

SISTEMA DI MONITORAGGIO AMBIENTALE CON STM32F3DISCOVERY

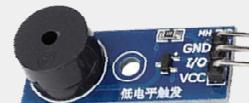
Federica Del Vecchio – M63001587
Università degli Studi di Napoli Federico II
Elaborato di «Architettura e Progetto di Calcolatori»



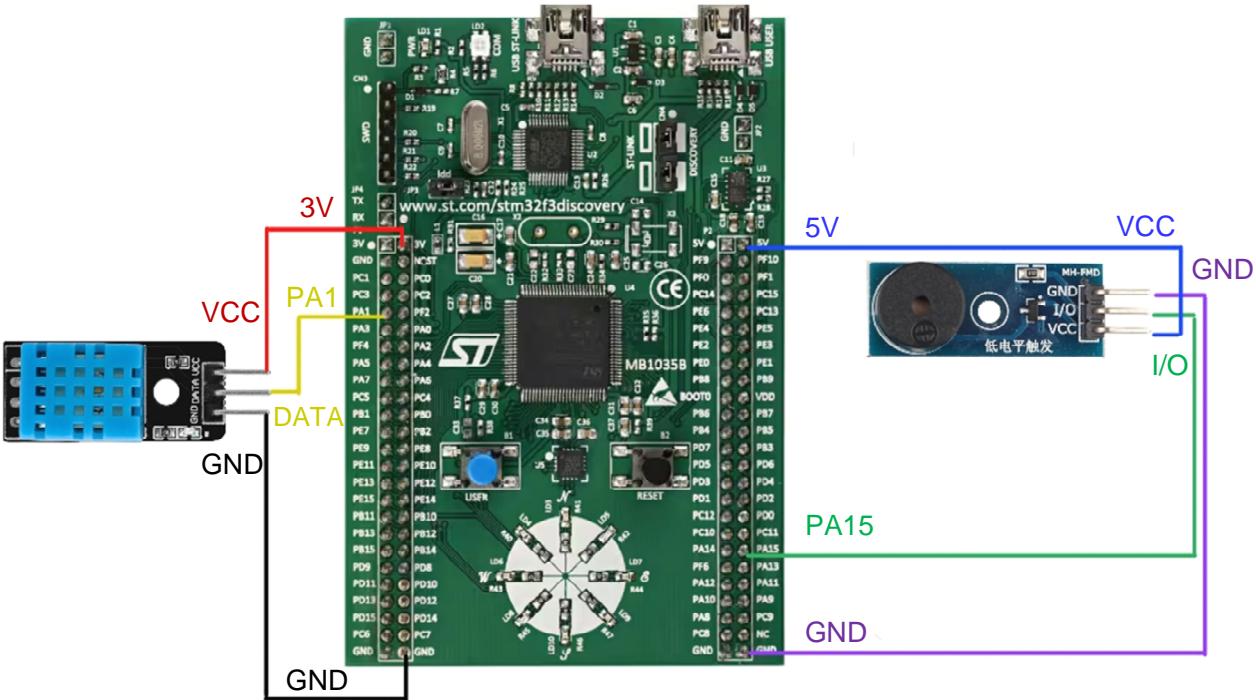
INTRODUZIONE

- L'obiettivo di questo progetto è sviluppare un **sistema di monitoraggio ambientale** utilizzando la scheda STM32F3DISCOVERY. La **scheda di sviluppo STM32F3DISCOVERY**, dotata del **microcontrollore STM32F303VCT6** della famiglia **STM32F3** di **STMicroelectronics**, costituirà il nucleo centrale del sistema.
- Il **sensore DHT11** sarà utilizzato per misurare con precisione **la temperatura e l'umidità** dell'ambiente circostante. DHT11 è un sensore **digitale** che comunica tramite **un'interfaccia seriale a singolo filo**; utilizza un protocollo proprietario per trasmettere i dati alla board.
- In caso di superamento di soglie prefissate di temperatura o umidità, il sistema emetterà un **allarme sonoro** tramite un **buzzer**, fornendo una notifica immediata in caso di condizioni non ottimali.
- Il valore di temperatura verrà immediatamente visualizzato tramite un numero variabile di LED accesi sulla scheda: maggiore è la temperatura, maggiore sarà il **numero di LED attivi**.
- I dati acquisiti saranno trasmessi al PC tramite un **cavo USB** per il monitoraggio continuo e il **logging** delle condizioni ambientali nel tempo.

