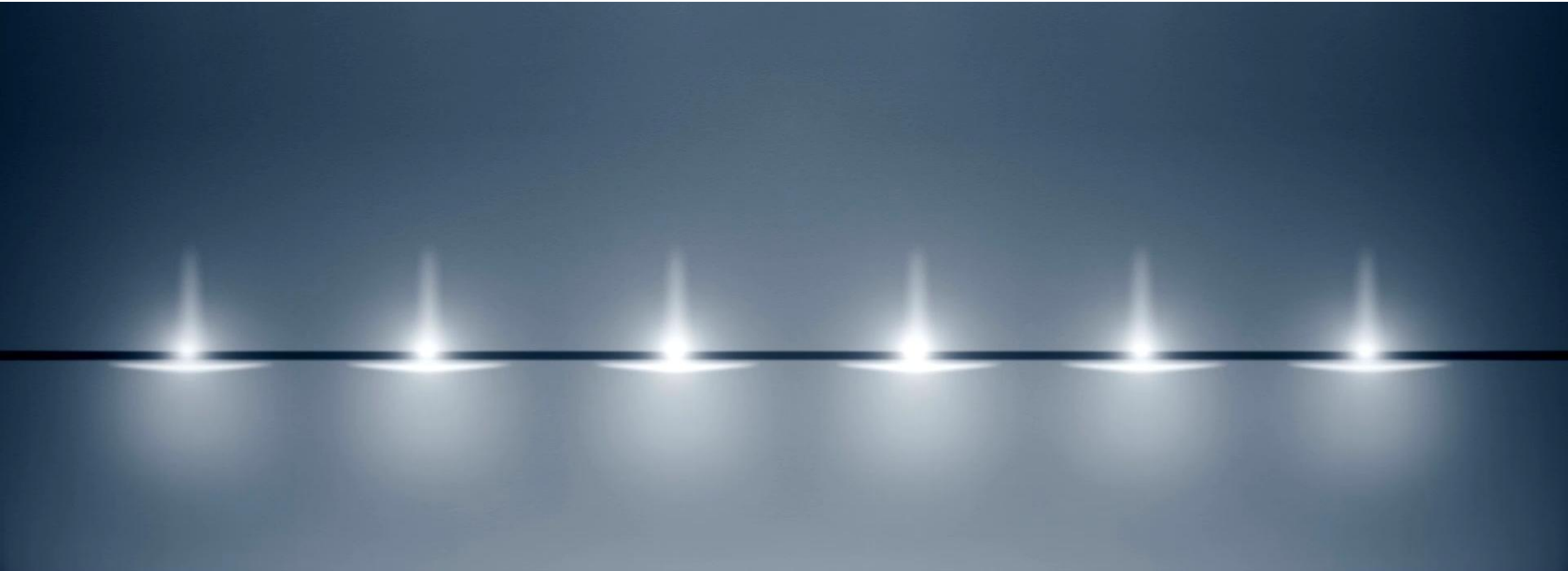
ARENA LETIZIA
FERRARA LEONARDO
RUSSO DAVIDE

