

Air *Humidifier* **STM32**

Computer System Design, A.A 2021/2022

Prof. Nicola Mazzocca

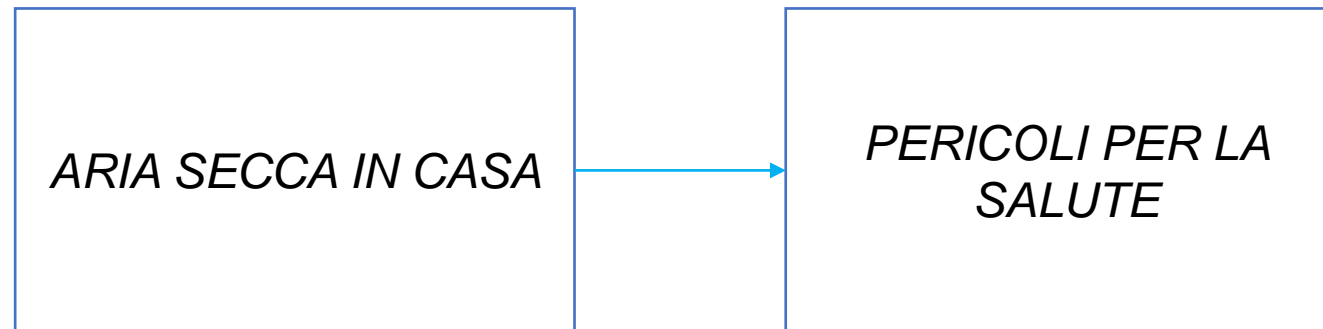
Antonio Romano M63001315

Giuseppe Riccio M63001314



L' idea

L'elettricità statica nei nostri capelli o le scintille che volano quando si tocca qualcuno o qualcosa (e.g. maniglia) in inverno e non solo sono sicuramente segnali che l'aria nella nostra casa è *troppo secca*.



Soluzione? **UMIDIFICATORE**

Quando usare un umidificatore in casa

- L'inverno è il periodo migliore per utilizzare un umidificatore poiché la temperatura sarà bassa e il livello di umidità relativa scenderà al di sotto del 30%-40%.
- Durante la fredda stagione, nelle nostre case accendiamo il riscaldamento. Senza la possibilità di aprire le finestre, l'aria calda non ricircolata asciugherà l'umidità nella stanza lasciando l'aria viziata, secca e scomoda da respirare.

Cosa può compromettere? Alcuni sintomi:

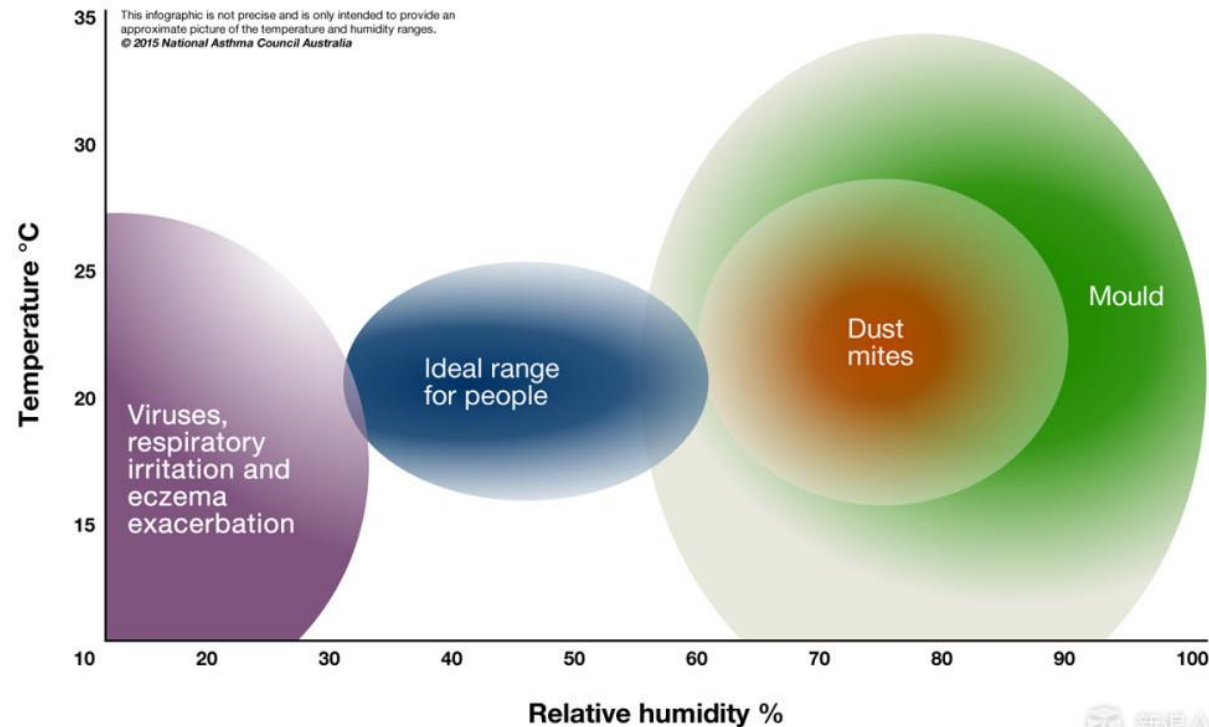
- *Naso che cola cronico*
- *Sintomi di febbre*
- *Asma e riacutizzazioni di allergie*
- *Gola secca*

Occhio anche ai mobili di legno della nostra casa, che possono danneggiarsi deformandosi.






L'obiettivo




L'obiettivo è realizzare un umidificatore in grado di attivarsi in **maniera automatica** e portare dopo un certo tempo di transizione l'umidità nell'ambiente intorno alle percentuali indicate nel seguente infographic della **National Asthma Council Australia**.



Componentistica e listino prezzi

Lista dei componenti utilizzati per la realizzazione dell'**AIR HUMIDIFIER – TOTALE (€111,21 – *€68,95)**

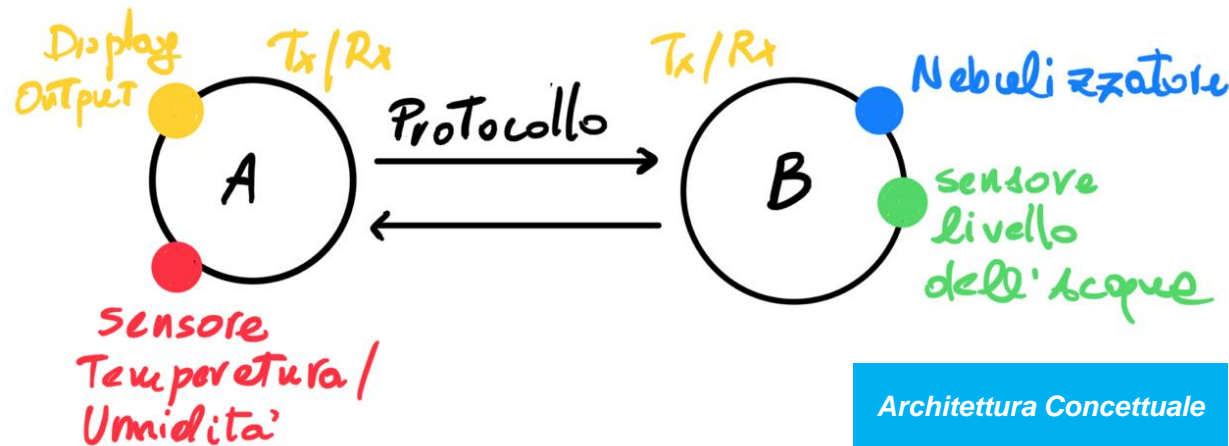
Nome		Prezzo
2xDiscovery Kit STM32F3 DISCOVERY*		€42,26
Sensore Temperatura/Umidità DHT11		€9,99
OLED Display SSD1306		€9,99
PowerBank (Alimentazione)		€7,99

Nome		Prezzo
Kit Elegoo Jumpers		€10,99
Sensore livello dell'acqua		€2.49
Nebulizzatore + Jumper Groove		€14.50
Materiale per struttura prototipo		€13



Requisiti

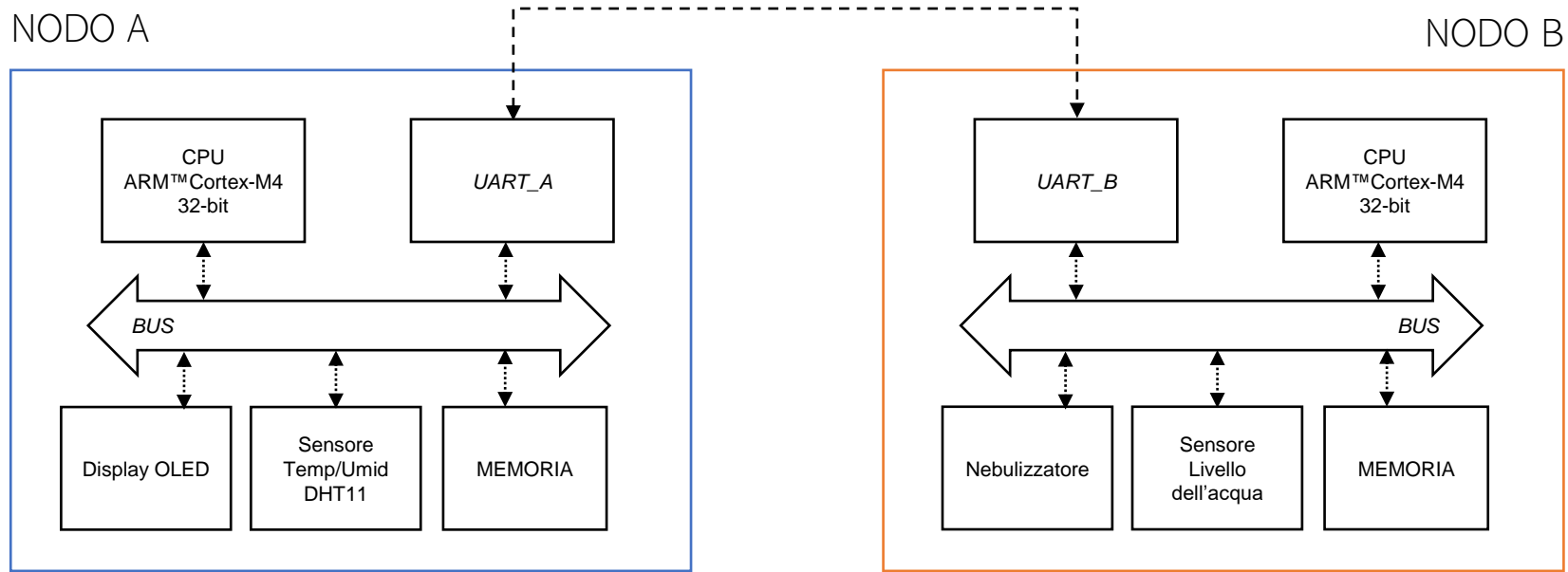
Si realizzeranno in ambiente **STM32Cube** due firmware (Nodo A, Nodo B) per la BOARD della famiglia **STM32F3Discovery** per la gestione di un sistema di umidificazione dell'ambiente.



NODO A: *Trasmette* al nodo B l'accensione del nebulizzatore se non rispetta i vincoli Temp/Umidità
Riceve dal nodo B il livello dell'acqua basso e mostrarlo a video

NODO B: *Trasmette* al nodo A livello dell'acqua insufficiente
Riceve dal nodo A il segnale di abilitazione del nebulizzatore

Architettura Complessiva

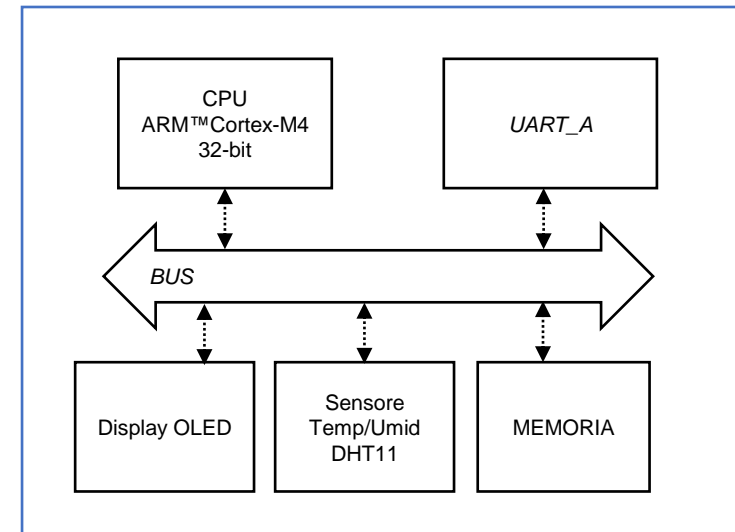


Sezione Nodo A

Nodo A

Analisi delle componenti

NODO A



Sensore Temperatura DHT11

Parti (1/7)

Il **DHT11** è un sensore di temperatura e umidità digitale.

Utilizza un sensore di umidità capacitivo e un termistore per misurare l'aria circostante ed emette *un segnale digitale sul pin dati* (non sono necessari pin di ingresso analogico). È abbastanza semplice da usare, ma richiede una tempificazione accurata tale per acquisire i dati.

Specifiche Tecniche Riassunte:

Alimentazione da 3 a 5 V e I/O

Lecture di umidità del 20-80% con una precisione del 5%.

