stm32f103c/(accedido el 29 de noviembre de 2022).
- [7] "Sensor de temperatura y humedad relativa DHT11". Naylamp Mechatronics - Perú. https://naylampmechatronics.com/sensores-temperatura-y-humedad/57-sensor-de-temperatura-y-humedad-relativa-dht11.html#:~:text=El%20DHT11%20es%20un%20sensor%20digital%20de%20temperatura%20y%20humedad,(no%20posee%20salida%20analógica). (accedido el 29 de noviembre de 2022).
- [8] "Display Oled 0.96" SPI 128*64 SSD1306". Naylamp Mechatronics - Perú. https://naylampmechatronics.com/oled/83-display-oled-096-spi-12864-ssd1306.html (accedido el 29 de noviembre de 2022).