NIGHTLIGHT

Anello LED ad Accensione Dinamica



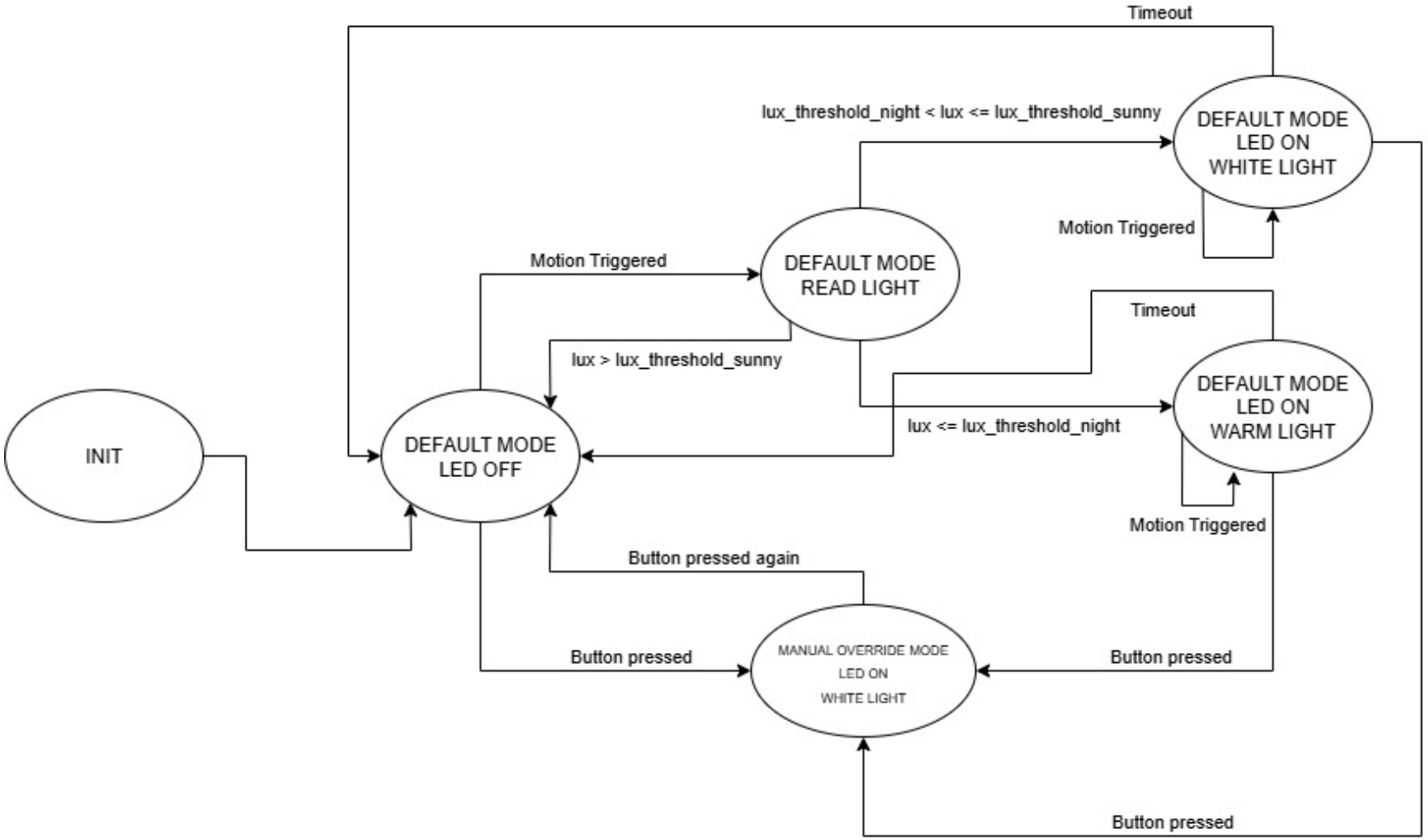
DESCRIZIONE FUNZIONALE

L’obiettivo è di creare un sistema di illuminazione installabile su di una parete o un comodino che può attivarsi automaticamente o manualmente, se lo si desidera.

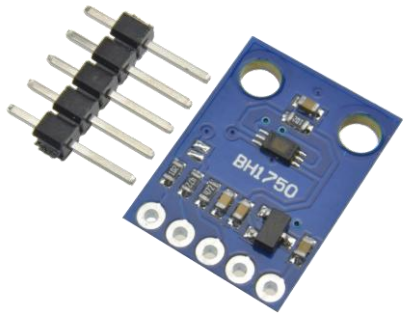
Nello specifico, il sistema opera in due modalità:

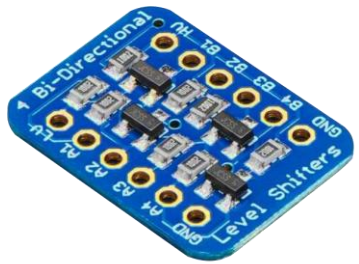
- **DEFAULT MODE**. La detezione di un movimento genera un impulso che segnala l’avvenuto movimento alla board; si procede ad acquisire il valore di luminosità ambientale e, se il valore è al di sotto di una soglia predeterminata, verrà acceso l’anello con un colore che dipende dal valore di luminosità acquisito.
- **MANUAL OVERRIDE MODE**. Se viene premuto un pulsante, la luce si accende indipendentemente dalle condizioni esterne.

AUTOMA A STATI FINITI



PANORAMICA DEI COMPONENTI





- Board STM32F303VCT6
- Breadboard
- Sensore di Movimento PIR HC-SR501
- Sensore di Luminosità BH1750
- Anello LED WS2812
- Shifter Logico 4CH BSS138

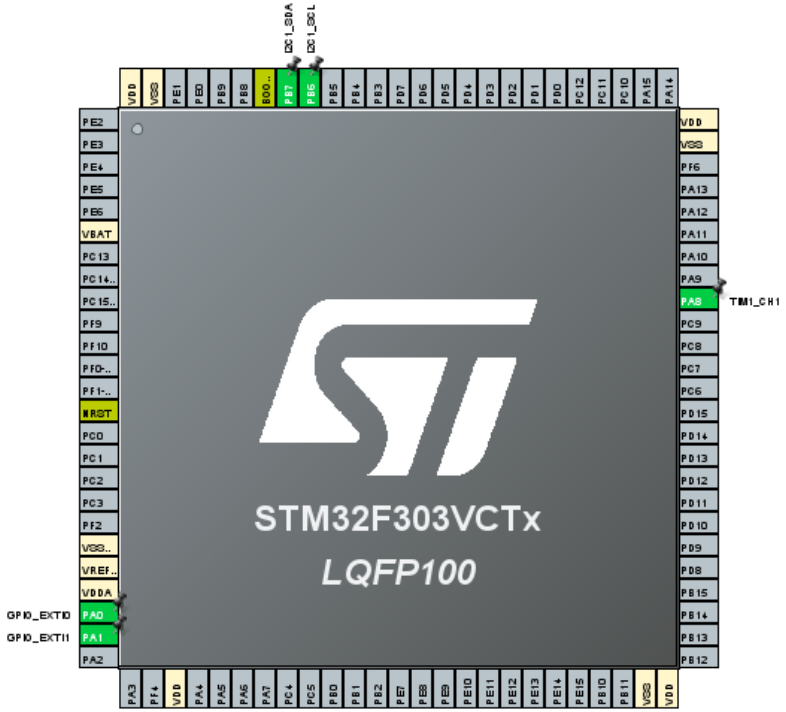
PANORAMICA DEI COMPONENTI (in dettaglio)