Lecture di temperatura da 0 a 50 °C con una precisione di ± 2 °C

Dimensioni del corpo 15,5 mm x 12 mm x 5,5 mm

4 pin con spaziatura di 0,1".

A norma RoHS



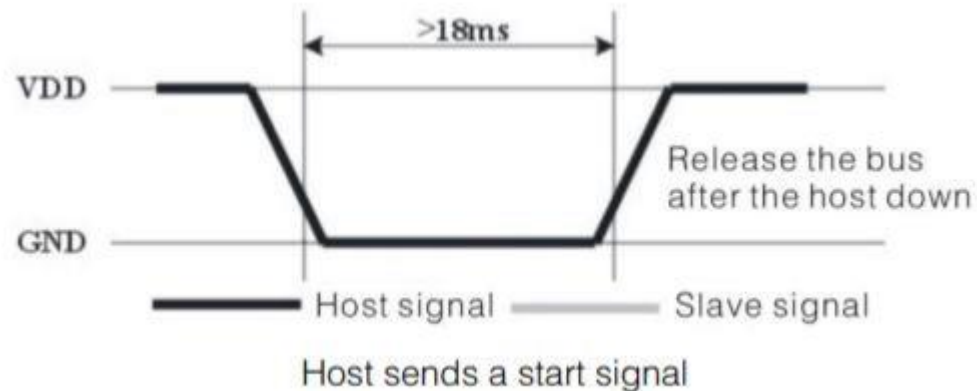
Sensore Temperatura DHT11

Parti (2/7)

Per l'acquisizione dei dati si necessita di un'adeguata tempificazione ed è dunque doveroso utilizzare un timer per fornire una base dei tempi. Si è scelto il **TIM6 (Basic Timers)**.

PROTOCOLLO

Impostazione del pin **DATO** come output.



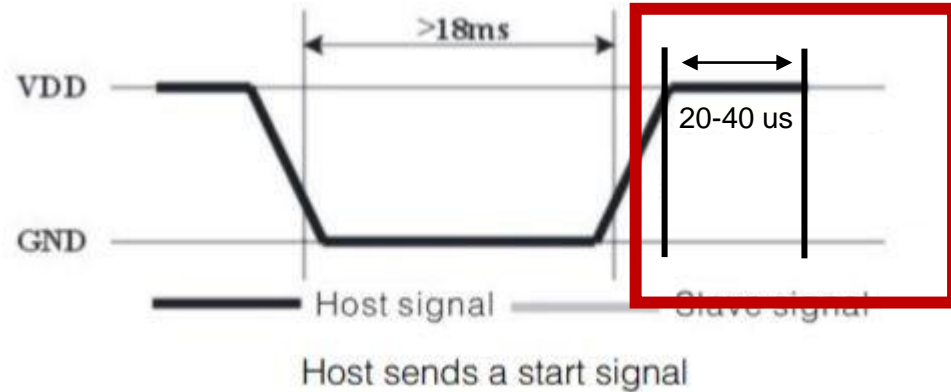
FASE INIZIALIZZAZIONE (**DHT11_Start**):

Per l'invio di un segnale di **START**, si abbassa l'host signal (segnale della Board) per 18 ms.



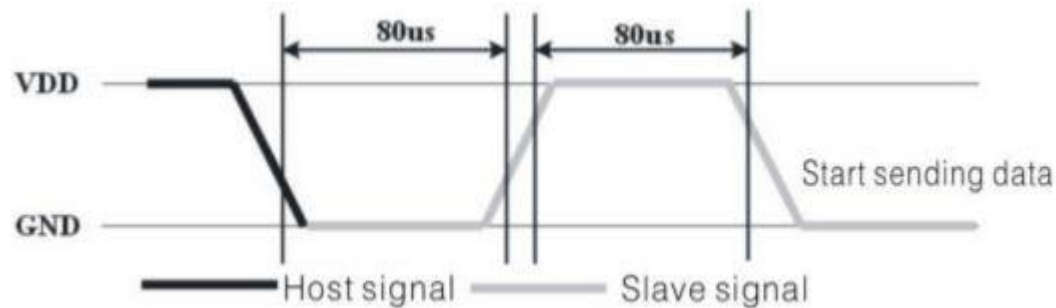
Sensore Temperatura DHT11

Parti (3/7)



FASE FINALE DI INIZIALIZZAZIONE:

Per segnalare la presenza del sensore, dopo aver ricevuto il segnale di start, egli invierà un segnale di presenza (tra i $20\text{ }\mu\text{s}$ – $40\text{ }\mu\text{s}$) prima di passare nella FASE 2.

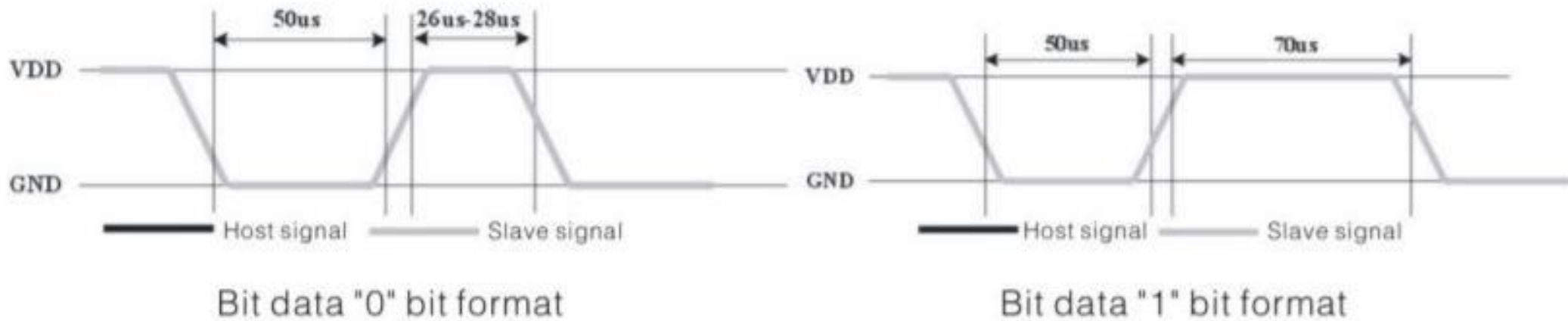


FASE DI RISPOSTA (ACK):

Si riabbassa per $80\text{ }\mu\text{s}$ (slave signal, segnale del sensore) e si rialza subito per $80\text{ }\mu\text{s}$ riabbassandola di nuovo. Fatto ciò il sensore verrà inizializzato e inizierà a trasmettere.

Sensore Temperatura DHT11

Parti (4/7)



FASE DI TRASMISSIONE DEI DATI ([DHT11_Read](#)):

Il sensore invierà **40 bit di dati**. La trasmissione di OGNI bit inizia con un livello di bassa tensione che dura 50 μs. Successivamente ad esso se il livello di alta tensione è:

- Circa 26-28 us, il bit è **0**
- Circa 70 us, il bit è **1**

Sensore Temperatura DHT11

Parti (5/7)



FORMATO DEI DATI

1° byte: dati di umidità relativa in % (parte intera)

2° byte: dati decimali sull'umidità relativa in % (parte frazionaria)

3° byte: temperatura in gradi Celsius (parte intera)

4° byte: temperatura in dati decimali in % (parte frazionaria)

5° byte: checksum (RHI+RHD+TCI+TCD = Sum)

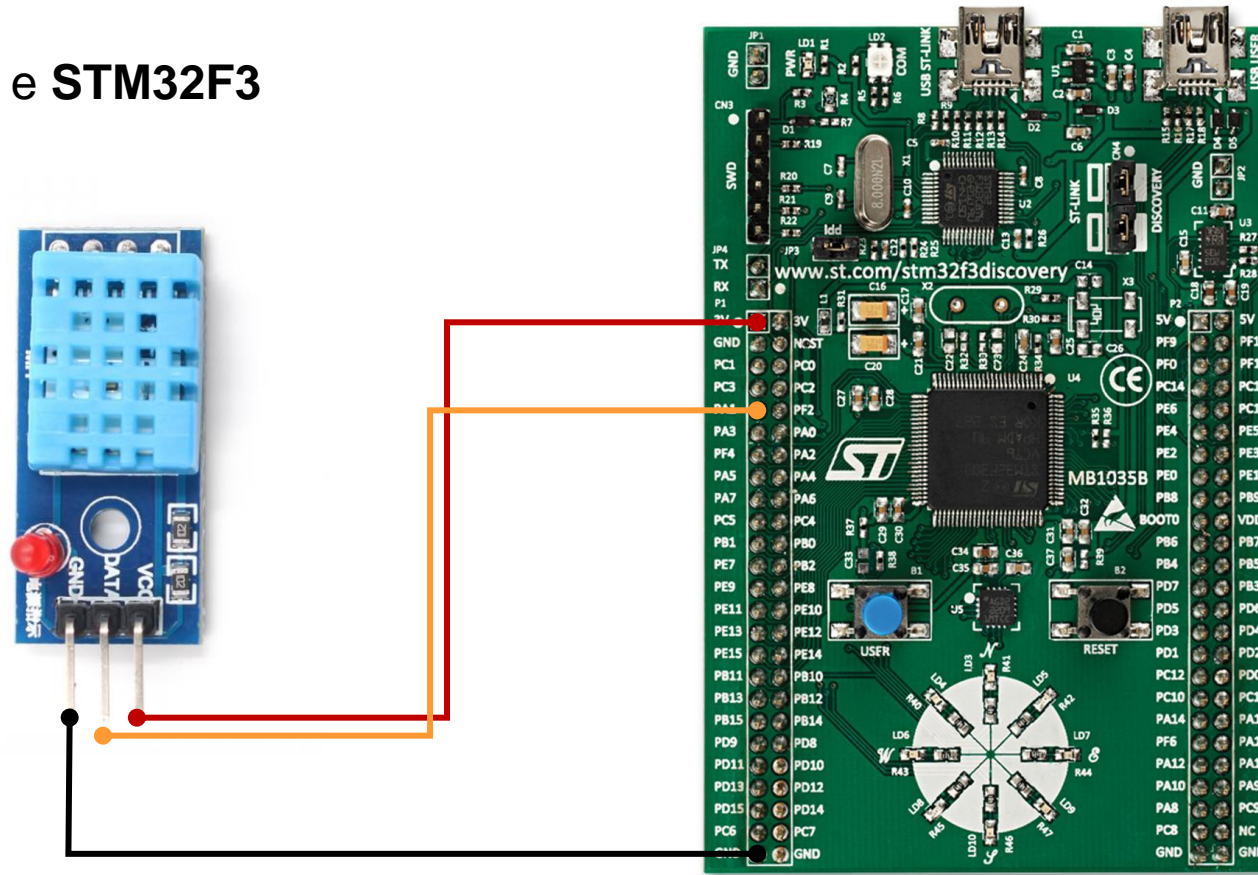
RH = Relative Humidity in % - T = Temperature in Deg. C



Sensore Temperatura DHT11

Parti (6/7)

Interconnessione DHT11 e STM32F3



Implementazione DHT11 su STM32Cube

Parti (7/7)

Si è scelto di utilizzare come base dei tempi il TIM6, in quanto insieme al TIM7, questi timer sono utilizzati per attivare i convertitori da digitale ad analogico (DAC) o come base di tempi.

In particolare, come è possibile vedere dalla configurazione a lato, il timer è usato come contatore a 16 bit, in modalità di funzionamento **Up counting**, ovvero il contatore parte da zero e si incrementa fino a un certo valore fissato.

Per gestire opportunamente il conteggio di tali timer secondo le nostre esigenze di progetto, abbiamo utilizzato il **prescaler**, che è semplicemente un **divisore di frequenza**.

Il TIM6 ha come clock source il clock interno della scheda (CK_INT), il prescaler configurato al valore di 47 ed è settato in modalità up counting. Con queste impostazioni, il periodo del clock è di 1 us.

$$Periodo = \frac{Prescaler + 1}{Freq_{clock_{Board}}}$$

TIM6 Mode and Configuration

Mode

☒ Activated

☐ One Pulse Mode

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC -... 48-1

Counter Mode Up

Counter Period (... 65535

auto-reload preload Disable

Trigger Output (TRGO) ...

Trigger Event Sel... Reset (UG bit from TIMx_E...



Implementazione DHT11 su STM32Cube

Parti (7/7)

Una volta configurato il TIM6, ed abilitate le interruzioni su di esso all'interno del NVIC.

