Sviluppo di un sistema di monitoraggio di LED per prove di vita accelerata

Partecipanti:

Daniele Angioni 70/83/65230 Giacomo Gallus 70/83/65234

Roberto Ruda 70/83/65240 Augusto Mura 70/83/65228

Presentazione Progetto

Sistemi a Microcontrollore A.A. 2019/2020

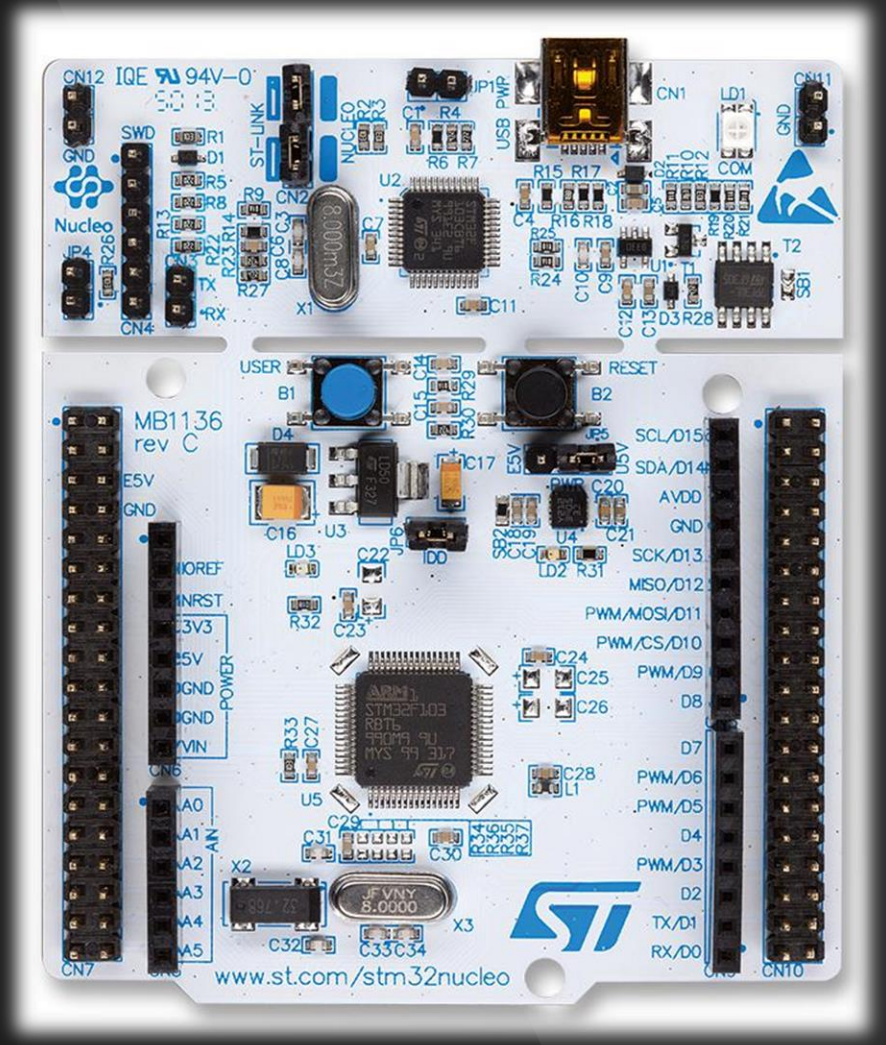
Specifiche del progetto

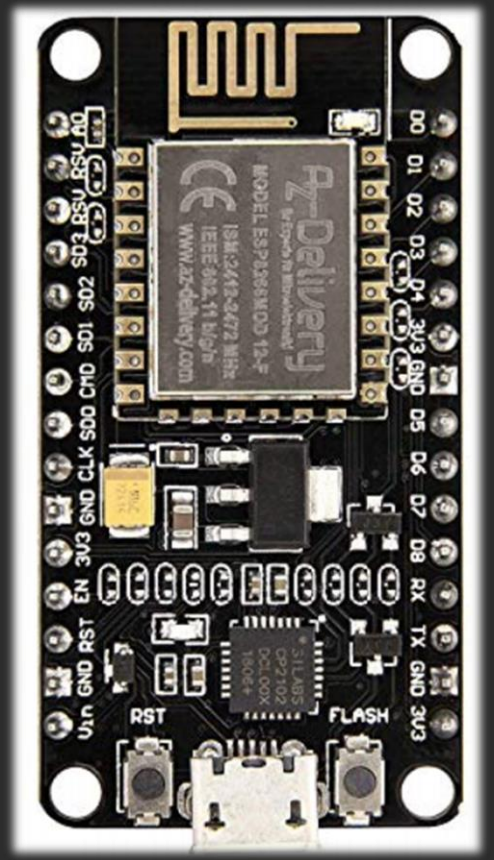
- Monitoraggio LED:
  - 3 test in corrente continua off-nominal a livelli di corrente diversi
  - Modo di guasto: spegnimento totale o -30% potenza ottica emessa
  - Monitoraggio con microcontrollore STM32 della potenza ottica in modalità discontinua (prove a tempi troncati) di 8 LED per prova
  - Potenza ottica rilevata tramite fotoresistenze
  - Alert via Wi-Fi di guasti e stato del sistema