COMPONENTISTICA

NOME	IMMAGINE
STM32F3DISCOVERY	 A green printed circuit board (PCB) featuring a central STM32F3 microcontroller, various component packages, and a breadboard-like area for prototyping.
DHT11 Humidity & Temperature Sensor	 A small blue plastic housing containing a DHT11 sensor chip, with three pins extending from the bottom.
Dupont jumper wires	 A bundle of several multi-colored Dupont jumper wires, used for connecting components on a breadboard.
Jopto DC 3.3-5V Passive Low Level Trigger Buzzer	 A blue PCB with a black cylindrical buzzer at the top, labeled with component values and connection pins (GND, I/O, VCC).

INTERCONNESSIONI



SENSORE DHT11 (1/7)

- Il sensore DHT11 offre una soluzione completa, combinando la misurazione di temperatura e umidità con un'**uscita digitale calibrata**, assicurando, così, un'integrazione semplice e affidabile.
- Il DHT11 combina un **componente resistivo** per la misurazione dell'umidità e un **sensore NTC** (Negative Temperature Coefficient) per la misurazione della temperatura. Questi sensori devono essere collegati a un **microcontrollore a 8 bit ad alte prestazioni**, il che garantisce una risposta rapida e misurazioni precise.
- Ogni DHT11 è calibrato in laboratorio per assicurare un'elevata precisione, in particolare, nella misurazione dell'umidità. I **coefficienti di calibrazione** sono memorizzati in un'opportuna **memoria OTP** (One-Time Programmable).
- Il DHT11 è dotato di un'**interfaccia seriale a singolo filo**, che ne semplifica l'integrazione con il STM32F3DISCOVERY. Le sue **dimensioni compatte**, il **basso consumo energetico** e la capacità di **trasmissione** del segnale **fino a 20 metri** lo rendono ideale per varie applicazioni.
- Il sensore DHT11 misura la temperatura **da 0 a 50°C** con una precisione di **±2°C** e l'umidità relativa **da 20 a 90%** con una precisione di **±5%**.

SENSORE DHT11 (2/7)

- Il sensore DHT11 può essere alimentato con una tensione compresa **tra 3 e 5.5V DC**. Dopo aver fornito l'alimentazione, è necessario attendere almeno un secondo prima di cominciare a inviare istruzioni al sensore per permettere al dispositivo di stabilizzarsi.
- La trasmissione dei dati tra il microcontrollore (MCU) e il sensore DHT11 impiega circa **4 millisecondi** per ogni ciclo di comunicazione.
- Un ciclo completo di trasmissione dati è composto da **40 bit**, con il sensore che invia **i bit più significativi per primi**. I dati inviati includono sia la parte intera che quella frazionaria.
- Il formato dei dati è il seguente:
 - **1° byte**: parte intera dell'umidità relativa in %
 - **2° byte**: parte frazionaria dell'umidità relativa in %
 - **3° byte**: parte intera della temperatura in gradi Celsius
 - **4° byte**: parte frazionaria della temperatura in gradi Celsius
 - **5° byte**: checksum, che è la somma di tutti i valori precedenti