Timer 6 interrupt and DAC underrun interrupts	<input checked="" type="checkbox"/>	0
-----------------------------------------------	-------------------------------------	---



0

A questo punto è possibile avviare il protocollo con ***DHT11_Start()*** per inizializzare e leggere i valori dal sensore di temperatura/umidità DHT11, in particolare, viene utilizzata la funzione ***delay()*** per implementare la funzione di attesa attiva.

```
void delay (uint16_t time)
{
    __HAL_TIM_SET_COUNTER(&htim6, 0);
    while ((__HAL_TIM_GET_COUNTER(&htim6))<time);
}
```

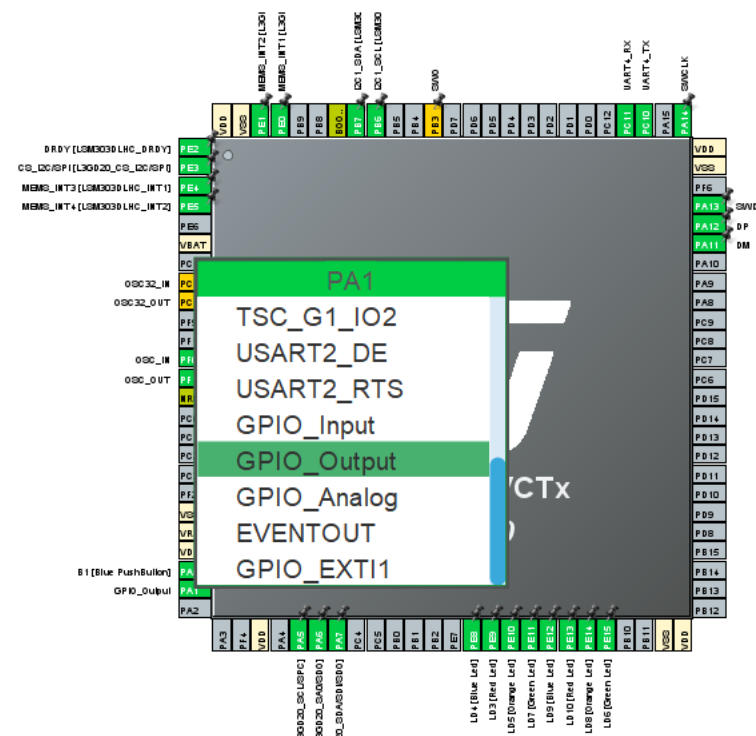
Questa funzione presenta come parametro di input il valore in microsecondi relativo al tempo che vogliamo attendere ed implementa una breve attesa attiva.



Parti (7/7)

Si è scelto come GPIO, il pin **PA1** e si è dunque impostato come **GPIO_Output** all'interno della configurazione del nodo A su STM32Cube.

```
#define DHT11_PORT GPIOA
#define DHT11_PIN GPIO_PIN_1
```



Implementazione DHT11 su STM32Cube

Parti (7/7)

Una volta inizializzato il sensore, tramite la funzione ***DHT11_Start()***, che non fa altro che replicare il protocollo visto nelle precedenti slide.

È possibile leggere i valori di temperatura ed umidità richiamando la funzione ***DHT11_Read()***, la quale alla prima lettura restituisce l'umidità (parte intera), e mano mano tutti gli altri valori, se la somma dei valori ottenuti è uguale al valore di checksum, allora posso prendermi i valori correnti di temperatura ed umidità.

```
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)
    {
        // TEMPERATURA IN GRADI CELSIUS
        tCelsius = (float)TCI + (float)(TCD/10.0);

        // TEMPERATURA IN GRADI FAHRENHEIT
        //tFahrenheit = tCelsius * 9/5 + 32;

        // UMIDITA' IN PERCENTUALE
        RH = (float)RHI + (float)(RHD/10.0);
    }
}
```



OLED Display SSD1306

Parti (1/6)

Si è scelto il display SSD1306 per la sua versatilità in termini di dimensioni ed interfaccia (I2C), in quanto a differenza di altri display che sfruttano collegamenti paralleli su 8 pin, questo ha solamente 4 pin.

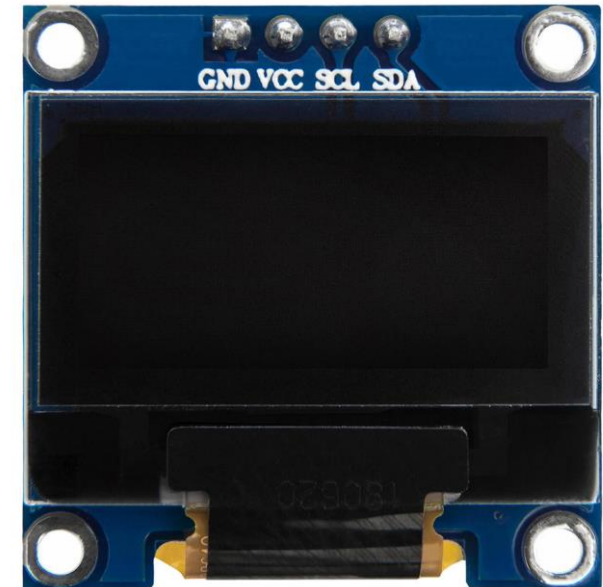
Specifiche Tecniche Riassunte:

Alimentazione da 3 a 5 V e I/O

Dimensioni matrice display 128 x 64

4 pin

Interfaccia I2C



OLED Display SSD1306

Parti (2/6)

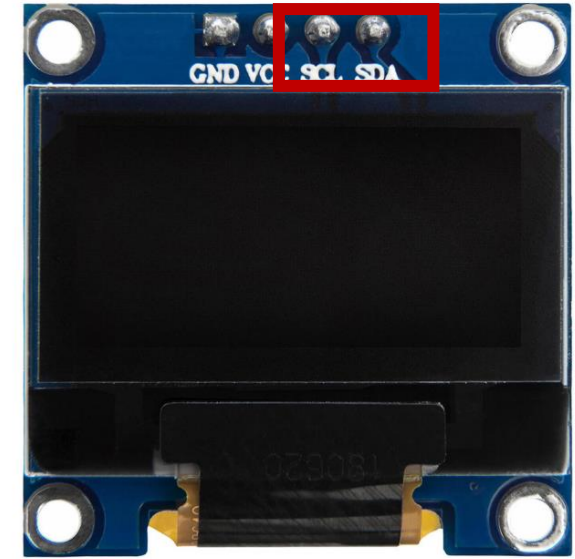
Per interfacciare la BOARD con il Display ci avvaliamo dell'interfaccia I2C.

Richiede due linee seriali di comunicazione:

- **SDA**(Serial DAta) per i dati
- **SCL**(Serial CLock) per il clock

Lo stato di riposo delle linee è il valore logico alto.

Ai due segnali SDA e SCL viene aggiunta una terza linea, VDD a cui sono connessi i rispettivi SDA e SCL.



La particolarità dell'I2C è permettere di utilizzare sulla stessa interfaccia più device. In modo tale da diminuire il numero di interconnessioni.

Per supportare la possibilità di ospitare più device master, questi ultimi pilotano il bus solo quando devono cominciare una connessione (**Arbitraggio**).

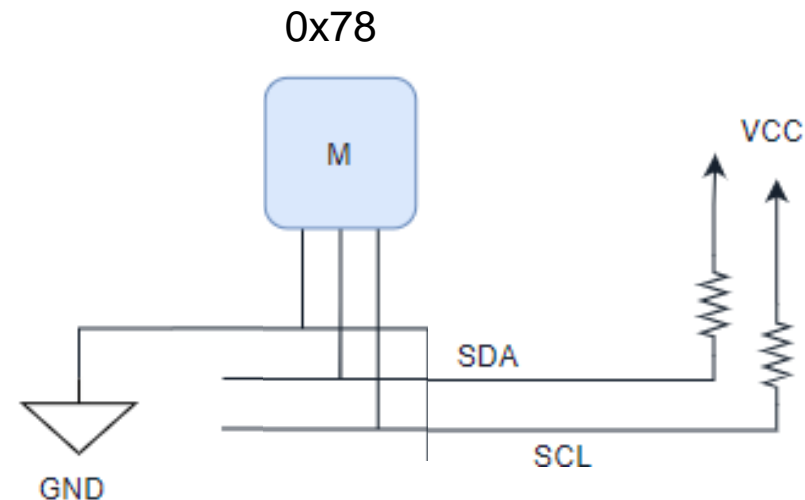


OLED Display SSD1306

Parti (3/6)

Ogni device connesso al bus è identificato da un indirizzo a 7 bit.

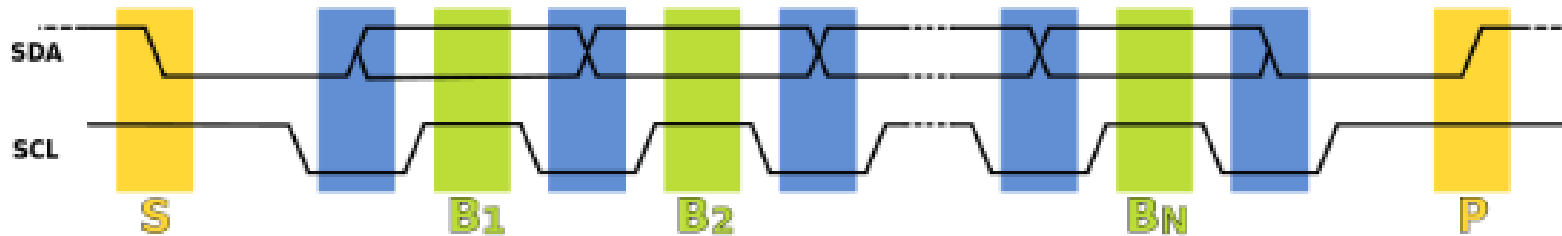
Nel caso in esame, si ha soltanto il display e pertanto si mostra nello schema un solo device che fungerà anche da master. In particolare, **l'indirizzo del display** in questione è **0x78**, tale indirizzo è configurabile nel caso in cui esistesse un altro dispositivo con il medesimo identificativo.



OLED Display SSD1306

Parti (4/6)

Essendo che si è connesso un solo master senza slave, si mostra il seguente protocollo:



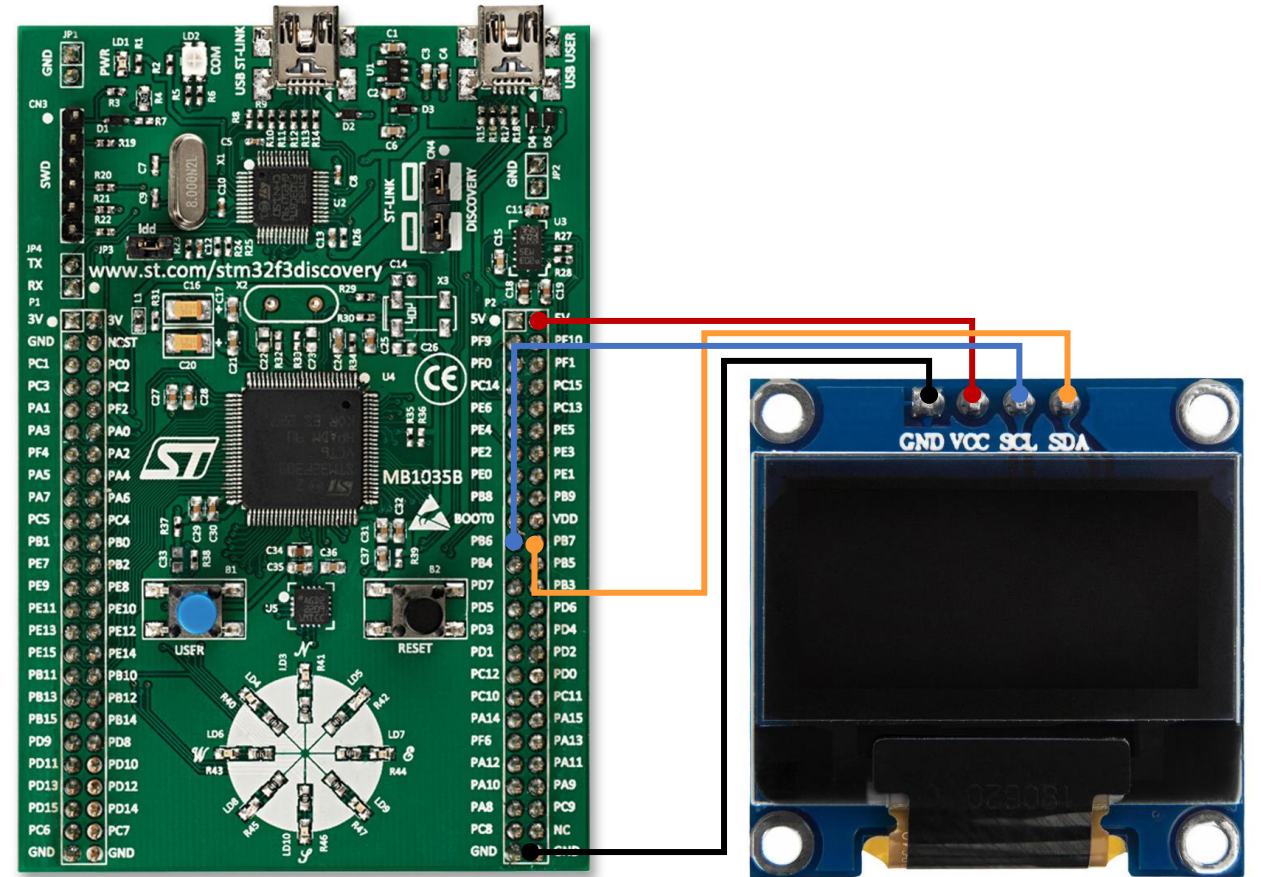
- **S** è lo **START** bit.
- La linea **SDA** viene forzata bassa dal master mentre il clock **SCL** è a livello logico alto.
- Il master (dopo un certo tempo) porta **SCL** a livello logico basso (primo tratto in blu).
- Il master impone sul segnale SDA il valore del primo bit (B1).
- La **commutazione** di SCL indica che il dato è stabile e può essere letto (tratto in verde).
- Si continua trasmettendo tutti gli altri bit.
- La transazione termina con lo **STOP** bit (P).
- **SDA** viene commutato da basso ad alto quando SCL è alto.