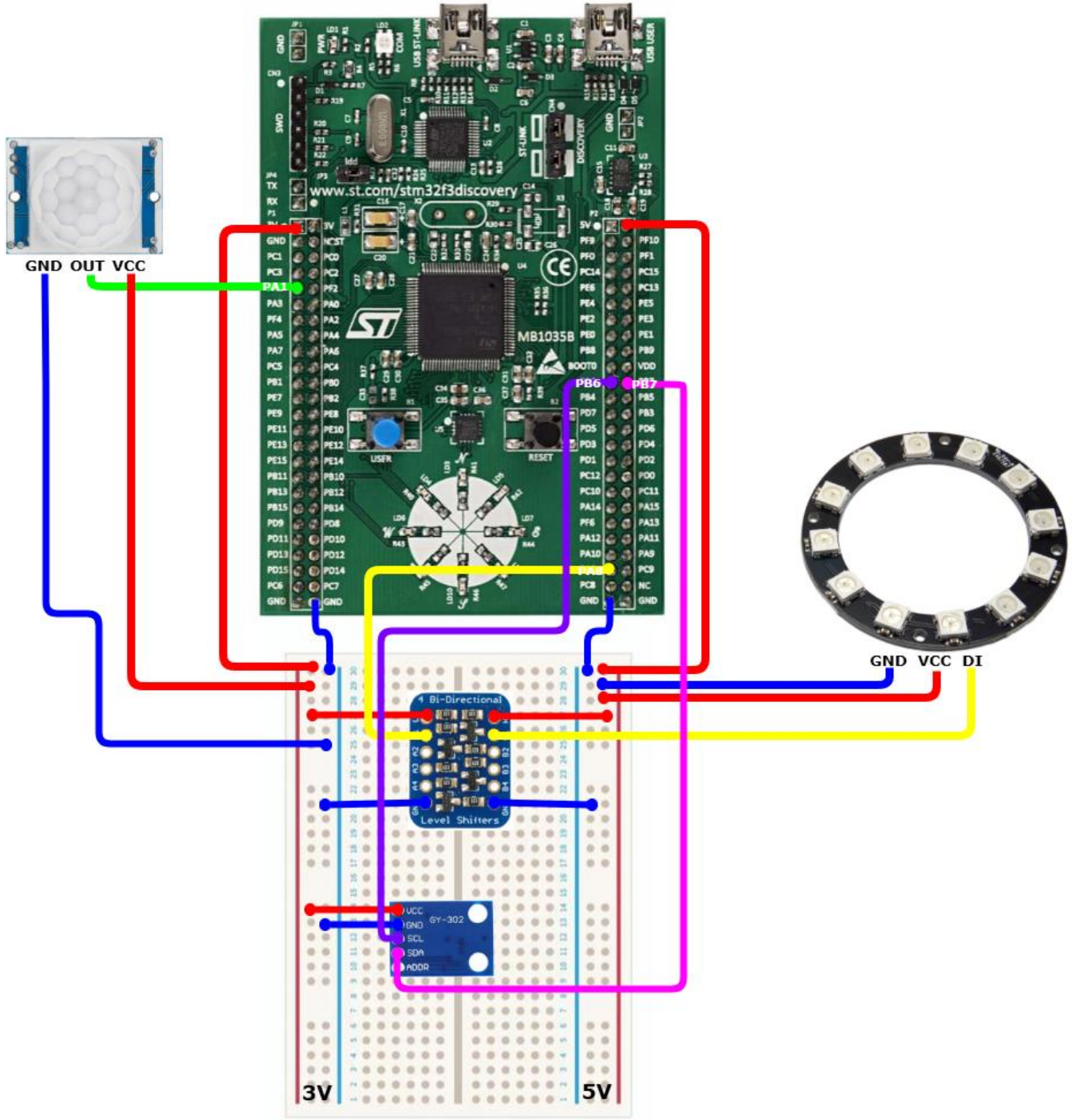
- **Sensore di movimento PIR HC-SR501**. È un sensore a infrarossi in grado di fornire un segnale alto a 3V quando rileva un corpo caldo in movimento. Ha due trimmer, uno dei quali regola la sensibilità, mentre l'altro regola i tempi di uscita; è inoltre dotato di un jumper per selezionare la modalità tra «single» e «multiple» trigger (nel nostro caso è stata utilizzata la seconda, e il segnale è alto finché viene rilevato movimento).
- **Sensore di luminosità BH1750**. Il sensore è in grado di trasformare l'intensità della luce, compresa in un range da 1 a 65.535 lux, in un segnale digitale disponibile tramite interfaccia I2C. Un lux equivale ad 1 lumen/m^2.