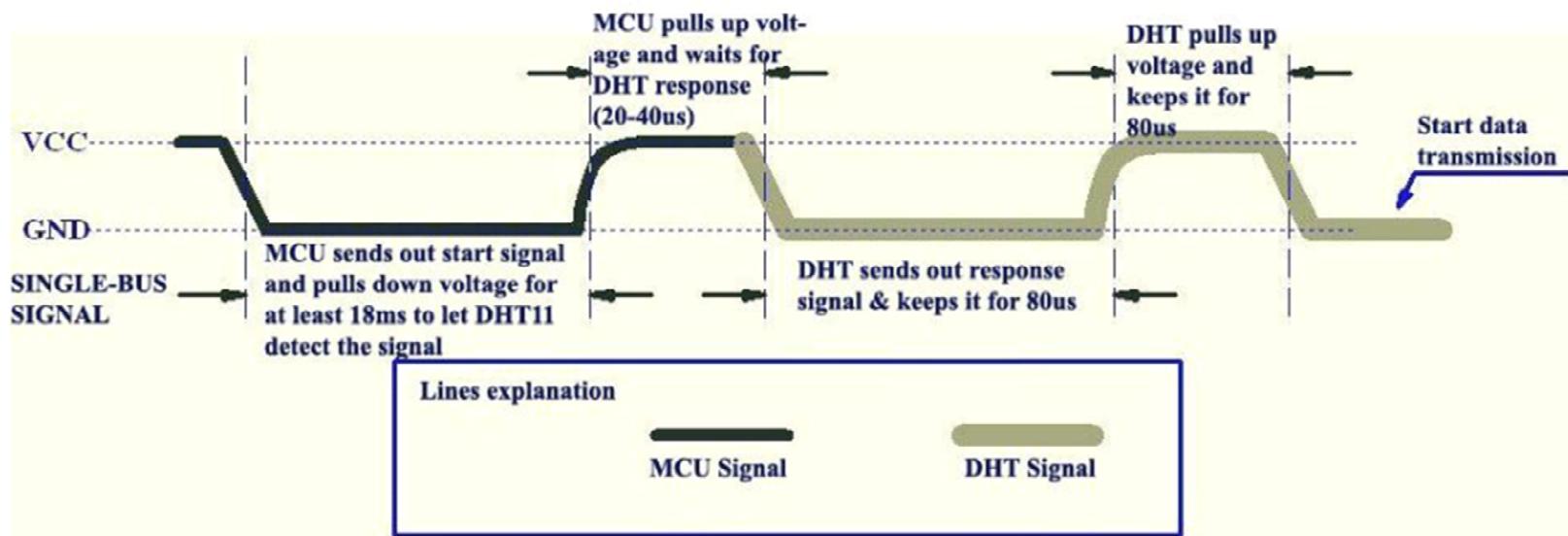
CODICE DEL PROGRAMMA (1/7)

```
/* Private includes -----  
/* USER CODE BEGIN Includes */  
#include "stdio.h"  
#include "usbd_cdc_if.h"  
/* USER CODE END Includes */  
  
/* Private define -----  
/* USER CODE BEGIN PD */  
#define NUM_LED 8  
#define DHT11_PORTA GPIOA  
#define DHT11_PIN GPIO_PIN_1      // PA1  
/* USER CODE END PD */  
  
/* USER CODE BEGIN PV */  
uint8_t umid_1, umid_2, temp_1, temp_2;  
uint16_t SUM, UMID, TEMP;  
  
float Temperatura = 0;  
float Umidita = 0;  
uint8_t Risultato = 0;  
  
GPIO_TypeDef *LED_PORT = GPIOE;  
uint16_t LED_PINS[NUM_LED] = {GPIO_PIN_8, GPIO_PIN_9, GPIO_PIN_10,  
                             GPIO_PIN_11, GPIO_PIN_12, GPIO_PIN_13, GPIO_PIN_14, GPIO_PIN_15};  
/* USER CODE END PV */
```

SENSORE DHT11 (3/7)

- Lo stato di riposo del bus dati è **a livello di tensione alto**. Per dare inizio alla comunicazione, il MCU **abbassa** il livello di tensione del bus dati per almeno **18 ms**.
- Successivamente, il MCU **rialza** il livello di tensione e attende **20-40 µs** per una risposta dal sensore DHT11.
- Una volta rilevato il segnale di avvio, DHT11 invia un segnale di risposta a livello **basso** che dura **80 µs**.
- Poi, il DHT11 **rialza** il livello di tensione per altri **80 µs** per prepararsi all'invio dei dati.
- Quando il bus dati torna a livello **basso**, il DHT11 sta iniziando a trasmettere i dati.

SENSORE DHT11 (4/7)



CODICE DEL PROGRAMMA (2/7)

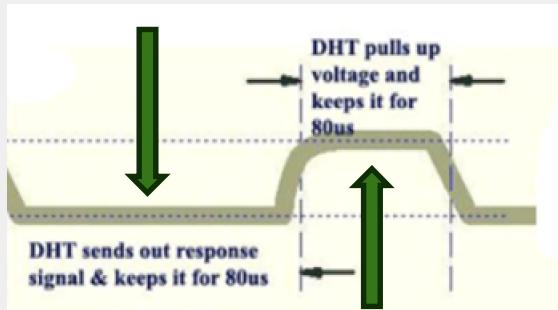
```
void set_pin_output (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; // mode: output
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
}

void set_pin_input (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT; // mode: input
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
}
```

```
/* USER CODE BEGIN 0 */
void ritarda (uint16_t time)
{
    __HAL_TIM_SET_COUNTER(&htim6, 0);
    while ((__HAL_TIM_GET_COUNTER(&htim6))<time);
}

void inizio_DHT11 ()
{
    set_pin_output (DHT11_PORTA, DHT11_PIN);           // setta il pin come output
    HAL_GPIO_WritePin (DHT11_PORTA, DHT11_PIN, 0);     // abbassa il pin
    ritarda (18000);                                // aspetta per 18 ms
    HAL_GPIO_WritePin (DHT11_PORTA, DHT11_PIN, 1);     // alza il pin
    ritarda (20);                                   // aspetta per 20 us
    set_pin_input(DHT11_PORTA, DHT11_PIN);           // setta il pin come input
}
```

CODICE DEL PROGRAMMA (3/7)