OLED Display SSD1306

Parti (5/6)

Interconnessione OLED SSD1306 e STM32F3



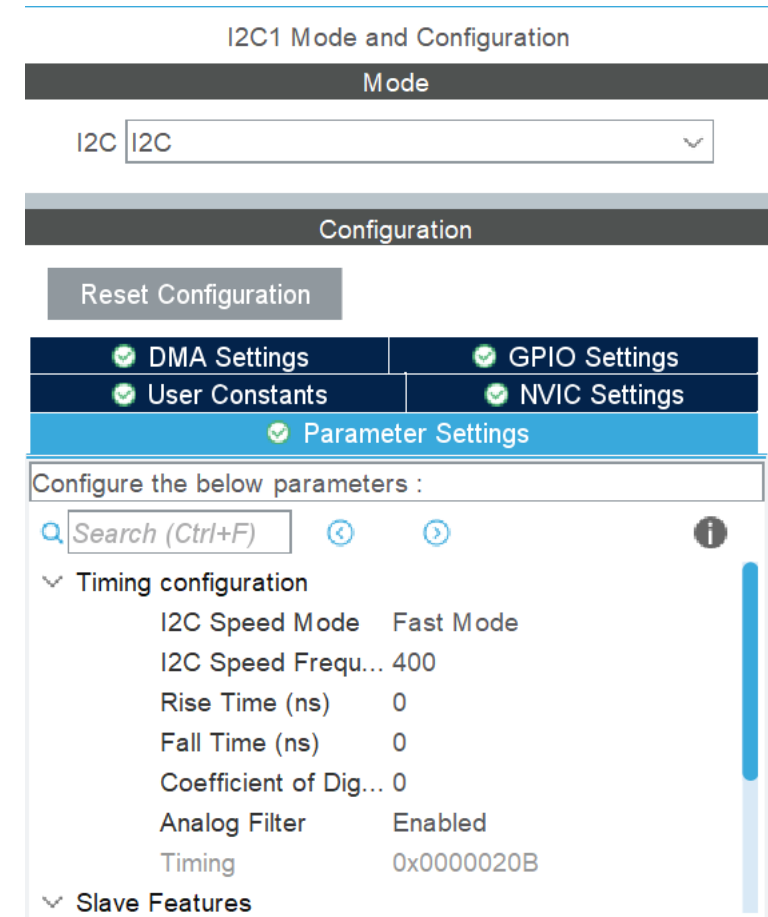
Implementazione OLED Display su STM32Cube

Parti (6/6)

Per usare il display OLED SSD1306, occorre abilitare l'interfaccia I2C all'interno della configurazione di STM32Cube per la BOARD STM32F3Discovery.

Questa interfaccia di default è settata in modalità **Fast mode** e con una frequenza di lavoro pari a 400 KHz.

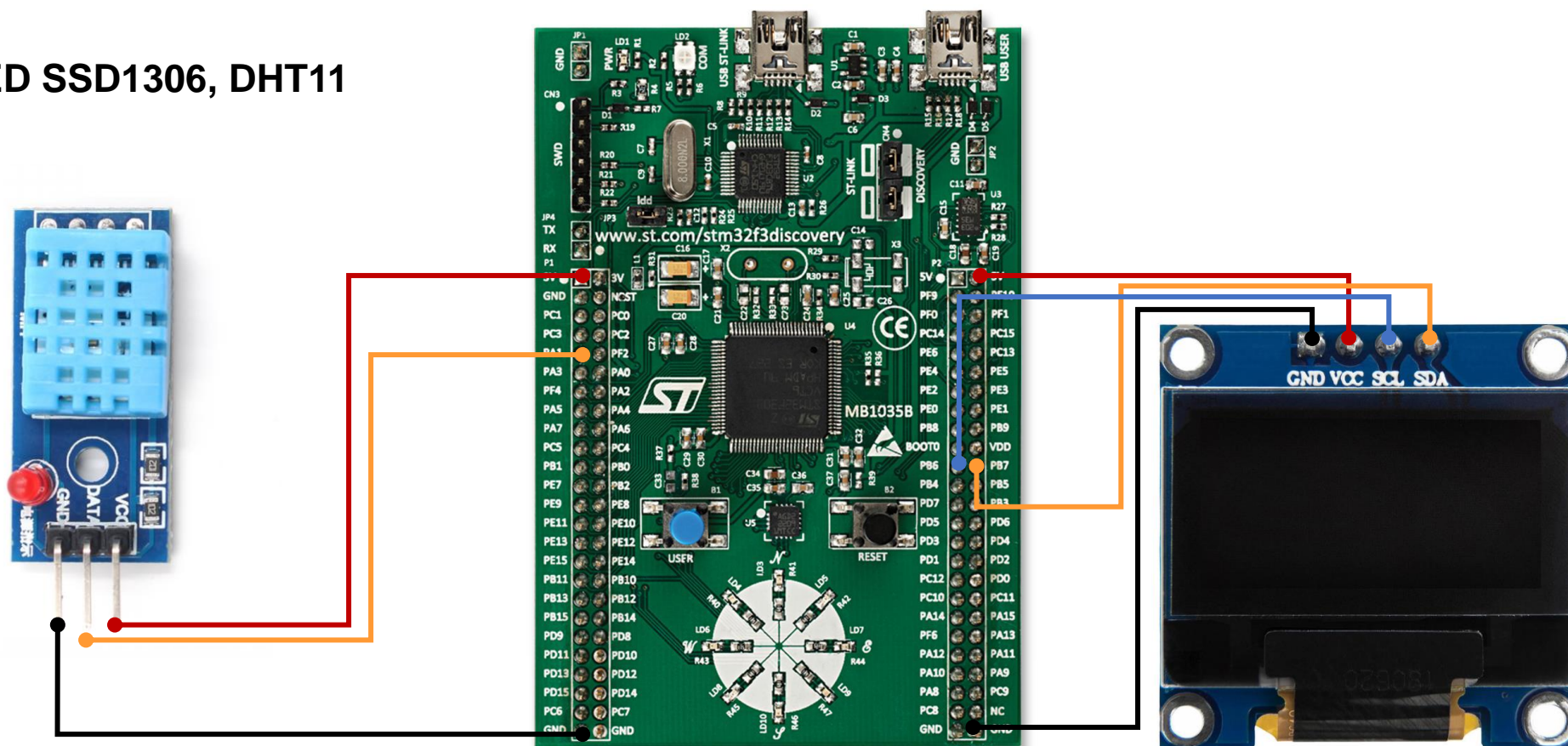
Una volta abilitata l'interfaccia, abbiamo usato le funzioni di libreria messe a disposizione dal costruttore del display stesso per la sua inizializzazione ed utilizzo.



Interconnessione completa Nodo A

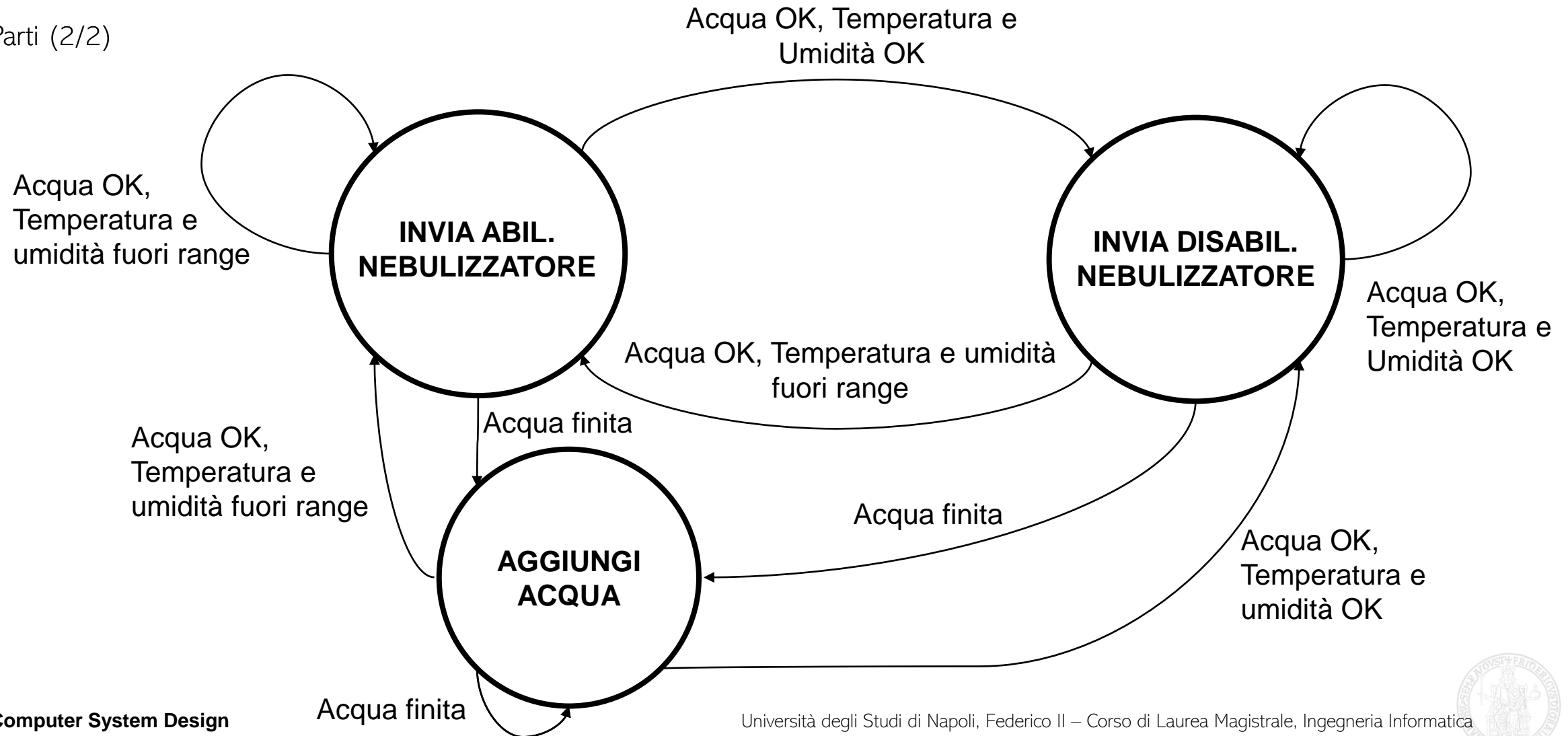
Parti (1/2)

Interconnessione **OLED SSD1306**, **DHT11**
e **STM32F3**



Automa Nodo A

Parti (2/2)

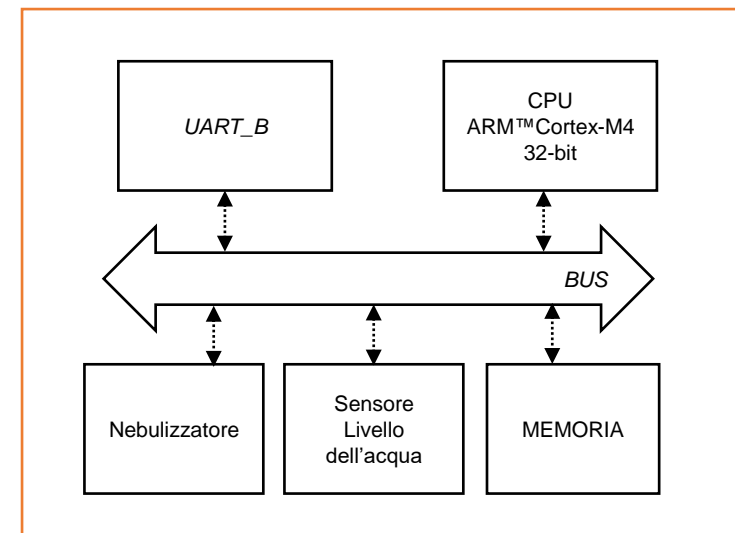


Sezione Nodo B

Nodo B

Analisi delle componenti

NODO B



Sensore Livello dell'Acqua

Parti (1/5)

Il **WaterLevelSensor** è un sensore per la rilevazione dell'acqua. Viene utilizzato per misurare il livello dell'acqua all'interno di un contenitore.

Si tratta di un sensore di livello dell'acqua di tipo conduttivo, in cui la variazione della resistenza di fili paralleli su diverse profondità dell'acqua viene convertita in una tensione.

Specifiche Tecniche D'interesse:

Alimentazione 5 V

Corrente d'esercizio: < 20mA

Tipo di sensore: Analogico

Superficie: 40mm x16mm

Temperatura d'esercizio: 10°C – 30°C

Umidità di funzionamento: 10% - 90%

Dimensioni del prodotto: 65mm x 20 mm x 8 mm

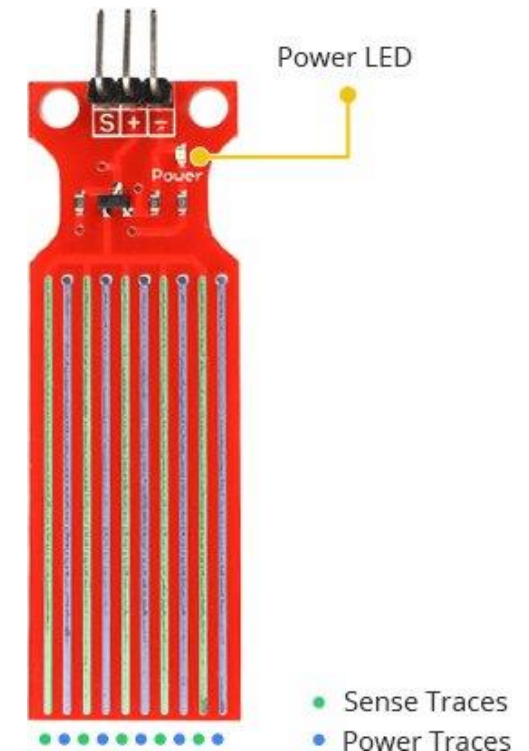


Sensore Livello dell'Acqua

Parti (2/5)

Il Sensore ha una serie di dieci «Traces» parallele di rame, cinque delle quali sono **traces di alimentazioni** e altre sono **traces sensoriali**.