PANORAMICA DEI COMPONENTI (in dettaglio)

- **Anello LED WS2812**. È un anello circolare composto da 12 LED RGB indirizzabili individualmente. Ogni LED contiene tre diodi (rosso, verde, blu) controllabili in intensità per generare milioni di colori. È alimentato a 5V e il pin di ingresso deve essere alimentato allo stesso modo.
- **Shifter Logico 4CH BSS138**. È un circuito elettronico che consente di convertire segnali logici tra due tensioni diverse, nel nostro caso utilizzato per convertire la tensione del pin di ingresso all'anello LED da 3.3V a 5V.

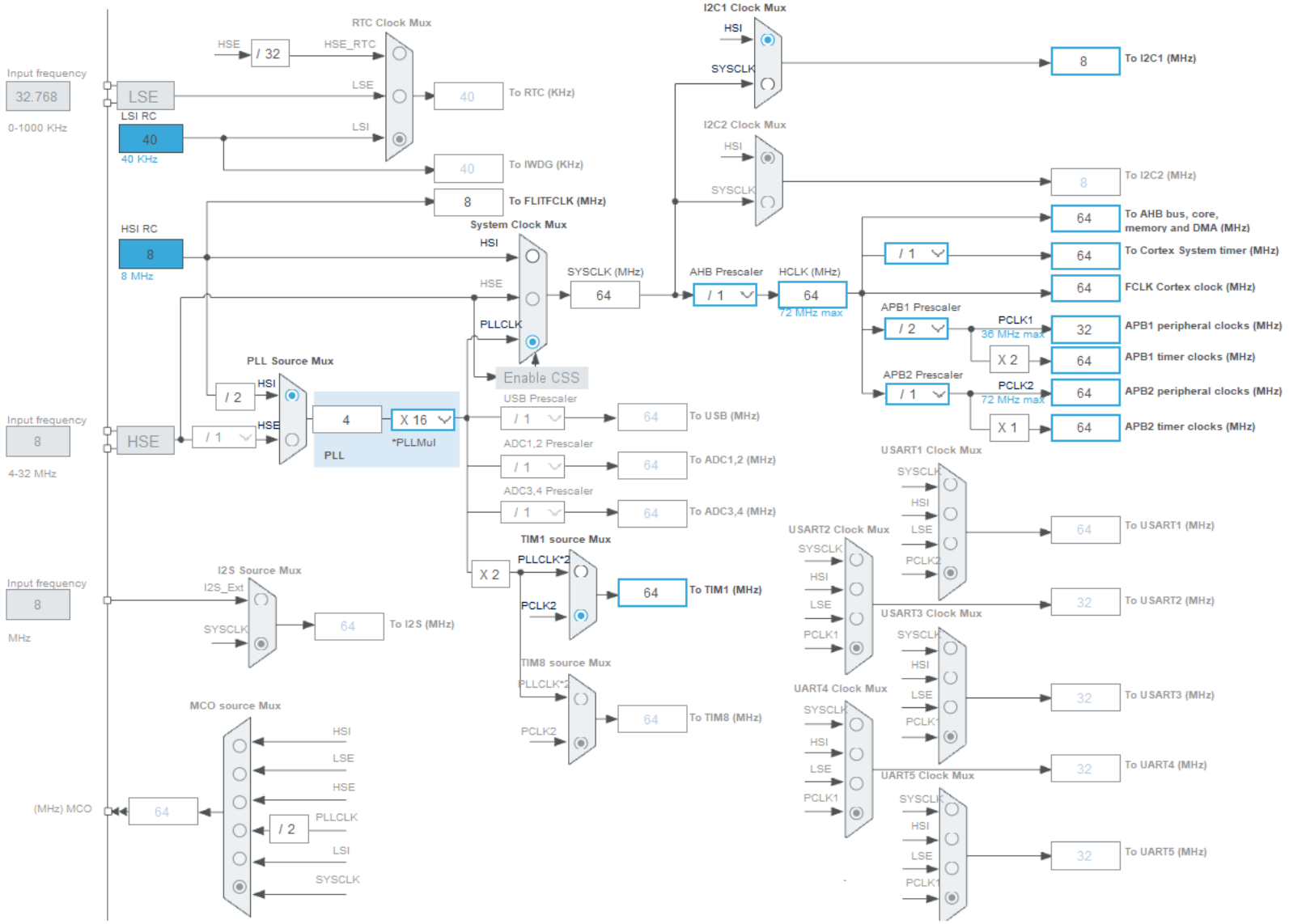
PINOUT DEL SISTEMA e CONFIGURAZIONE BOARD





CONFIGURAZIONE CLOCK

I LED WS2812 richiedono una tempistica molto precisa per distinguere i bit 0 e 1, al livello dei nanosecondi. Per ottenere questo, usiamo un segnale PWM generato da un timer hardware (TIM1), che è alimentato dal clock di sistema.
La frequenza PWM deve corrispondere al timing dei WS2812: Circa 800 kHz, cioè 1.25 µs per bit.
Il clock di sistema (HCLK) è impostato ad una frequenza sufficientemente alta per avere buona risoluzione nei tick del timer.
Il prescaler del TIM1 è a 0, quindi il timer lavora alla stessa velocità del clock.
Il valore ARR (Auto Reload Register) è calcolato per ottenere un periodo di 1.25 µs a 64 MHz, il che significa che ARR = 79.