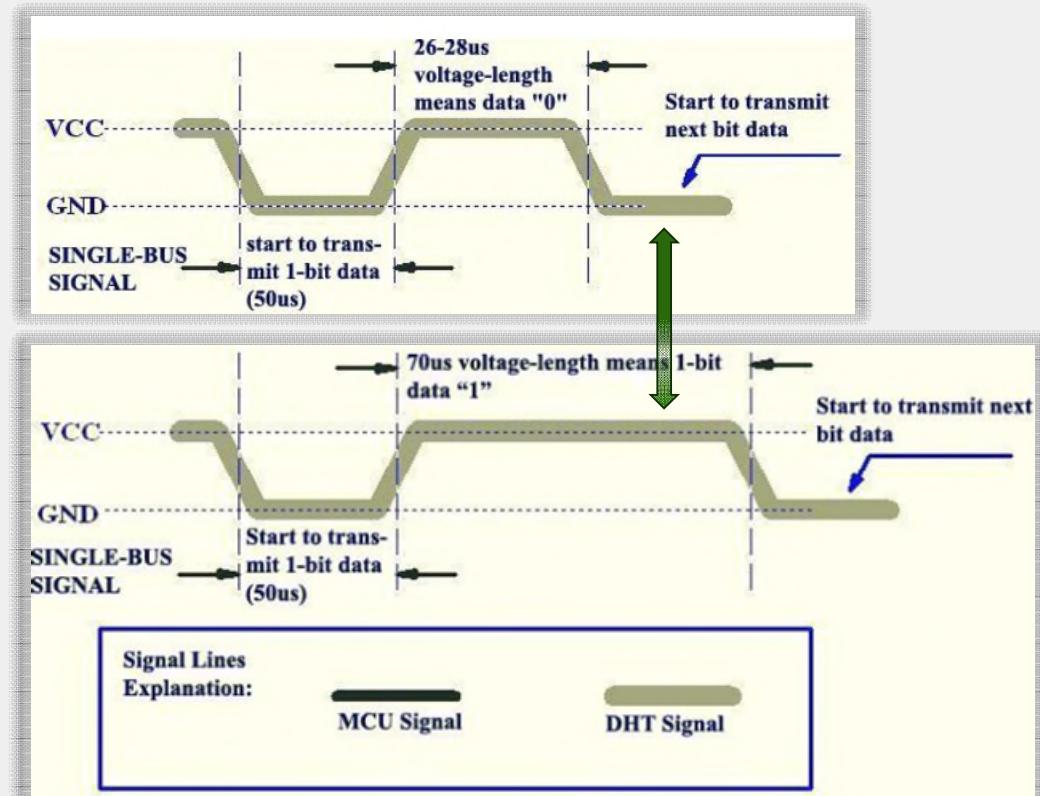


```
uint8_t controlla_risposta_DHT11 ()
{
    uint8_t risp = 0;
    ritarda (40);                                // aspetta 40 us (in modo tale da mettersi al "centro")
    if (!HAL_GPIO_ReadPin (DHT11_PORTA, DHT11_PIN)) // controlla che il pin sia basso
    {
        ritarda (80);                            // aspetta 80 us (sempre al "centro" del segnale)
        if ((HAL_GPIO_ReadPin (DHT11_PORTA, DHT11_PIN))) // controlla che il pin sia alto
            risp = 1;                          // se è così, il sensore è presente
        else risp = -1;                      // altrimenti, il sensore non è presente
    }
    while ((HAL_GPIO_ReadPin (DHT11_PORTA, DHT11_PIN))); // aspetta che il pin torni basso
    return risp;
}
```

SENSORE DHT11 (5/7)

- Dopo la risposta iniziale, il DHT11 inizia a trasmettere i dati al MCU. Ogni bit di dati inizia con **un livello basso di 50µs**.
- La durata del **livello alto successivo** determina se il bit di dati è "0" o "1":
 - Se il livello alto dura **circa 26-28µs**, il bit è "**0**".
 - Se il livello alto dura **circa 70µs**, il bit è "**1**".
- Quando l'ultimo bit è stato trasmesso, il DHT11 **abbassa** il livello di tensione per altri **50µs**.
- Successivamente, la tensione del bus dati viene **rialzata** da una resistenza per riportarla allo stato di riposo.