Queste traces non sono collegate inizialmente, ma appena sono sommerse in acqua si crea un collegamento tra di esse che genera una variazione di resistenza.



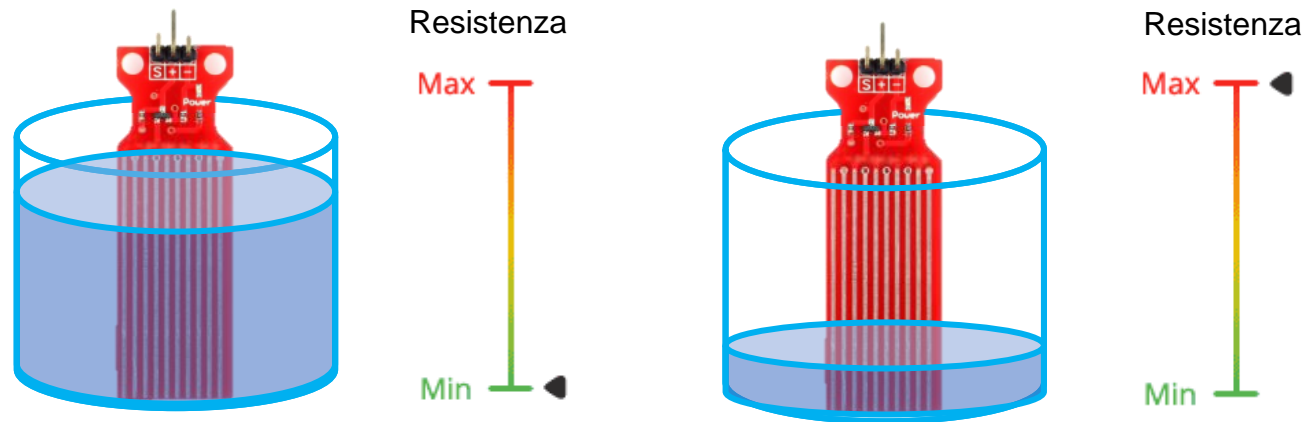
Sensore Livello dell'Acqua

Parti (3/5)

Tale resistenza è inversamente proporzionale all'altezza dell'acqua:

- Più acqua è immersa nel sensore, **maggiore** sarà la conduttività e minore sarà la resistenza.
- Meno acqua è immersa nel sensore, **minore** sarà la conduttività e maggiore sarà resistenza.

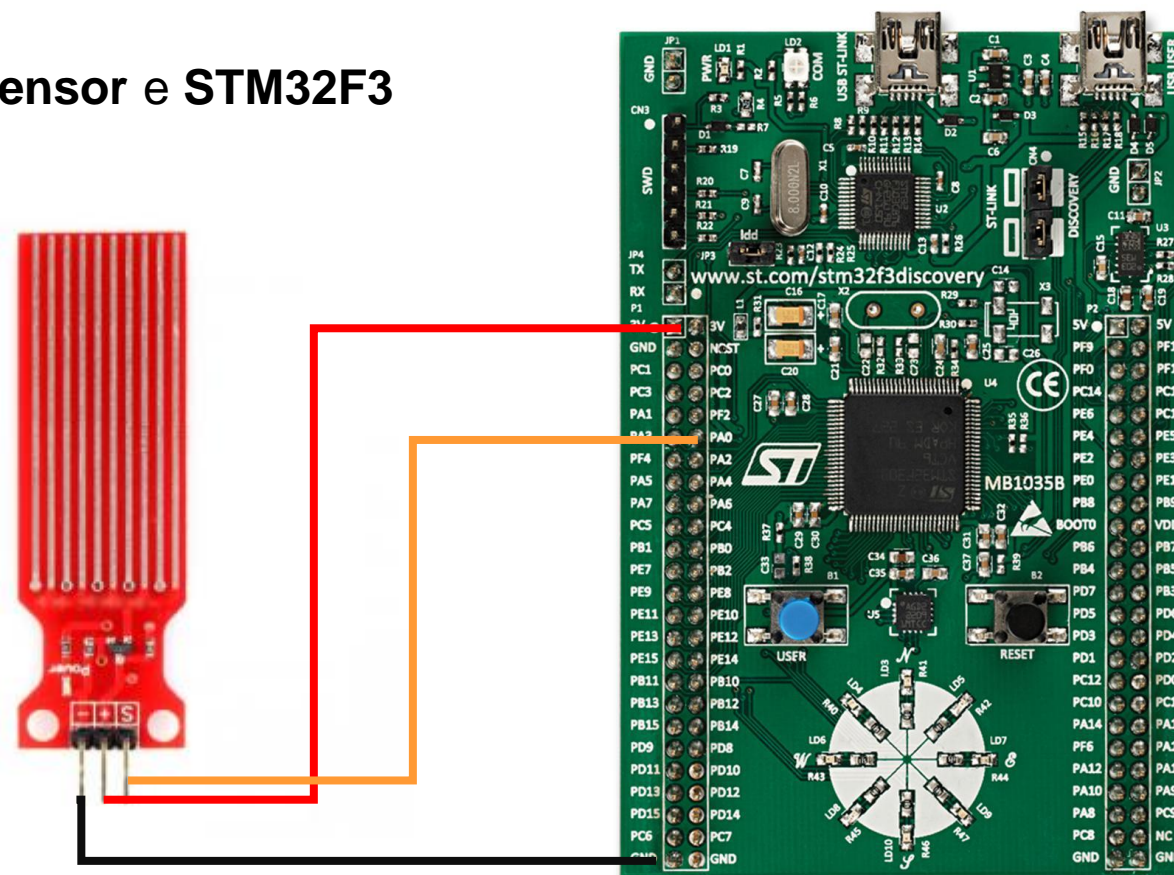
Il sensore produce, dunque, una tensione di uscita in base alla resistenza variabile attraverso la quale è possibile determinare il livello dell'acqua all'interno del contenitore.



Sensore Livello dell'acqua

Parti (4/5)

Interconnessione **WaterLevelSensor** e **STM32F3**



Implementazione sensore Livello dell'acqua su STM32Cube

Parti (5/5)

Al fine di utilizzare il sensore di livello dell'acqua occorre configurare sulla BOARD STM32F3Discovery un ADC (Convertitore Analogico Digitale) in modalità **Single-ended**. Su questo componente si abilitano anche le **interruzioni** in maniera tale che al termine di una conversione sia possibile gestire i valori ottenuti.

Per fare ciò, serve individuare un GPIO che fungerà da pin DATA tramite il quale leggere il valore da convertire (**ADC_In**), il GPIO individuato è il **PA0**, dunque si configurerà tale pin come nella slide successiva.

The screenshot shows the 'ADC1 Mode and Configuration' window in STM32CubeMX. It is divided into two main sections: 'Mode' and 'Configuration'.

Mode Section:

- IN1: IN1 Single-ended (dropdown)
- IN2: Disable (dropdown)
- IN3: Disable (dropdown)
- IN4: Disable (dropdown)

Configuration Section:

- A 'Reset Configuration' button is present.
- Below it are four settings, each with a green checkmark icon:
 - DMA Settings
 - User Constants
 - GPIO Settings
 - NVIC Settings
- A 'Parameter Settings' button is located below these.
- At the bottom, there is a table for 'NVIC Interrupt Table':

NVIC Interrupt Table	Enabled	Preemption Priority
ADC1 and ADC2 interrupts	<input checked="" type="checkbox"/>	0

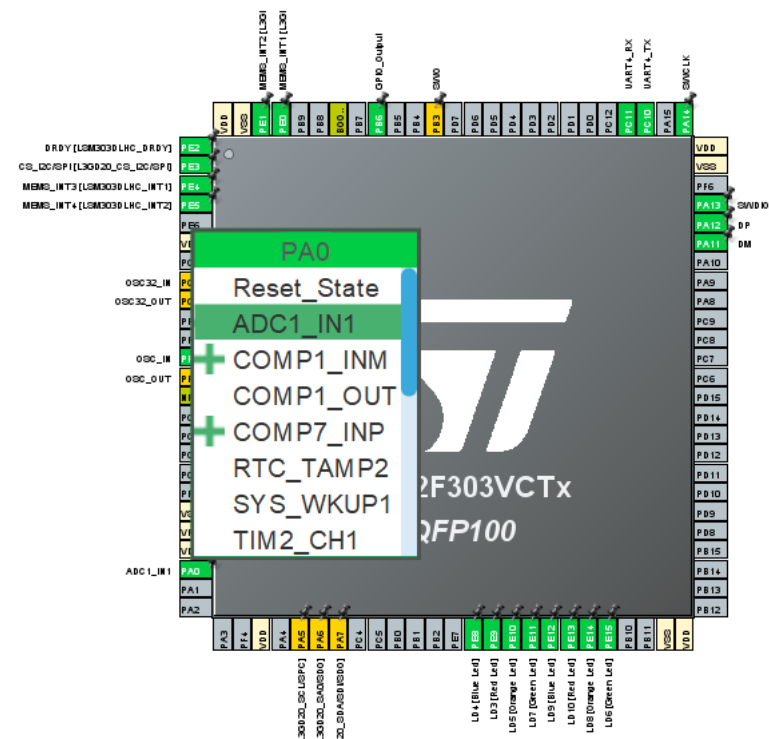


Implementazione sensore Livello dell'acqua su STM32Cube

Parti (5/5)

Una volta configurato il **PA0**, si procede alla gestione della **Callback** per salvare il valore ottenuto dalla conversione analogica-digitale in maniera opportuna.

```
void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef* hadc)
{
    adc_value = HAL_ADC_GetValue(hadc); //valore di tensione
    if (adc_value > 200){
        flag = 1;
    } else {
        flag = 0;
    }
}
```



In particolare, in seguito, a varie sperimentazioni si è scelto come valore di soglia per la rilevazione dell'acqua, il valore di 200.



Nebulizzatore e interfaccia GROVE

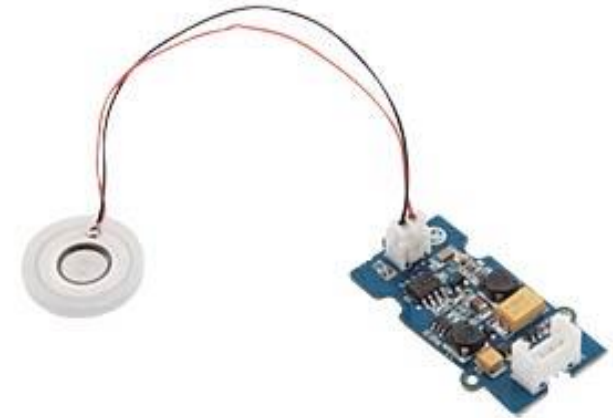
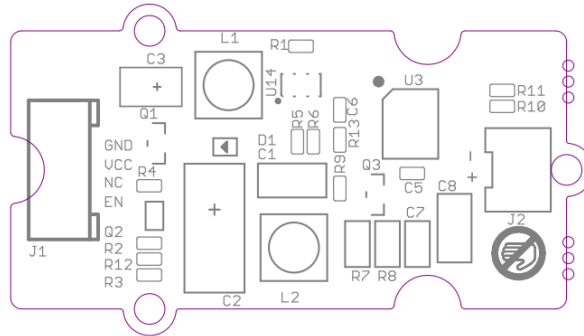
Parti (1/3)

Il Nebulizzatore è un atomizzatore ad ultrasuoni, non fa altro che ridurre un liquido in minutissime goccioline. Per usarlo, si fa uso dell'interfaccia GROVE.

L'interfaccia GROVE è largamente usata nell'ambito della prototipazione, con schede Arduino in prevalenza, e fa leva sulla semplicità nell'effettuare i collegamenti, nonché sulla semplicità di programmazione.