CODICE ANELLO LED WS2812

```c
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim) {                    //Callback invocata quando tutti i dati sono inviati dal DMA (PWM-based) al timer
    if (htim->Instance == TIM1) {                                                    //Controlla se la callback è inviata dal timer giusto
        HAL_TIM_PWM_Stop_DMA(htim, TIM_CHANNEL_1);                                   //Libera il DMA prima che mandi altri segnali
        dma_done = true;
    }
}

void build_pwm_buffer(uint32_t *led_colors, uint16_t count) {                        //Traduce i valori di colore che mandiamo in corrispondenza di un movimento in un'onda PWM
    for (uint16_t i = 0; i < count; i++) {
        uint32_t color = led_colors[i];
        for (int bit = 23; bit >= 0; bit--) {
            if (color & (1 << bit)) {
                pwm_buffer[i * BITS_PER_LED + (23 - bit)] = PULSE_WIDTH_ONE;         //Impulso di lunga durata
            } else {
                pwm_buffer[i * BITS_PER_LED + (23 - bit)] = PULSE_WIDTH_ZERO;        //Impulso di breve durata
            }
        }
    }
}

void WS2812_Send(uint32_t *led_colors, uint16_t count) {
    build_pwm_buffer(led_colors, count);

    dma_done = 0;                                                                    //Il DMA va riavviato quando dobbiamo inviare un nuovo comando

    HAL_TIM_PWM_Stop_DMA(&htim1, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start_DMA(&htim1, TIM_CHANNEL_1, (uint32_t *)pwm_buffer, count * BITS_PER_LED);

    while (!dma_done);                                                               //Attendiamo che l'interruzione sia servita
    HAL_Delay(1);                                                                    //Attesa funzionale all'applicazione dei dati ricevuti da parte di un latch (impiega più di 50
                                                                                     //microsecondi)
}
```

CODICE ANELLO LED WS2812

Il codice riportato contiene le funzioni necessarie al funzionamento dell'anello LED WS2812. In particolare:

- `build_pwm_buffer`. Questa funzione acquisisce un array di valori di colore a 32 bit e converte ciascun bit di ogni colore in un pattern di temporizzazione da memorizzare in un buffer; il buffer verrà poi svuotato per avviare un trasferimento mediante DMA, che passerà l'ampiezza di ogni impulso al registro CCR1 del timer creando la giusta sequenza di durate lunghe/brevi per ogni bit di ogni LED.
- `WS2812_Send`. Questa funzione manda il frame all'anello usando l'onda PWM generata dal timer che era stato caricato precedentemente con i valori passati dal DMA.
- `HAL_TIM_PWM_PulseFinishedCallback`. Funzione invocata quando il DMA termina il trasferimento dell'onda PWM per pilotare i LED mediante timer.

CODICE SENSORE DI LUMINOSITÀ BH1750

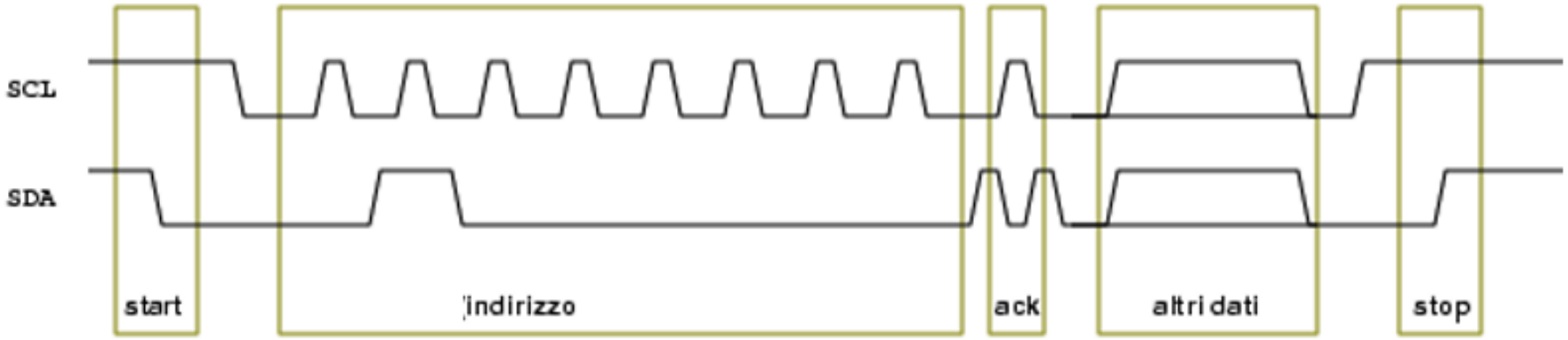
```c
HAL_StatusTypeDef BH1750_Init(I2C_HandleTypeDef *hi2c)
{
    uint8_t cmd = BH1750_CONT_HIRES_MODE;                    //Modalità di funzionamento: continuous (misura continuamente);
                                                             //Risoluzione: elevata; Precisione: 1 lux; Tempo di misura: 120 ms
    return HAL_I2C_Master_Transmit(hi2c, BH1750_I2C_ADDR, &cmd, 1, HAL_MAX_DELAY);    //Invia i dati in blocking mode (performa un polling)
}

HAL_StatusTypeDef BH1750_ReadLight(I2C_HandleTypeDef *hi2c, float *lux)
{
    uint8_t data[2];
    if (HAL_I2C_Master_Receive(hi2c, BH1750_I2C_ADDR, data, 2, HAL_MAX_DELAY) != HAL_OK)    //Riceve i dati dal bus se la trasmissione si è conclusa in maniera corretta
        return HAL_ERROR;

    uint16_t raw = (data[0] << 8) | data[1];                 //Giustapposizione dei due byte per formare un intero da 16 bit
    *lux = raw / 1.2;
    return HAL_OK;
}
```