SENSORE DHT11 (6/7)



CODICE DEL PROGRAMMA (4/7)

```
uint8_t leggi_DHT11 (void)
{
    uint8_t i,j;
    for (j=0;j<8;j++)
    {
        while (!(HAL_GPIO_ReadPin (DHT11_PORTA, DHT11_PIN))); // aspetta che il pin sia alto
        ritarda (40); // aspetta per 40 us
        if (!(HAL_GPIO_ReadPin (DHT11_PORTA, DHT11_PIN))) // se il pin è basso...
        {
            i &= ~ (1<<(7-j)); // ...scrive "0" nella corretta posizione...
        }
        else i |= (1<<(7-j)); // ...altrimenti scrive "1"
        while ((HAL_GPIO_ReadPin (DHT11_PORTA, DHT11_PIN))); // aspetta che il pin sia basso
    }
    return i; // i è il byte ricevuto
}
```

SENSORE DHT11 (7/7)

I timer sono essenziali per gestire con precisione le operazioni di misurazione e comunicazione con il sensore DHT11 nel progetto.

- **Timer 6:** Gestisce la temporizzazione delle fasi di inizializzazione e lettura del sensore, grazie all'uso della funzione «ritardo». Il periodo del timer è configurato per essere di **1 nanosecondo**.
- **Timer 7:** Ogni **5 secondi**, attiva le misurazioni della temperatura e dell'umidità (e tutte le operazioni correlate) attraverso il meccanismo delle interruzioni.

PINOUT & CONFIGURATION (1/3)

TIMER 6

Parameter Settings

Configure the below parameters :

Search (Ctrl+F) 48-1

Counter Settings

- Prescaler (PSC - 16 bits value) 48-1
- Counter Mode Up
- Counter Period (AutoReload Re... 0xffff-1
- auto-reload preload Disable

Trigger Output (TRGO) Parameters

- Trigger Event Selection Reset (UG bit from TIMx_EGR)

TIMER 7

Parameter Settings

Configure the below parameters :

Search (Ctrl+F) 48000 - 1

Counter Settings

- Prescaler (PSC - 16 bits value) 48000 - 1
- Counter Mode Up
- Counter Period (AutoReload Re... 5000 - 1
- auto-reload preload Disable

Trigger Output (TRGO) Parameters

- Trigger Event Selection Reset (UG bit from TIMx_EGR)

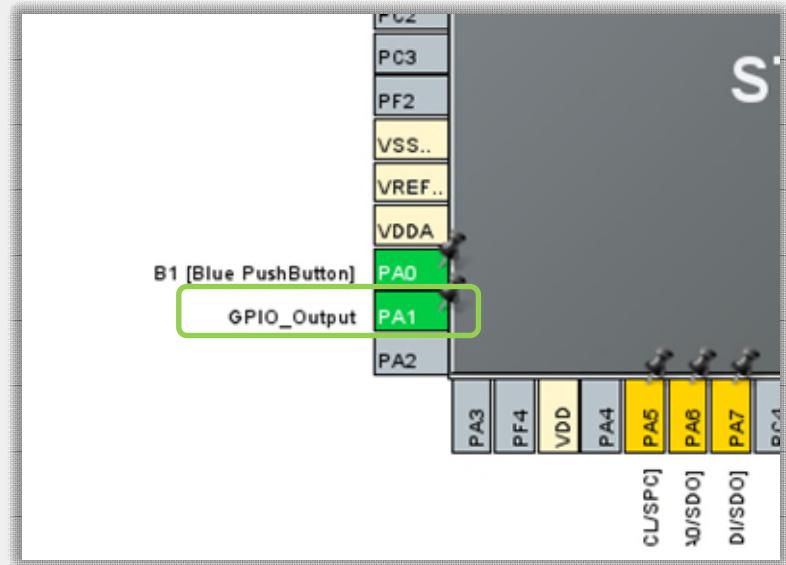
Parameter Settings	User Constants	NVIC Settings	DMA Settings
NVIC Interrupt Table TIM7 global interrupt	Enabled <input checked="" type="checkbox"/>	Preemption Priority 0	Sub Priority 0

PINOUT & CONFIGURATION (2/3)

CLOCK DI SISTEMA

RCC Mode and Configuration
Mode
High Speed Clock (HSE) Crystal/Ceramic Resonator

«DATA» DEL SENSORE DHT11



CODICE DEL PROGRAMMA (5/7)

```
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start(&htim6);
HAL_TIM_Base_Start_IT(&htim7);
HAL_Delay(2000);      // necessario
```

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
// funzione chiamata automaticamente quando...
// ...scade il periodo di un timer configurato
{
    if (htim == &htim7)    // TIM7
    {
        inizio_DHT11();
        Risultato = controlla_risposta_DHT11();

        umid_1 = leggi_DHT11();           // 1o byte
        umid_2 = leggi_DHT11();           // 2o byte
        temp_1 = leggi_DHT11();           // 3o byte
        temp_2 = leggi_DHT11();           // 4o byte
        Checksum = leggi_DHT11();         // 5o byte (non usato)