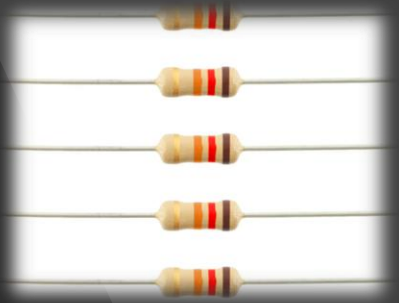
Hardware utilizzato

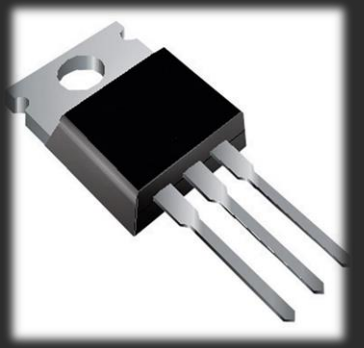
- STM32 NUCLEO-F030R8:
  scheda utilizzata per effettuare il controllo sui guasti e per mandare i segnali di allerta al modulo WiFi
- NODEMCU ESP8266:
  microcontrollore con modulo WiFi che riceve i segnali dalla scheda di controllo e li trasmette via Internet tramite un bot Telegram
- 8 LED C535A-WJN
- 8 resistenze da 12 kΩ
- 8 fotoresistenze (10 Ω – 100 kΩ)
- MOSFET di potenza (NMOS)