Il codice riportato contiene le funzioni necessarie al funzionamento del sensore di luminosità BH1750. In particolare:

- `BH1750_ReadLight`. Questa funzione acquisisce il valore di intensità luminosa mediante un interfaccia I^2C e genera un valore a 16 bit che rappresenta la luminosità in lux.

PROTOCOLLO I2C

L'I2C è un protocollo sincrono (dispone di una linea di clock) che utilizza solo due linee di comunicazione (clock e dati). Nel caso in questione la comunicazione avviene tra un singolo master (il microcontrollore) ed un singolo slave (il sensore BH1750). Il master avvia la comunicazione con una condizione di start: abbassa la linea SDA mentre SCL è alto, indicando che sta per iniziare una trasmissione; successivamente invia 7 bit che rappresentano l'indirizzo dello slave che vuole contattare, ed un ultimo bit che indica l'azione da svolgere (0 se vuole scrivere 1 se vuole leggere). Se lo slave legge il proprio indirizzo invia un bit di ACK e da qui inizia l'effettivo trasferimento di dati: lo slave invia i dati ed il master risponde con un ACK ad ogni byte ricevuto. Infine, il master invia uno STOP alzando SDA mentre SCL è alto per indicare che la trasmissione si è conclusa. Si noti che le linee SDA e SCL sono open-drain, per cui i dispositivi possono solo metterle a 0.



CODICE MAIN

La logica del sistema può essere articolata in due parti:

- **Main Loop**. Il ciclo si occupa di controllare la modalità di esecuzione. Se siamo in DEFAULT MODE e la luminosità si trova all'interno di un range predeterminato, accendiamo le luci, dopodiché, se entro i prossimi 10 secondi non viene rilevato alcun movimento, queste si spegneranno, altrimenti verrà esteso il TIMEOUT. Se siamo in MANUAL OVERRIDE MODE, impostiamo l'accensione delle luci se pulsante è stato acceso per la prima volta, ignorando da quel momento in poi le rilevazioni del PIR, oppure torniamo alla modalità di default spegnendo le luci se il pulsante era già stato premuto