        TEMP = (temp_1 << 8) | temp_2;
        Temperatura = (float) (TEMP);
        Temperatura /= 256.0;            // temperatura misurata

        UMID = (umid_1 << 8) | umid_1;
        Umidita = (float) (UMID);
        Umidita /= 256.0;                // umidità relativa misurata

        invia_dati(Temperatura, Umidita);
        aggiorna_led(Temperatura);
    }
}
/* USER CODE END 4 */
```

VISUALIZZAZIONE E TRASMISSIONE DEI DATI

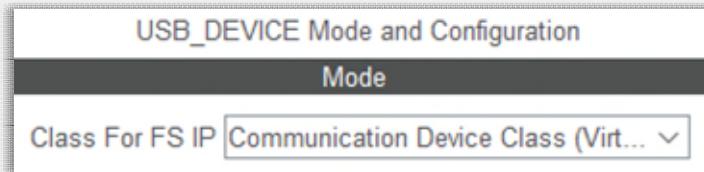
- La funzione «aggiorna_led» consente di visualizzare, in maniera immediata, l'attuale valore di temperatura tramite gli **8 LED** presenti sulla scheda di sviluppo. Il numero di LED accesi è proporzionale alla temperatura misurata secondo i seguenti intervalli:
 - Da 0 a 7°C: 0 LED acceso
 - Da 8 a 14°C: 1 LED acceso
 - Da 15 a 21°C: 2 LED accesi
 - Da 22 a 28°C: 3 LED accesi, e così via.
- La funzione «invia_dati» è responsabile di formattare e trasmettere i dati di temperatura e umidità al PC attraverso un'**interfaccia USB** basata sul **protocollo CDC (Communication Device Class)**. Questo protocollo consente al dispositivo embedded di comunicare con l'host tramite USB, simulando il comportamento di una porta **seriale** tradizionale come la UART. Ciò avviene senza la necessità di installare driver aggiuntivi e utilizzando un comune cavo USB, semplificando così l'interazione tra il dispositivo e il computer in modo immediato e diretto.

CODICE DEL PROGRAMMA (6/7)

```
void invia_dati (float temp, float umid)
{
    char buffer[64];
    int length = sprintf(buffer, "Temperatura: %.2f; Umidita': %.2f\n",temp,umid);
    CDC_Transmit_FS((uint8_t*)buffer, length);
}

void aggiorna_led(float temperatura)
{
    uint8_t num_led_accesi = (uint8_t)(temperatura / 7.0);
    for (int i = 0; i < NUM_LED; i++)          // specifica quanti i LED
    {
        HAL_GPIO_WritePin(LED_PORT, LED_PINS[i], GPIO_PIN_RESET);
    }