Hardware utilizzato

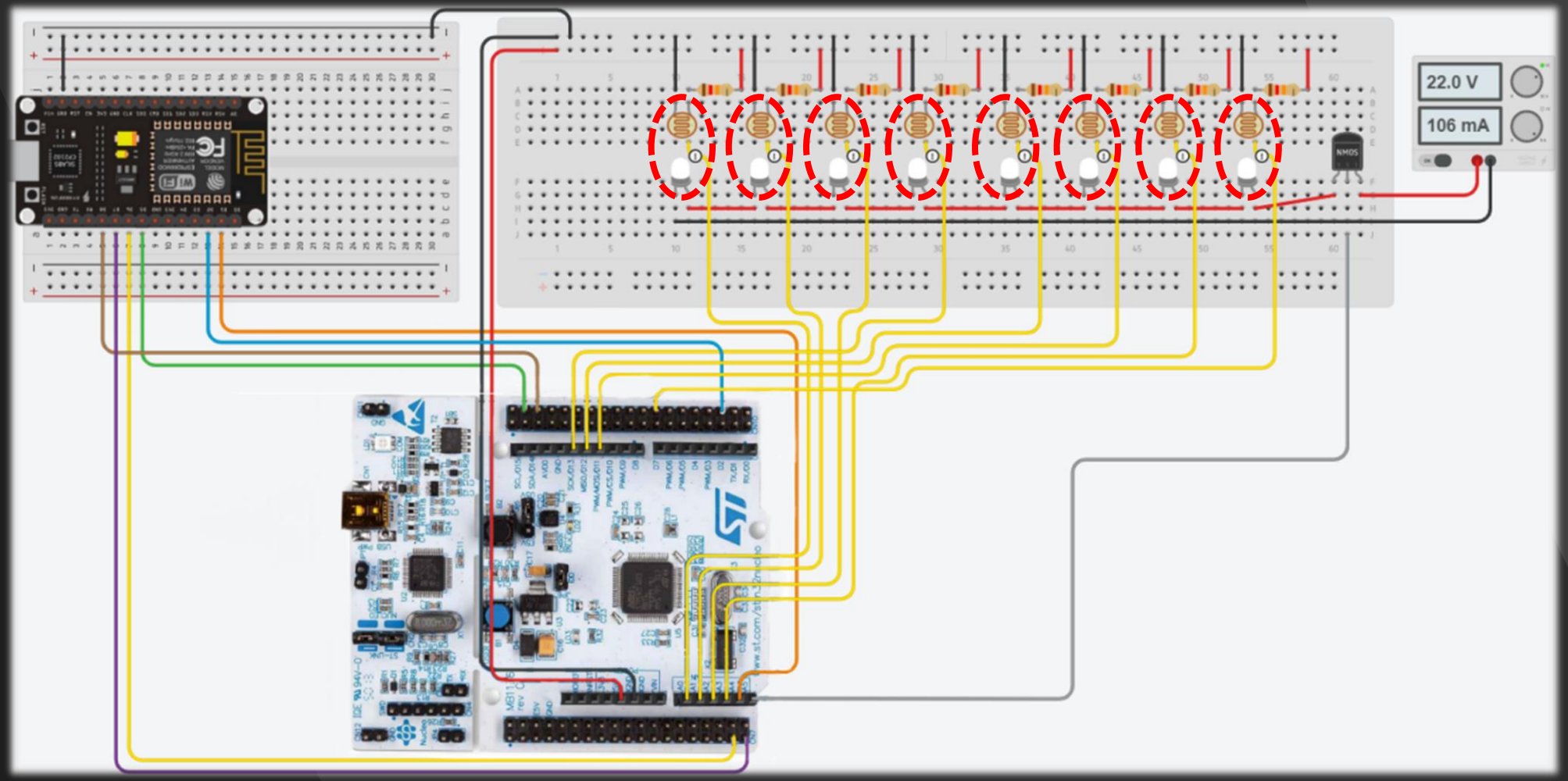




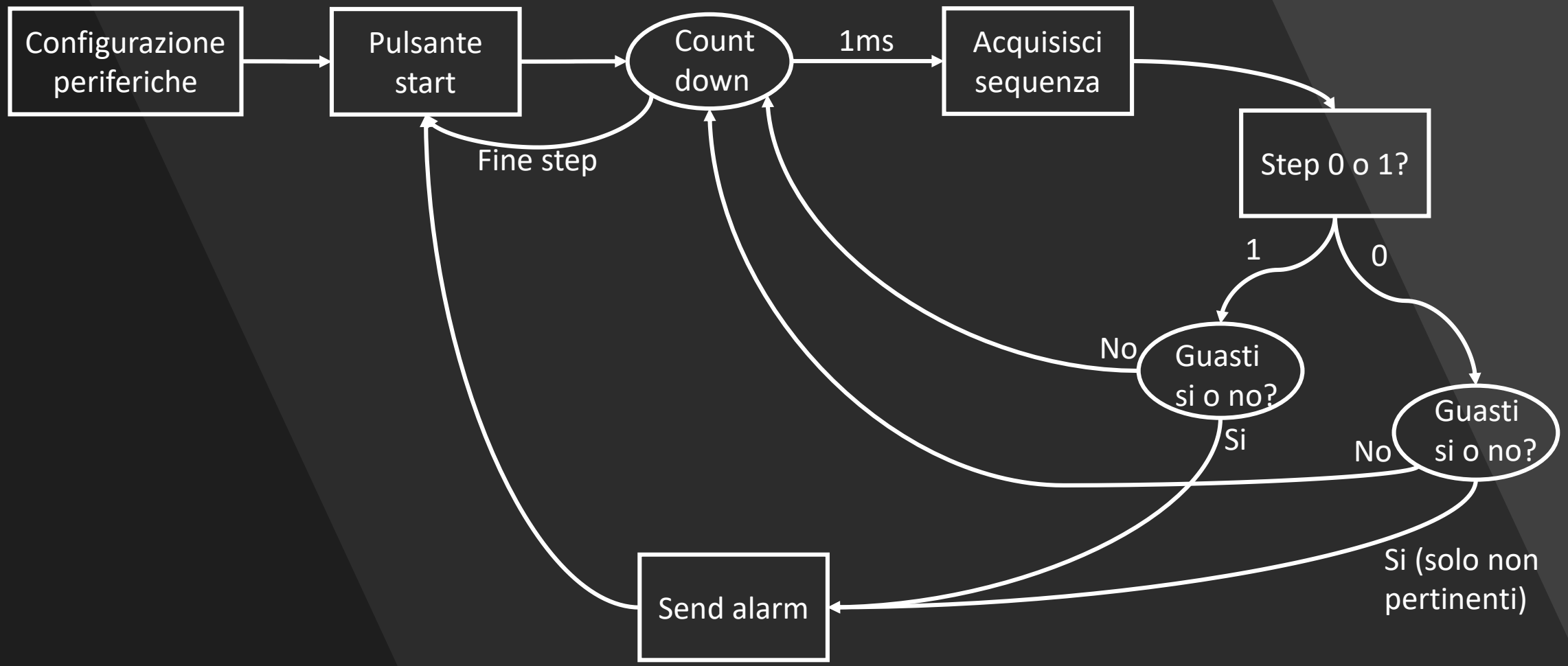




Setup del circuito



Schema logico



Codice STM32: configurazione GPIO

```
void configuraGPIO(void){
    //CONFIGURAZIONE PORTE OUTPUT PER MANDARE I DATI
    //1 transistor per comandare il passaggio di corrente a monte
    //6 pin per mandare il messaggio al modulo Wi-Fi
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

    //CONFIGURO pin C0 IN OUTPUT PER COMANDARE IL TRANSISTOR CHE COMANDA LA CORRENTE
    //IL RESTO DEI PIN DI C SCELTI COME OUTPUT PER INVIARE IL MESSAGGIO AL MODULO WI-FI
    //pin C7 è usato come interrupt esterno
    GPIOC->MODER |= GPIO_MODER_MODER0_0 |  GPIO_MODER_MODER1_0 | GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 | GPIO_MODER_MODER4_0 |
            GPIO_MODER_MODER5_0 | GPIO_MODER_MODER6_0; // !GPIO_MODER_MODER7_0; questo è già  a 0, quindi in input
    GPIOC->ODR &= !(0xFFFF);
    GPIOC->PUPDR |= GPIO_PUPDR_PUPDR7_1;//resistenza di pull down per portare il pin flottante a 0 quando interruttore aperto

    //CONFIGURAZIONE PORTE INPUT PER ACQUISIRE I SEGNALI NON è NECESSARIA
    //POICHE' L'ADC è GIà  COLLEGATO AI PIN DI DEFAULT

    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[1] |= 0x2 << SYSCFG_EXTICR2_EXTI7_Pos;//abilito porta C per interrupt
    EXTI->IMR |= EXTI_IMR_IM7;
    EXTI->RTSR |= EXTI_RTSR_RT7; //valore di default