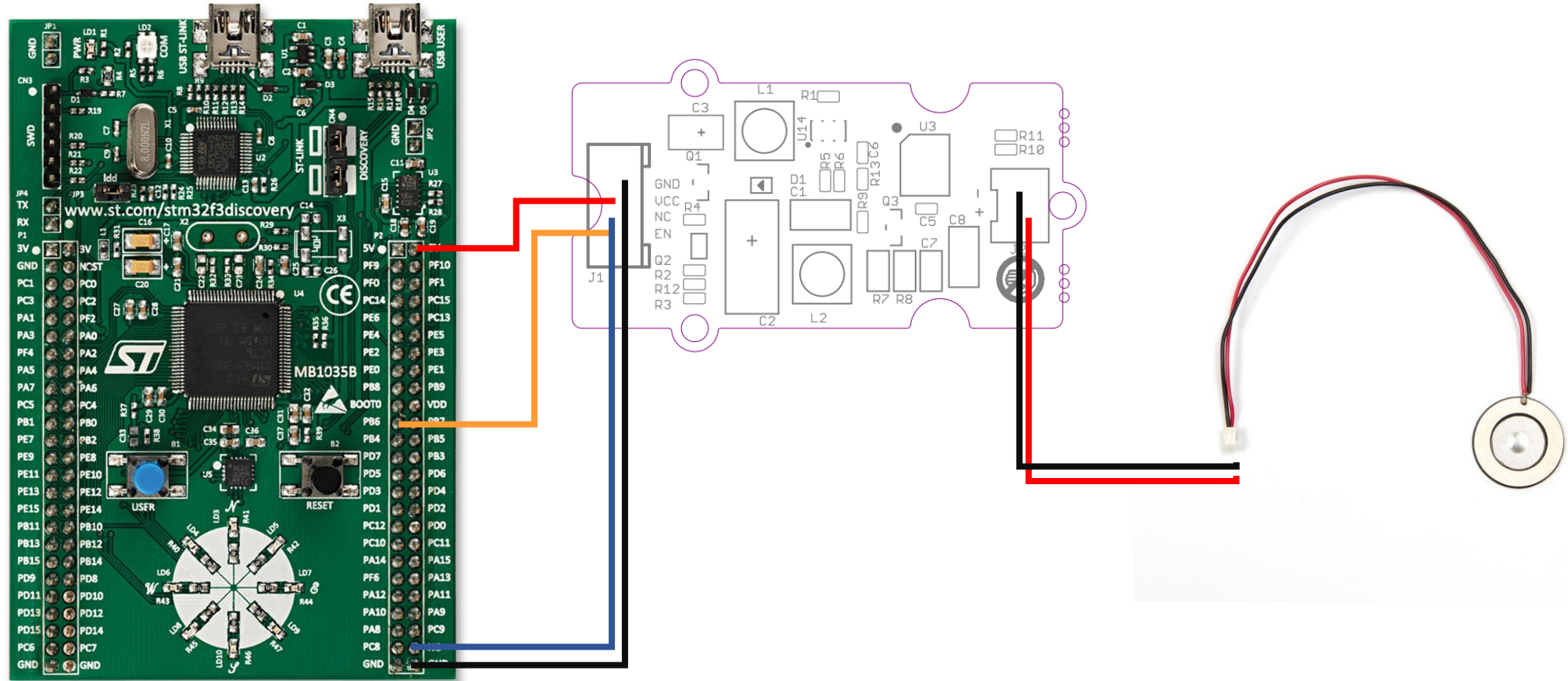
In particolare, quest'interfaccia è composta da 4 pin:

- **GND**
- **VCC**
- **NC (not connected)**
- **EN (enable)**



Nebulizzatore e interfaccia GROVE

Parti (2/3)

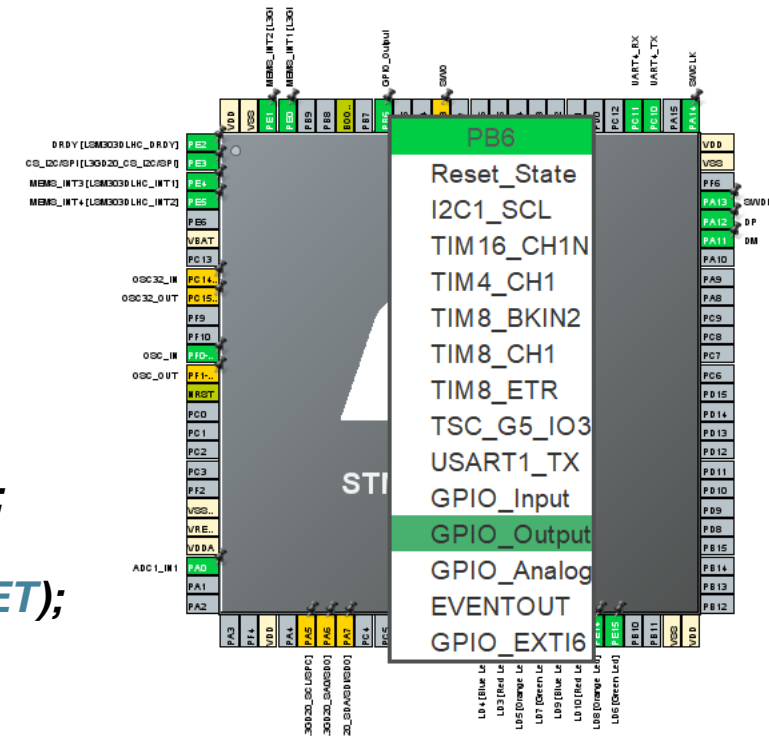


Implementazione Nebulizzatore su STM32Cube

Parti (3/3)

Per la gestione del nebulizzatore non occorre configurare nessun componente, ma semplicemente si usa un GPIO, in questo caso il **PB6**, come **GPIO_Output** questo pin in particolare è collegato all'**ENABLE** del (GROVE) nebulizzatore.

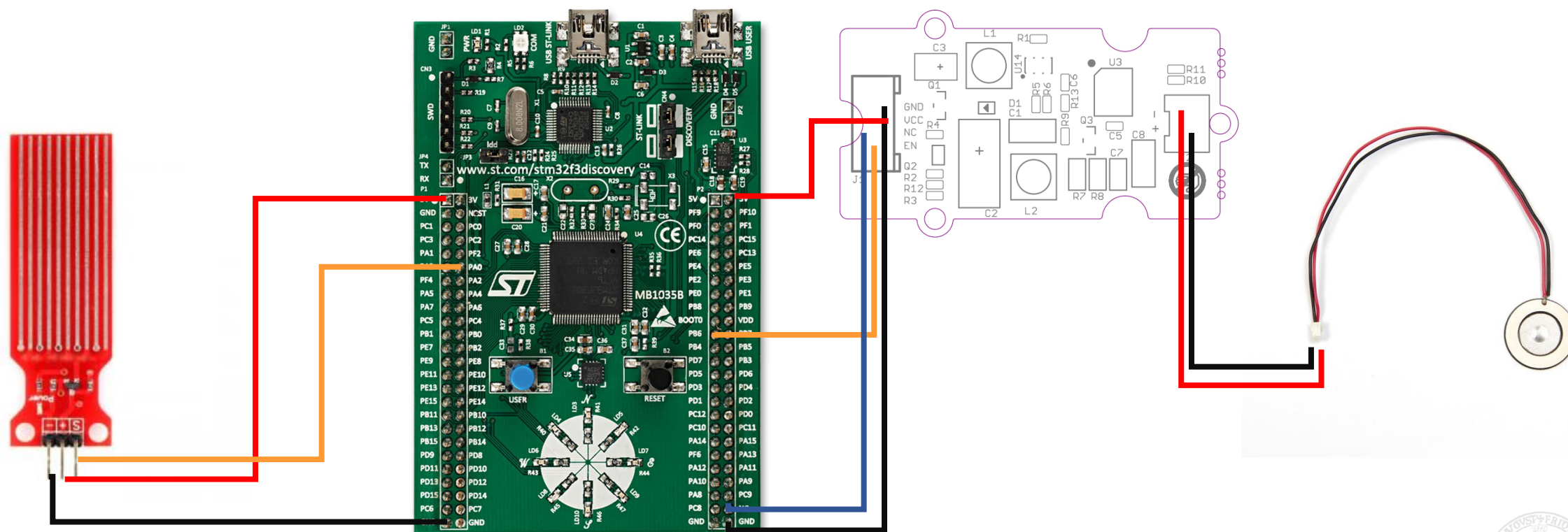
```
void gestisci_nebulizzatore (int acceso)
{
    if (flag == 1 && acceso == 1)
    {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
    }
    else
    {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
    }
}
```



Interconnessione completa Nodo B

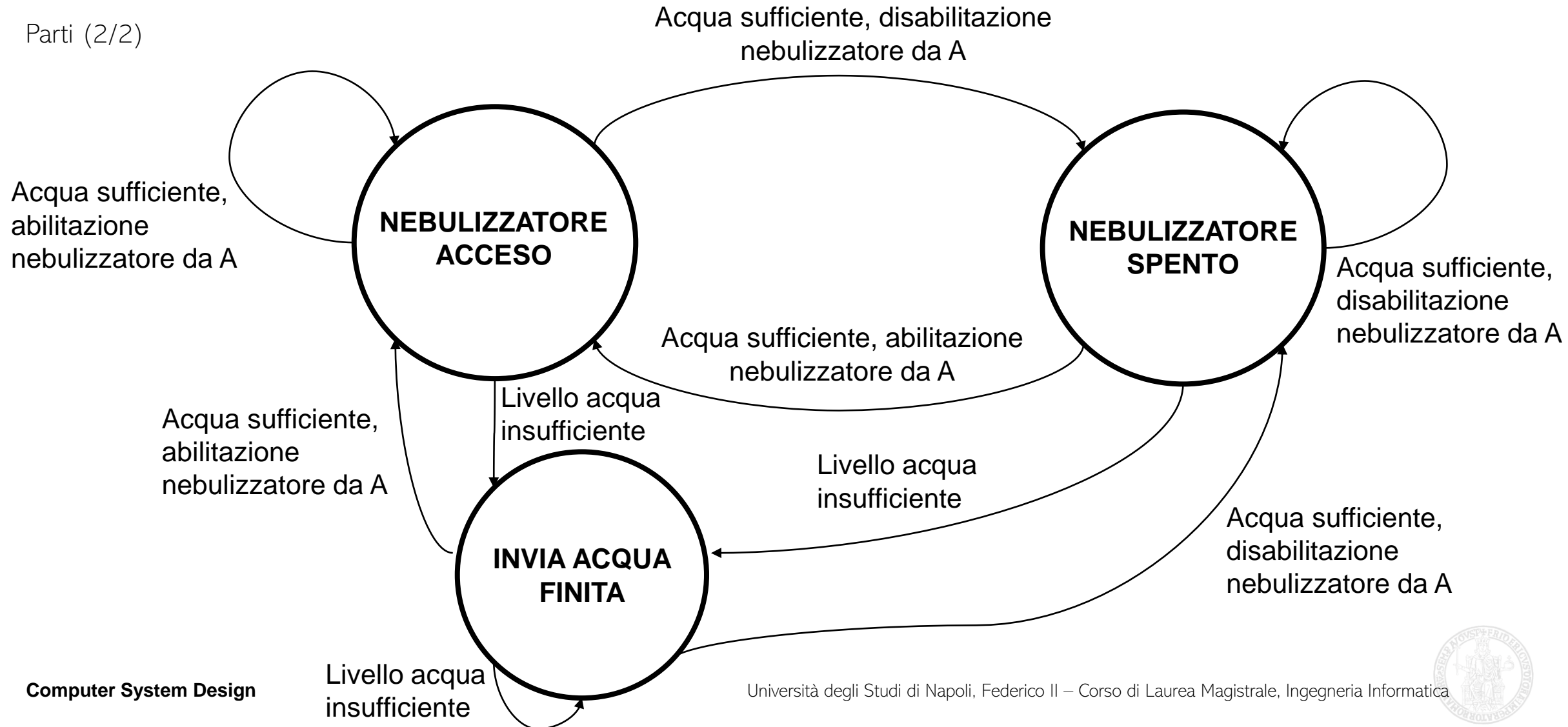
Parti (1/2)

Interconnessione **WaterLevelSensor**, **Nebulizzatore**
e **STM32F3**



Automa Nodo B

Parti (2/2)



Sezione Interconnessione

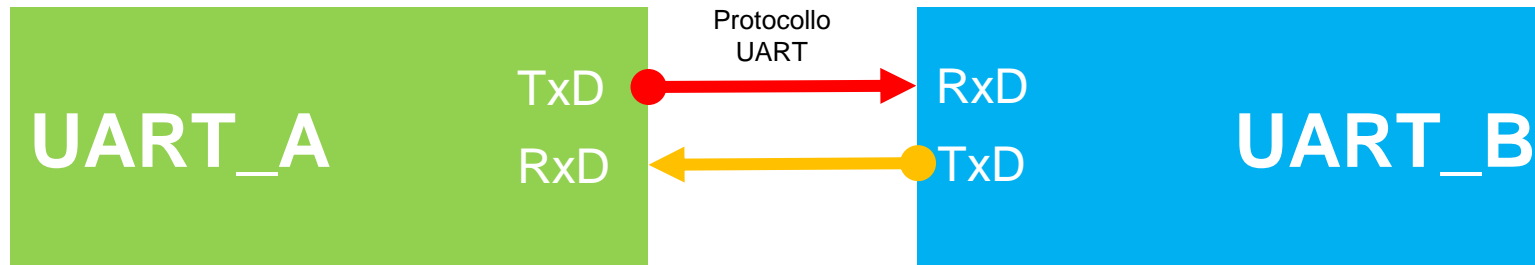
Interconnessione Nodo A – Nodo B



Interconnessione UART

Parti (1/5)

Per l'interconnessione dei due nodi si è deciso di usare il protocollo UART (Comunicazione seriale asincrona). Tale protocollo invia i bit di dati uno per uno, dal meno significativo al più significativo.