    for (int i = 0; i < num_led_accesi; i++) // accende un certo numero di LED
    {
        HAL_GPIO_WritePin(LED_PORT, LED_PINS[i], GPIO_PIN_SET);
    }
}
```



BUZZER

- L'allarme sonoro integrato nel sistema è un buzzer che è in grado di operare con una tensione di alimentazione compresa **tra 3.3V e 5V**.
- È caratterizzato da un **trigger a livello basso**, il che significa che emette un suono quando il pin di controllo viene portato a livello basso (GND) rispetto alla tensione di alimentazione.
- Il buzzer agisce come un **trasduttore elettrico-acustico**: il segnale elettrico applicato induce una vibrazione meccanica nella sua membrana, generando un suono udibile.
- Poiché è **passivo**, richiede solo un segnale di controllo per attivarsi e non necessita di circuiti di guida complessi come quelli dei buzzer attivi, che richiedono forme d'onda specifiche per produrre suoni.
- Nel sistema, il buzzer è gestito tramite un timer **PWM (Pulse Width Modulation)**. Il suono viene emesso variando la frequenza del segnale PWM.
- Il buzzer è programmato per emettere un segnale acustico quando vengono rilevate condizioni fuori da limiti predefiniti, come temperature e umidità critiche.

CODICE DEL PROGRAMMA (7/7)

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // per il buzzer
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    if(Temperatura < 15 || Temperatura > 32 || Umidita < 40 || Umidita > 70)
    {
        for(int x = 10; x < 30; x++)
        {
            HAL_TIM_Set_AUTORELOAD(&htim2, x*2); // modifica la frequenza del PWM
            HAL_TIM_Set_COMPARE(&htim2,TIM_CHANNEL_1, x); // imposta il duty cycle del PWM
            HAL_Delay(100); // attende 100ms
        }
    }
    else HAL_TIM_Set_COMPARE(&htim2, TIM_CHANNEL_1, 0); // spegne il buzzer
    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

PINOUT & CONFIGURATION (3/3)

TIM2 Mode and Configuration

Mode

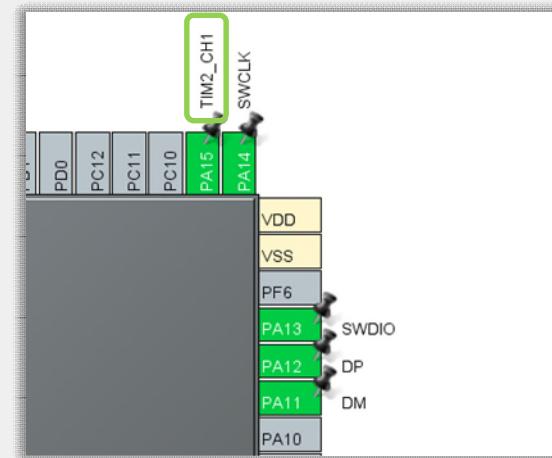
- Slave Mode: Disable
- Trigger Source: Disable
- Clock Source: Internal Clock
- Channel1: PWM Generation CH1
- Channel2: Disable

Configuration

- Reset Configuration
- NVIC Settings
- DMA Settings
- GPIO Settings
- Parameter Settings
- User Constants

Configure the below parameters :

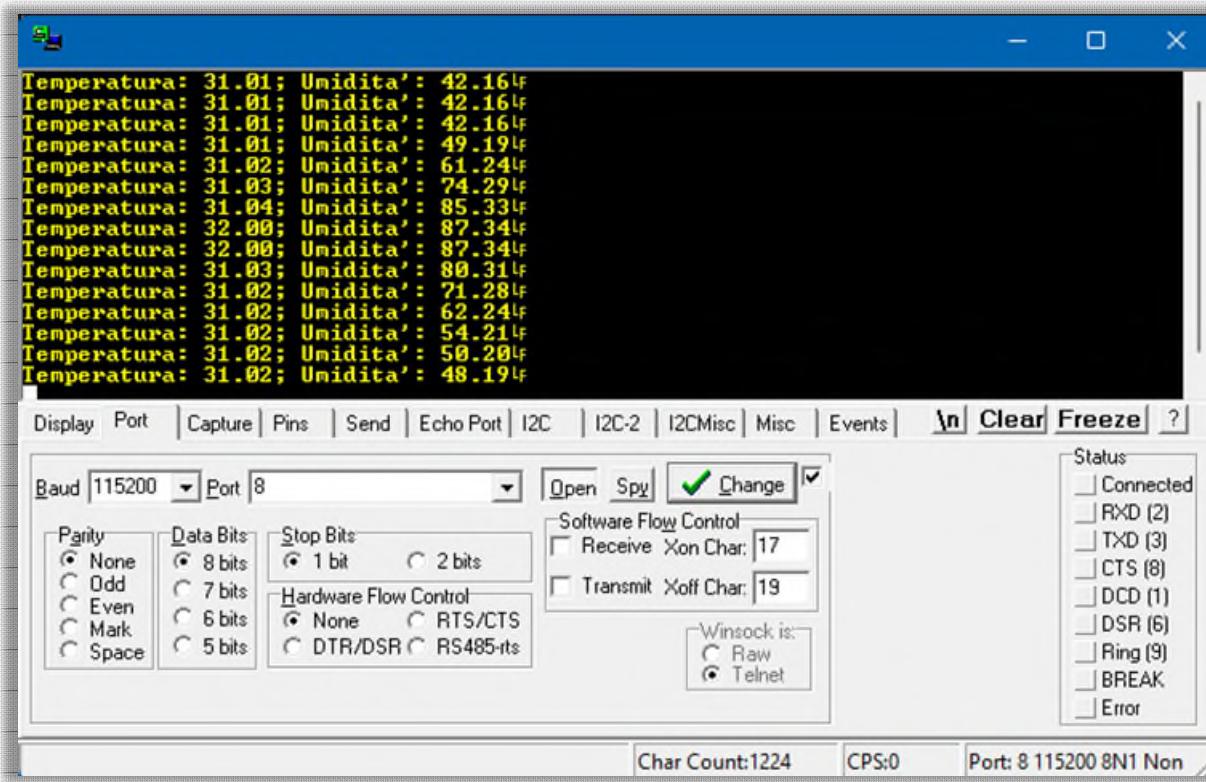
- Search (Ctrl+F)
- Counter Settings
 - Prescaler (PSC - 16 bits value): 128-1
 - Counter Mode: Up
 - Counter Period (AutoReload Reg...): 20
 - Internal Clock Division (CKD): No Division
 - auto-reload preload: Enable



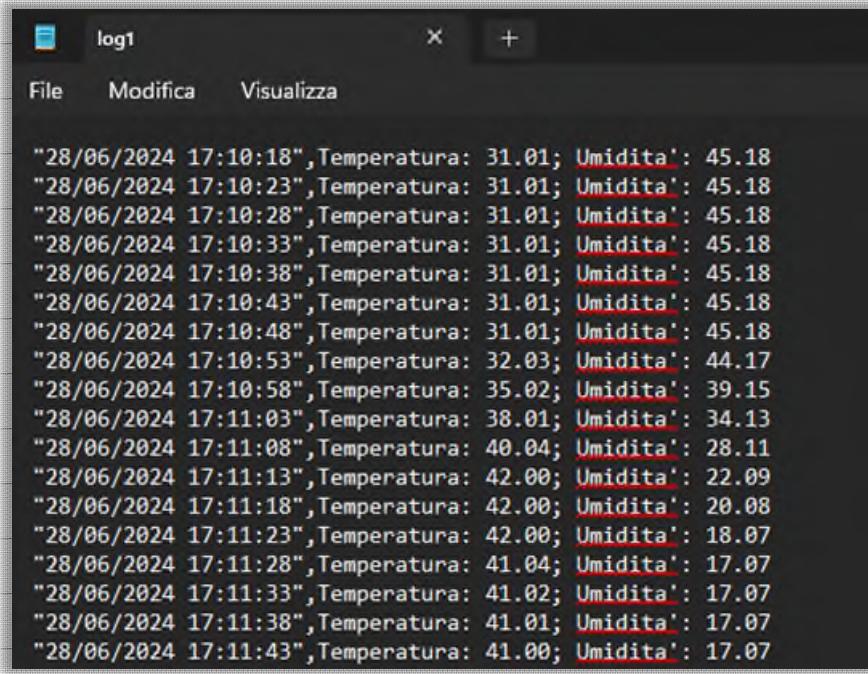
REALTERM (1/3)

- RealTerm è un'applicazione per Windows usata per la comunicazione seriale e il debug. Nel progetto, il software è stata utilizzato per:
 - Ricevere i dati trasmessi dalla board attraverso l'USB CDC.
 - Monitorare in tempo reale i valori di temperatura e umidità misurati.
 - Visualizzare i dati in vari formati, facilitando l'analisi delle informazioni ricevute.
 - Registrare i dati ricevuti in un file di log con timestamp per ulteriori analisi e debugging.

REALTERM (2/3)



REALTERM (3/3)



The screenshot shows a terminal window titled "log1". The window has a dark background and contains a list of log entries. Each entry consists of a timestamp followed by a semicolon and a JSON object containing "Temperatura" and "Umidita'". The timestamps are in the format "28/06/2024 17:10:XX". The "Umidita'" values are underlined, suggesting they are being processed or highlighted.