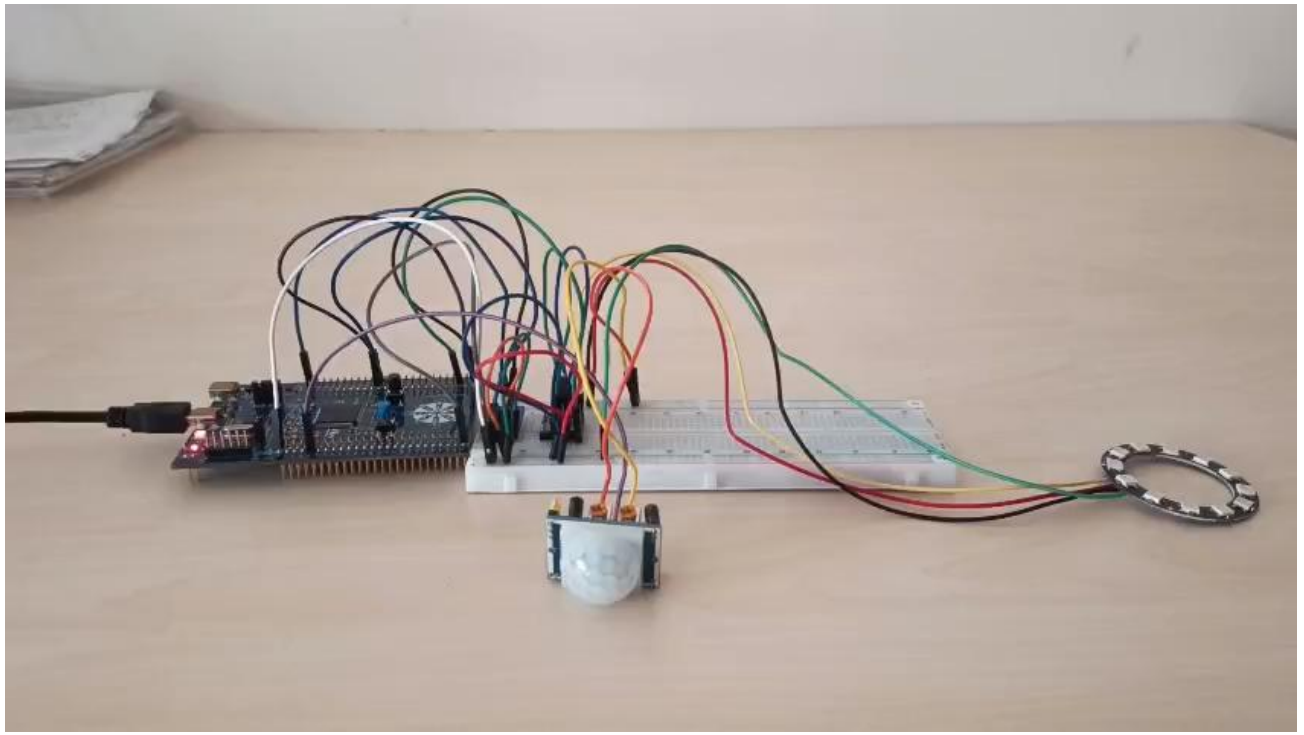
```
while (1)
        {

        if (override_led_update_needed)                                 //Se abbiamo premuto il Bottone...
        {
            while (!dma_done);                                          //...Aspettiamo che il DMA abbia ultimato il trasferimento...
            WS2812_Send(led_colors, LED_COUNT);                         //...E sovrascriviamo i LED
            override_led_update_needed = false;
        }
            if (current_mode == MODE_DEFAULT)                           //Se, invece, la modalità è quella di DEFAULT...
            {
                if (motion_triggered)                                   //...Ed è stato rilevato un movimento...
                {
                    motion_triggered = false;

                    if (!lights_on)                                     //...Se le luci non sono già accese...
                    {
                        if (BH1750_ReadLight(&hi2c1, &lux) == HAL_OK)   //...Se la lettura del valore di luminosità è avvenuta con successo...
                        {
                            if (lux_threshold_night < lux && lux <= lux_threshold_sunny)    //...Se la lettura è effettuata di giorno e i valori sono sotto-soglia
                            {
                                for (int i = 0; i < LED_COUNT; i++)
                                    led_colors[i] = 0x00FFFFFF;         //...Colora i LED di bianco
                            }
                            else if (lux < lux_threshold_night)         //Se invece siamo di notte e i valori sono sotto-soglia
                            {
                                for (int i = 0; i < LED_COUNT; i++)
                                    led_colors[i] = 0x00051619;         //Colora i LED di giallo
                            }

                            while (!dma_done);                          //Attendiamo il trasferimento
                            WS2812_Send(led_colors, LED_COUNT);         //Accendiamo i LED con il colore impostato
                            lights_on = true;
                            last_motion_time = HAL_GetTick();           //E cominciamo a contare per capire quando occorrerà un TIMEOUT
                        }
                    }
                }

                if (lights_on && HAL_GPIO_ReadPin(PIR_PORT, PIR_PIN) == GPIO_PIN_SET)   //Se un movimento viene effettuato con le luci già accese...
                {
                    last_motion_time = HAL_GetTick();                   //...Estendiamo il TIMEOUT
                }

                if (lights_on && (HAL_GetTick() - last_motion_time > MOTION_TIMEOUT))   //Se è occorso il TIMEOUT...
                {
                    for (int i = 0; i < LED_COUNT; i++)
                        led_colors[i] = 0x000000;                       //...Spegniamo i LED
                    WS2812_Send(led_colors, LED_COUNT);
                    lights_on = false;
                }
```

CODICE MAIN

- **Callback.** Viene invocata quando viene generata l'interruzione di uno dei due pin su cui sono mappati il pulsante ed il PIR. Identifica il dispositivo responsabile dell'interruzione. Se è il PIR ad essere responsabile dell'interruzione e il movimento si è verificato dopo un tempo di DEBOUNCE (200 ms), quest'ultimo verrà registrato. Se è il pulsante ad essere responsabile dell'interruzione, valutiamo la modalità operativa: se siamo in DEFAULT MODE, accendiamo le luci di bianco passando a MANUAL OVERRIDE MODE; se siamo in MANUAL OVERRIDE MODE torniamo a DEFAULT MODE spegnendo le luci.