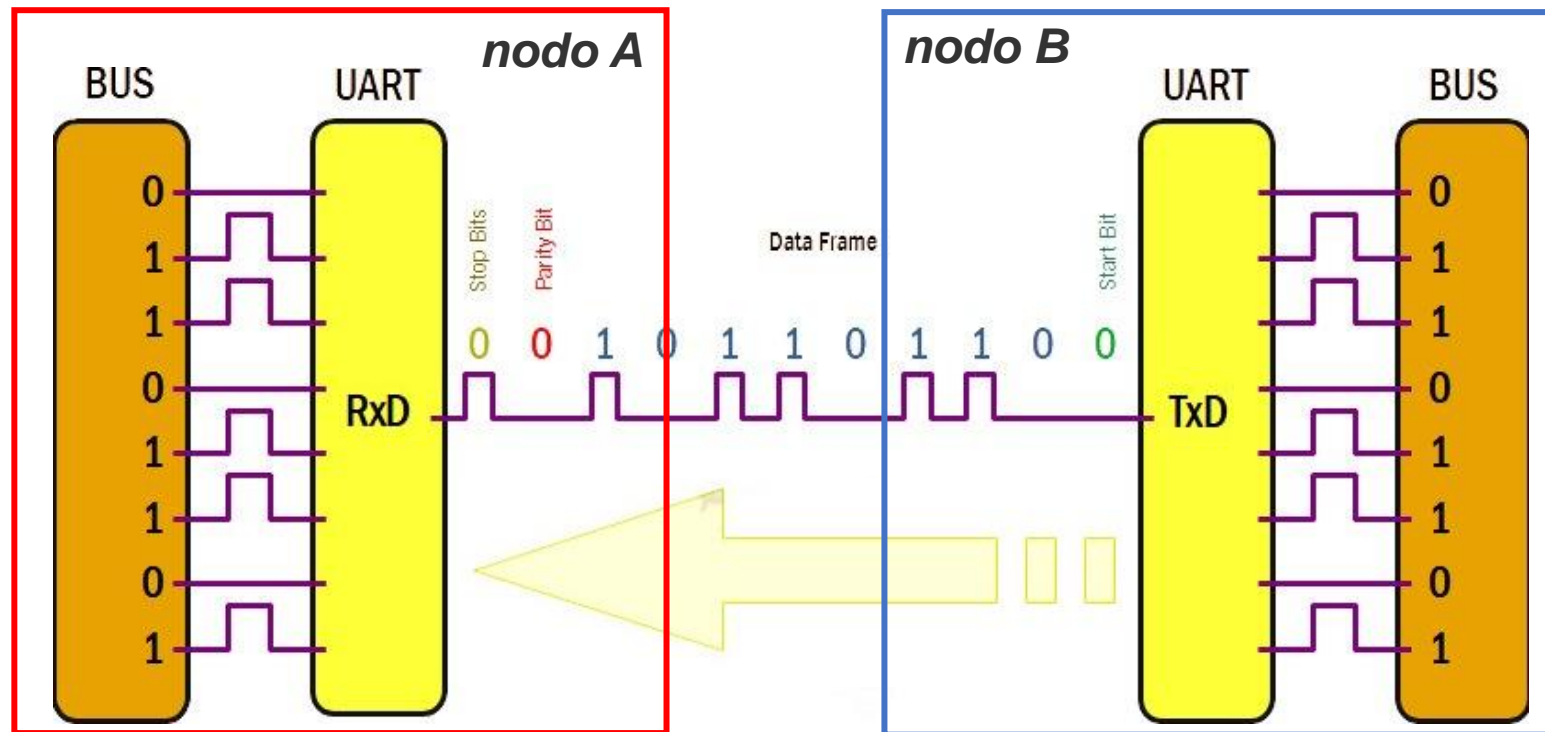


Si è scelta la modalità operativa **FULL-DUPLEX** (trasmissione dati in entrambe le direzioni simultaneamente)

Interconnessione UART

Parti (2/5)

Esempio di trasmissione dal *nodo B* al *nodo A* di 8 bit dato (011101101).



Interconnessione UART

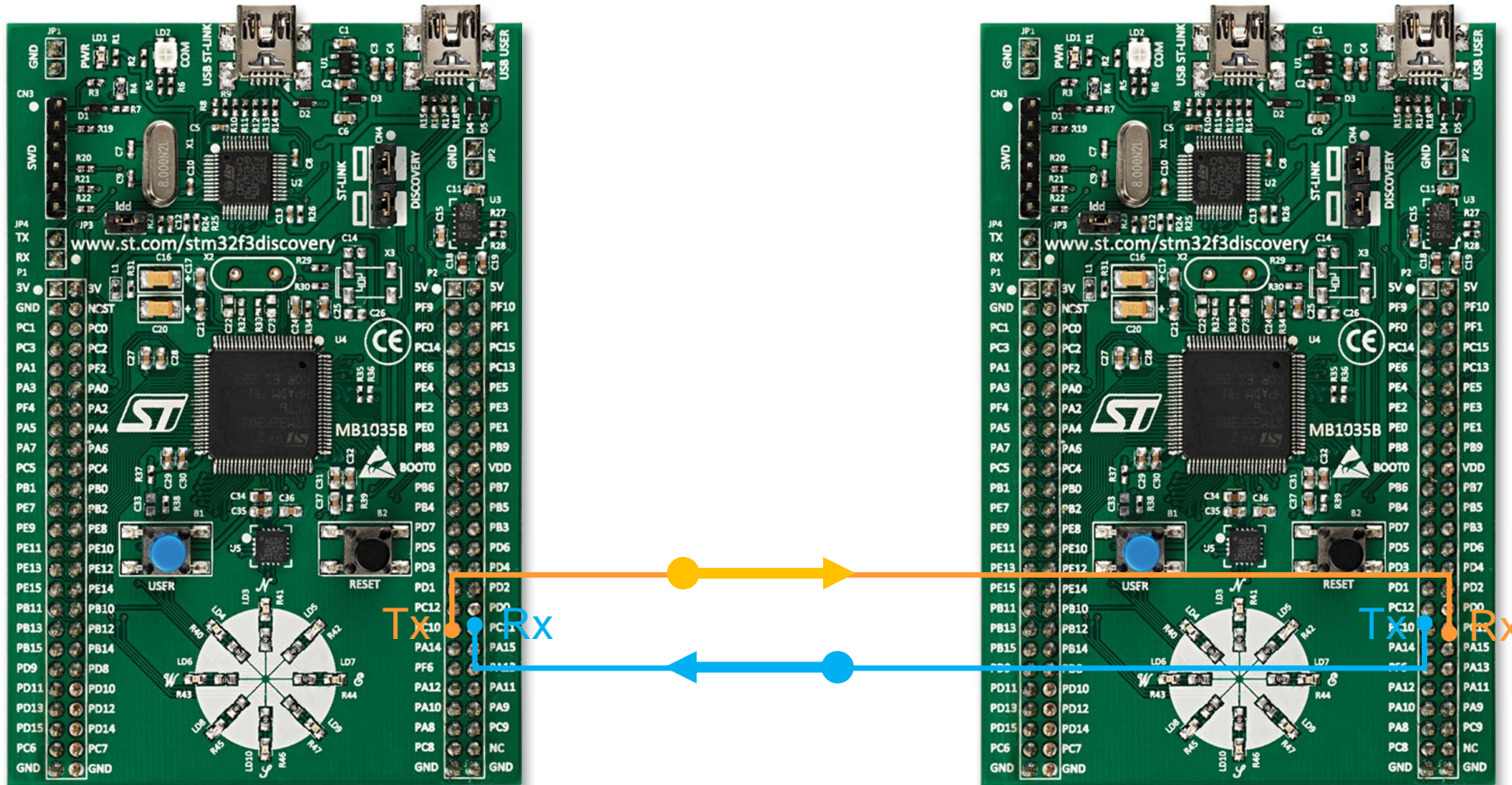
Parti (3/5)

Formato dei dati:



Interconnessione UART

Parti (4/5)



Implementazione UART (B) su STM32Cube

Parti (5/5)

Abbiamo configurato la UART4 impostando il Baud Rate (115200 – circa 8,7 us per ogni bit), il formato dei messaggi e abilitato le interruzioni in ricezione.

UART4 global interrupt / UART4 wake-up interrupt through EXTI line 34 ☒ 0

Al momento della ricezione di un carattere, la **Callback** della UART è invocata dalla relativa ISR e lo stato del nebulizzatore è modificato in base al segnale ricevuto.

Il codice relativo alle Callback, del Nodo A e del Nodo B, sono riportate nelle slide successive.

The screenshot shows the 'UART4 Mode and Configuration' window in STM32CubeMX. It is divided into two main sections: 'Mode' and 'Configuration'. In the 'Mode' section, the 'Mode' dropdown is set to 'Asynchronous'. The 'Configuration' section contains a 'Reset Configuration' button and a list of settings: 'DMA Settings', 'GPIO Settings', 'User Constants', and 'NVIC Settings', all of which are checked. Below these is a 'Parameter Settings' section, which is currently selected. This section contains a search bar and a list of parameters to be configured. The 'Basic Parameters' section includes 'Baud Rate' (115200 Bits/s), 'Word Length' (8 Bits (including Parity)), 'Parity' (None), and 'Stop Bits' (1). The 'Advanced Parameters' section includes 'Data Direction' (Receive and Transmit), 'Over Sampling' (16 Samples), and 'Single Sample' (Disable).

UART4 Mode and Configuration	
Mode	
Mode	Asynchronous
Configuration	
Reset Configuration	
<input checked="" type="checkbox"/> DMA Settings	<input checked="" type="checkbox"/> GPIO Settings
<input checked="" type="checkbox"/> User Constants	<input checked="" type="checkbox"/> NVIC Settings
<input checked="" type="checkbox"/> Parameter Settings	
Configure the below parameters :	
Search (Ctrl+F)	
Basic Parameters	
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable



Implementazione UART (Nodo A) su STM32Cube

Parti (5/5)

```
uint8_t receive_data_UART ( )
```

```
{
```

```
    HAL_UART_Receive_IT (&huart4, RX_BUFFER, BUFFER_LEN);
```

```
    return RX_BUFFER[0];
```

```
}
```

```
void HAL_UART_RxCpltCallback (UART_HandleTypeDef* huart)
```

```
{
```

```
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
```

```
    if (huart->Instance == huart4.Instance)
```

```
    {
```

```
        uint8_t data = receive_data_UART ( ); //da nodo B
```

```
        if (data == 1)
```

```
        {
```

```
            acqua = 1;
```

```
        } else {
```

```
            acqua = 0;
```

```
        }
```

```
    }
```

```
}
```



Implementazione UART (Nodo B) su STM32Cube

Parti (5/5)

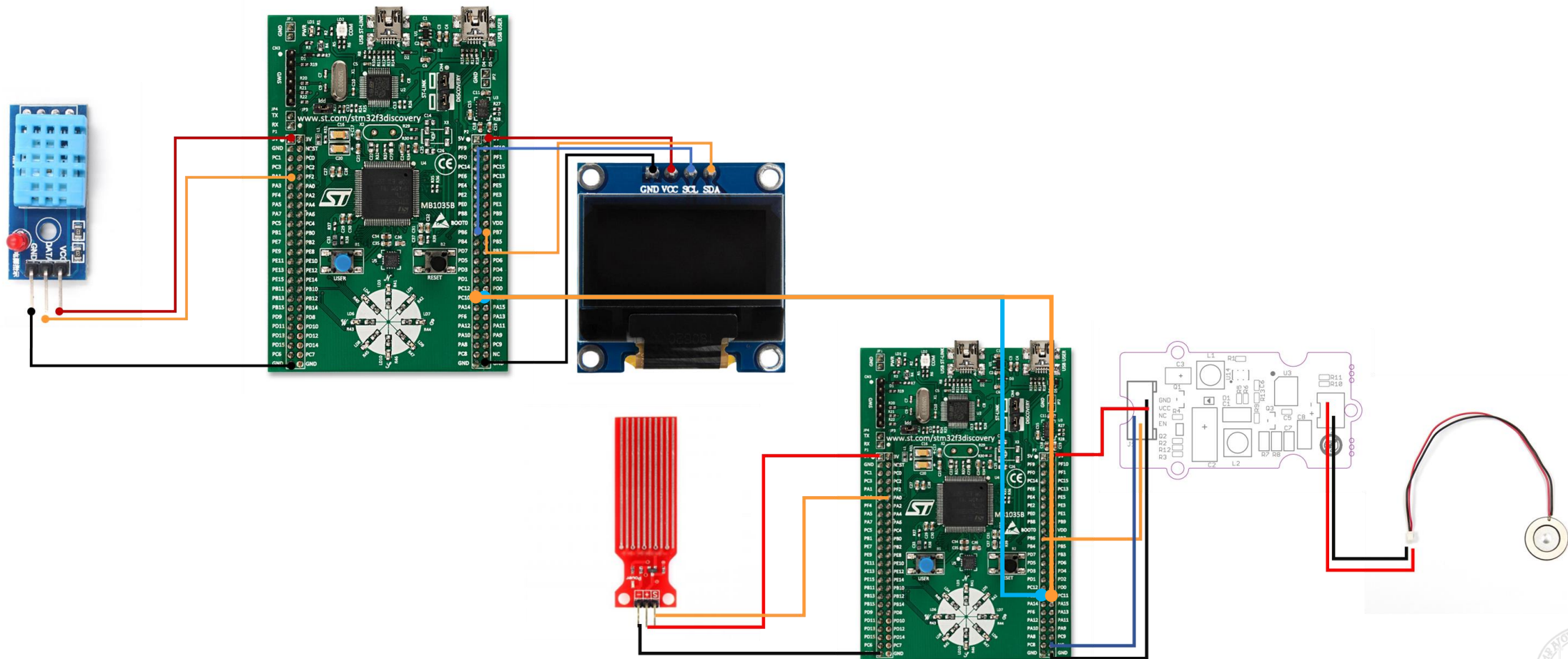
```
uint8_t receive_data_UART ( )
{
    HAL_UART_Receive_IT (&huart4, RX_BUFFER, BUFFER_LEN);
    return RX_BUFFER[0];
}

void HAL_UART_RxCpltCallback (UART_HandleTypeDef* huart)
{
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_9);
    if (huart->Instance == huart4.Instance)
    {
        uint8_t data = receive_data_UART ( ); //da nodo A
        if (data == 1)
        {
            neb_accesso = 1;
        } else {
            neb_accesso = 0;
        }
        gestisci_nebulizzatore(neb_accesso);
    }
}
```