```
"28/06/2024 17:10:18",Temperatura: 31.01; Umidita': 45.18
"28/06/2024 17:10:23",Temperatura: 31.01; Umidita': 45.18
"28/06/2024 17:10:28",Temperatura: 31.01; Umidita': 45.18
"28/06/2024 17:10:33",Temperatura: 31.01; Umidita': 45.18
"28/06/2024 17:10:38",Temperatura: 31.01; Umidita': 45.18
"28/06/2024 17:10:43",Temperatura: 31.01; Umidita': 45.18
"28/06/2024 17:10:48",Temperatura: 31.01; Umidita': 45.18
"28/06/2024 17:10:53",Temperatura: 32.03; Umidita': 44.17
"28/06/2024 17:10:58",Temperatura: 35.02; Umidita': 39.15
"28/06/2024 17:11:03",Temperatura: 38.01; Umidita': 34.13
"28/06/2024 17:11:08",Temperatura: 40.04; Umidita': 28.11
"28/06/2024 17:11:13",Temperatura: 42.00; Umidita': 22.09
"28/06/2024 17:11:18",Temperatura: 42.00; Umidita': 20.08
"28/06/2024 17:11:23",Temperatura: 42.00; Umidita': 18.07
"28/06/2024 17:11:28",Temperatura: 41.04; Umidita': 17.07
"28/06/2024 17:11:33",Temperatura: 41.02; Umidita': 17.07
"28/06/2024 17:11:38",Temperatura: 41.01; Umidita': 17.07
"28/06/2024 17:11:43",Temperatura: 41.00; Umidita': 17.07
```