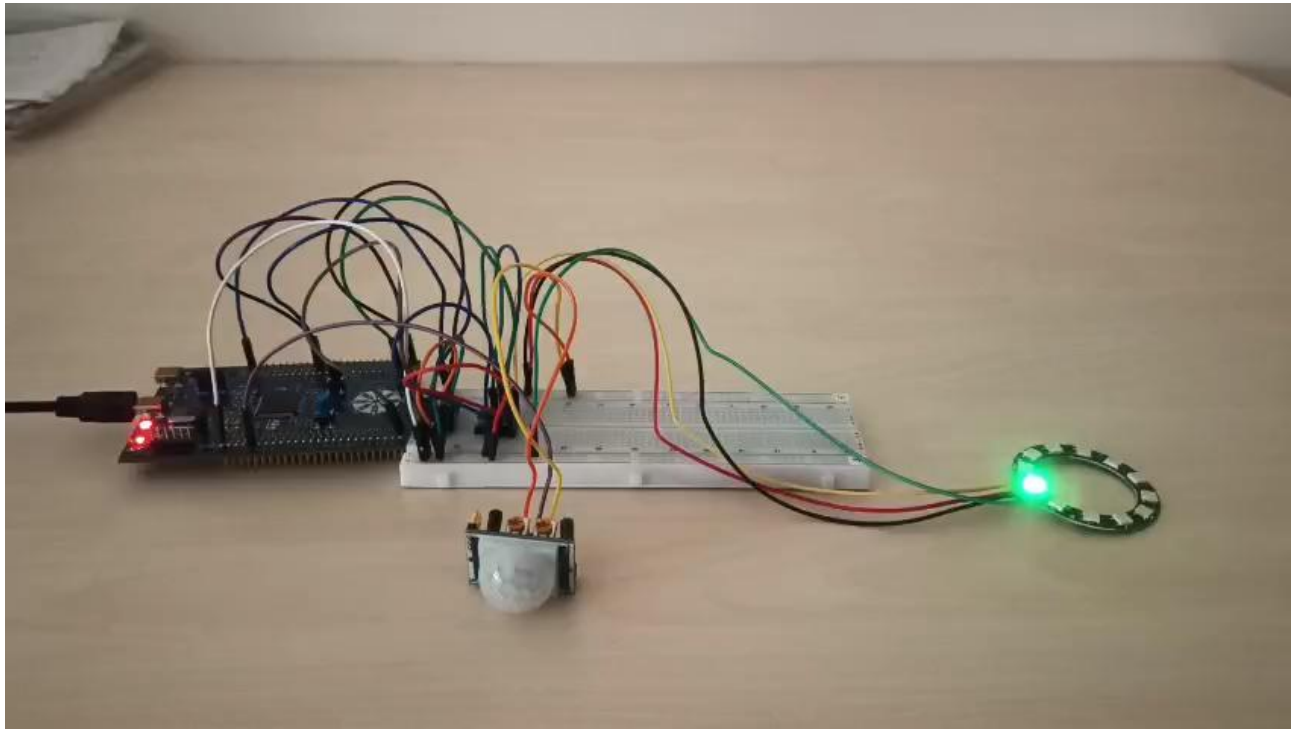
```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    static uint32_t last_pir_irq_time = 0;                                  //Il tempo trascorso dall'ultima interruzione del PIR
    static uint32_t last_button_irq_time = 0;                               //Il tempo trascorso dall'ultima interruzione del Bottone
    uint32_t now = HAL_GetTick();

    if (GPIO_Pin == PIR_PIN)                                                //Se il PIR ha attivato l'interruzione...
    {
        if ((now - last_pir_irq_time) > PIR_DEBOUNCE_TIME &&                //...Se il movimento si verifica dopo 200 ms...
            HAL_GPIO_ReadPin(PIR_PORT, PIR_PIN) == GPIO_PIN_SET &&          //...Se il PIR è acceso...
            current_mode == MODE_DEFAULT)                                   //...E se la modalità è quella di default
        {
            motion_triggered = true;                                        //...Segnaliamo che c'è stato un movimento
            last_pir_irq_time = now;
        }
    }
    else if (GPIO_Pin == BUTTON_PIN)                                        //Se, invece, è stato premuto il bottone...
    {
        if ((now - last_button_irq_time) > 250)                             //...Se sono passati più di 250 ms dall'ultima pressione...
        {
            if (current_mode == MODE_DEFAULT)                               //...Se siamo nella modalità di DEFAULT...
            {
                current_mode = MODE_MANUAL_OVERRIDE;                        //...Passiamo alla modalità di MANUAL OVERRIDE...
                lights_on = true;

                for (int i = 0; i < LED_COUNT; i++)
                    led_colors[i] = 0x00FFFFFF;                             //...E coloriamo le luci di bianco
            }
            else
            {
                current_mode = MODE_DEFAULT;                                //Se invece eravamo in modalità MANUAL OVERRIDE ritorniamo a quella di DEFAULT
                lights_on = false;

                for (int i = 0; i < LED_COUNT; i++)
                    led_colors[i] = 0x00000000;                             //E spegniamo le luci
            }

            override_led_update_needed = true;                              //Segnaliamo ad ogni modo che è necessario sovrascrivere i LED
            last_button_irq_time = now;
        }
    }
```

VIDEO DIMOSTRATIVO – CASO SOLEGGIATO



La luminosità della stanza è al di sopra della soglia, per cui i LED non si accendono al rilevamento di movimento ma solo se manualmente sollecitati mediante il bottone.

VIDEO DIMOSTRATIVO – CASO PENOMBRA



La stanza ha luminosità moderata, per cui i LED si accendono sia al rilevamento di movimento sia se manualmente sollecitati mediante il bottone. La luce è bianca ed intensa.

VIDEO DIMOSTRATIVO – CASO BUIO

La stanza è al buio, per cui i LED si accendono sia al rilevamento di movimento sia se manualmente sollecitati mediante il bottone. La luce è calda e meno intensa per non risultare eccessivamente abbagliante.