Interconnessione Nodi Completa

Parti (1/2)

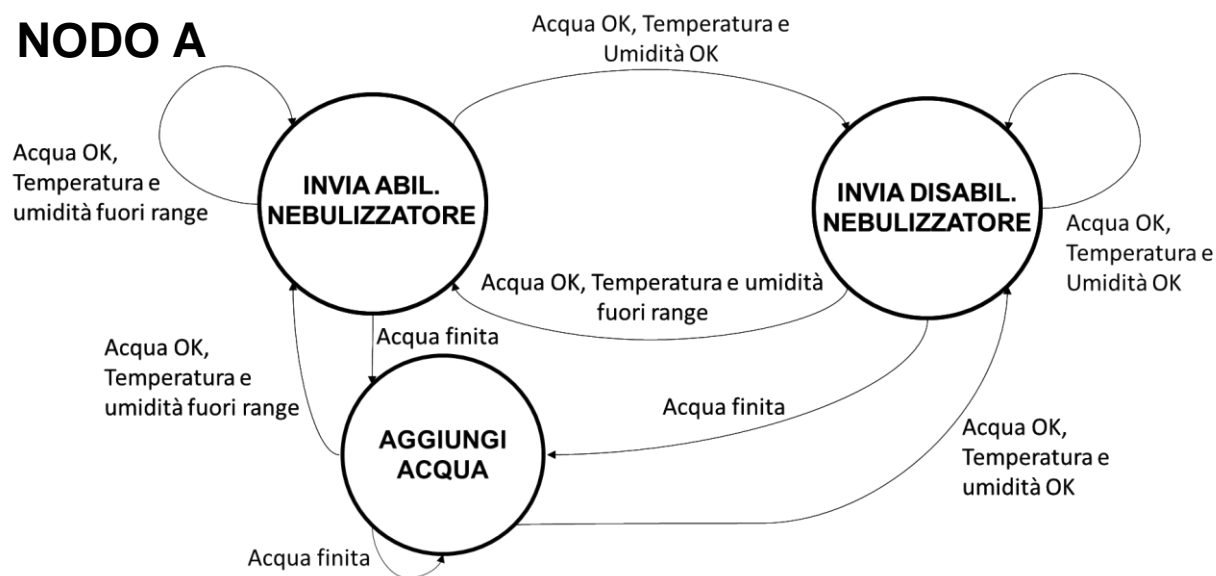


Automa Completo

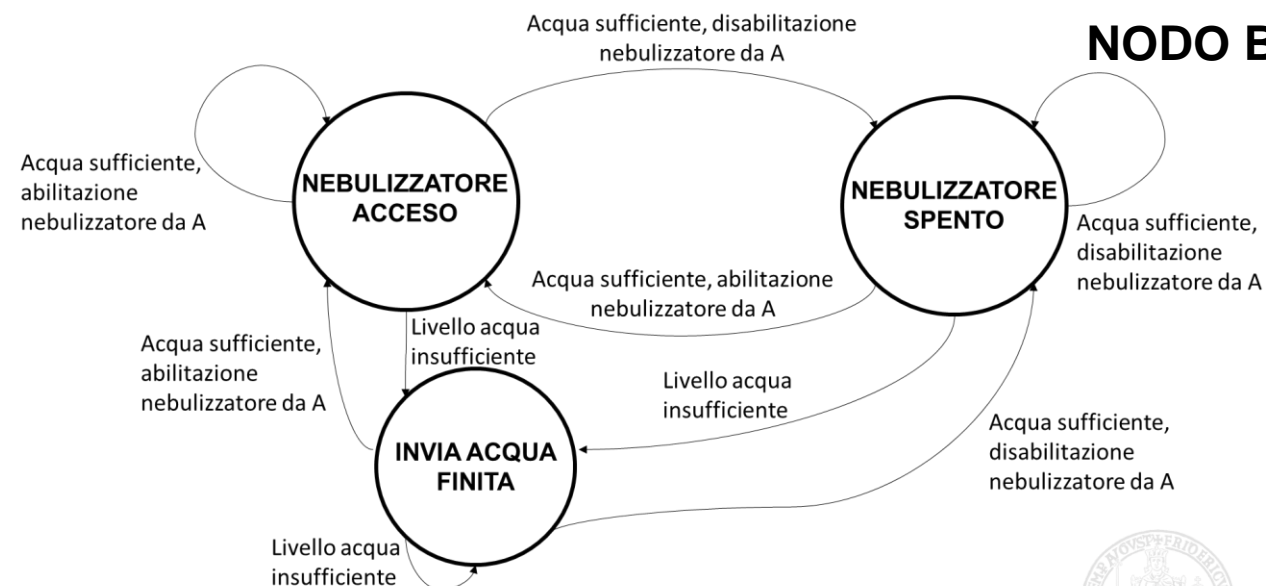
Parti (2/2)

É di seguito riportato l'automa a stati finiti **COMPLETO** di alto livello che descrive la dinamica del firmware di gestione del nebulizzatore.

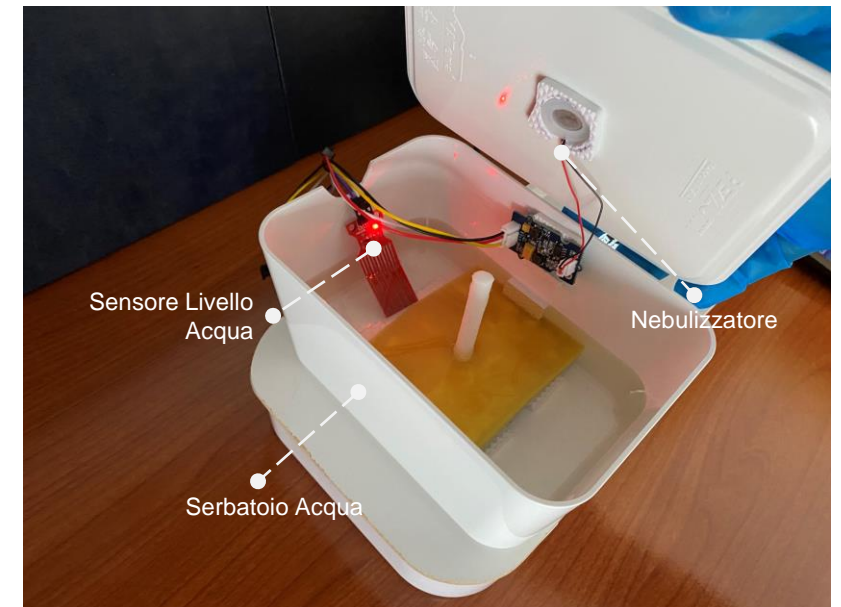
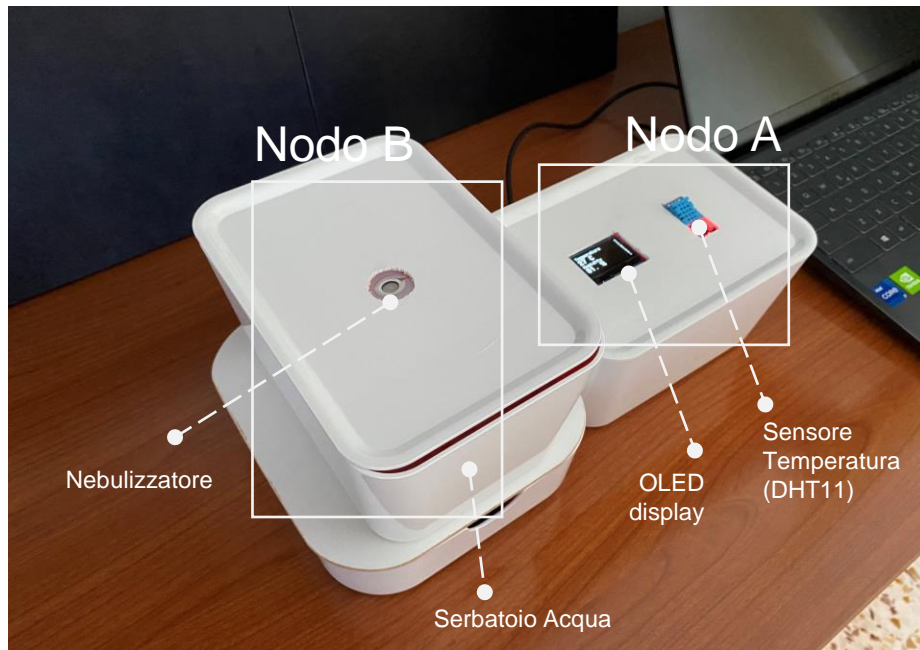
NODO A



NODO B



Overview del sistema «Air Humidifier»



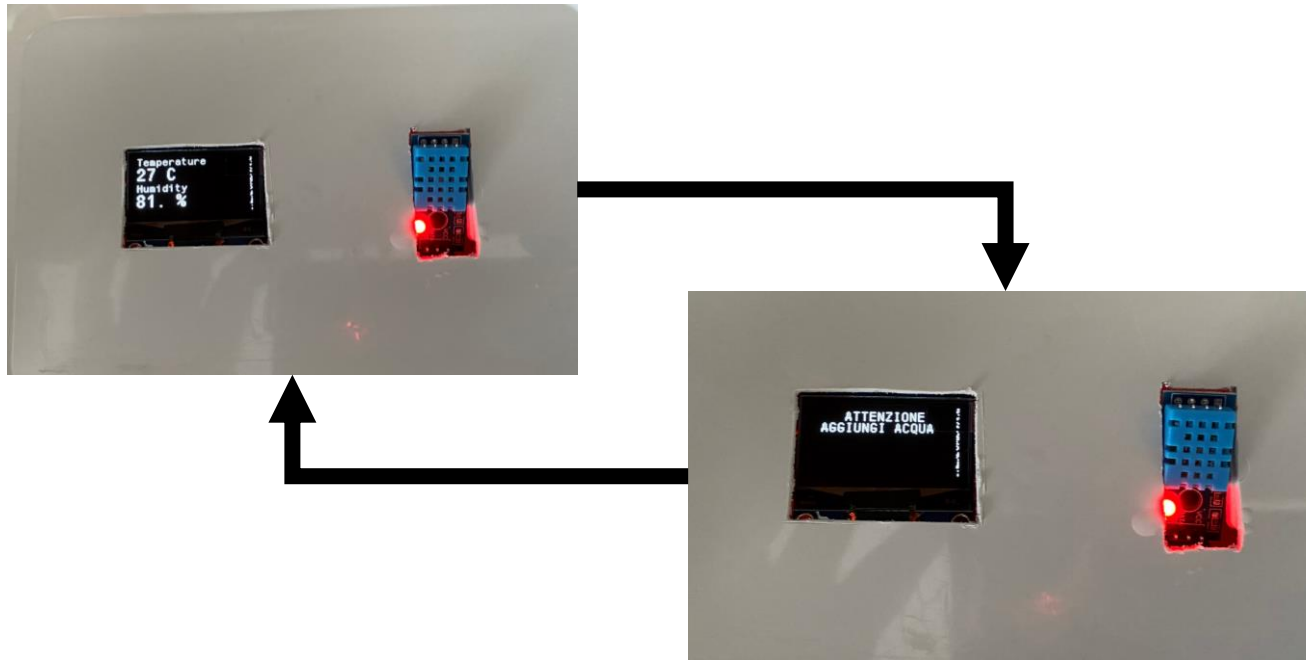
Video illustrativo

All'interno del video illustrativo di fianco, è possibile visualizzare una simulazione di funzionamento del nebulizzatore nel caso in cui la temperatura è maggiore di 23°C e l'umidità è minore del 55%, per ricreare tale situazione si è fatto uso di ghiaccio secco.



Video illustrativo

All'interno del video illustrativo di fianco, è possibile visualizzare una simulazione del caso in cui finisse l'acqua nel nebulizzatore e della sua seguente aggiunta.



Scopi futuri

- Interconnessione Nodo A – Nodo B via Bluetooth
- Nebulizzatore più potente
- Intelligenza Artificiale: gestione del range di temperatura e umidità in base alle caratteristiche della casa e del luogo dove è ubicata (Machine Learning)
- Controllo alimentazione B, se il nodo B è spento, il nodo A scriverà sul display che il nodo B è spento.
- Altro



Ringraziamenti

Grazie per l'attenzione



Per approfondimenti, si invita il lettore a visitare il repository:

https://github.com/LaErre9/CSD_Air_Humidifier