    NVIC_EnableIRQ(EXTI4_15_IRQn);
    NVIC_SetPriority(EXTI4_15_IRQn, 1);
}
```

Codice STM32: configurazione timer

```
void configuraTIM14(void){
    RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;//accendo il timer
    TIM14->PSC = 7;//feq=1 Mhz Teq=1us
    TIM14->ARR = T_SAMPLE - 1;
    //il timer e il relativo interrupt viene fatto partire con l'interrupt dell'EXTI
    NVIC_EnableIRQ(TIM14_IRQn);
    NVIC_SetPriority(TIM14_IRQn, 2);//interrupt meno prioritario
}

void configuraTIM15(void){
    RCC->APB2ENR |= RCC_APB2ENR_TIM15EN;//accendo il timer
    TIM15->PSC = 799;//feq=10 Khz Teq=0.1ms
    TIM15->ARR = T_SAMPLE_WAIT - 1;// conta 10000 volte 0.1 ms ovvero 1 sec
    //il timer e il relativo interrupt viene fatto partire con l'interrupt dell'EXTI
    NVIC_EnableIRQ(TIM15_IRQn);
    NVIC_SetPriority(TIM15_IRQn, 0);//interrupt più prioritario
}
```

Codice STM32: configurazione ADC

```
void configuraADC1(void){
    //CALIBRAZIONE
    RCC->APB2ENR |= RCC_APB2ENR_ADCEN;
    if ((ADC1->CR & ADC_CR_ADEN) != 0){
        ADC1->CR |= ADC_CR_ADDIS;
    }
    while ((ADC1->CR & ADC_CR_ADEN) != 0);
    ADC1->CR |= ADC_CR_ADCAL;
    while ((ADC1->CR & ADC_CR_ADCAL) != 0);

    //ABILITAZIONE
    if ((ADC1->ISR & ADC_ISR_ADRDY) != 0){
        ADC1->ISR |= ADC_ISR_ADRDY;
    }
    ADC1->CR |= ADC_CR_ADEN;
    while ((ADC1->ISR & ADC_ISR_ADRDY) == 0);

    //CONFIGURAZIONE
    ADC1->CFGR2 &= ~ADC_CFGR2_CKMODE;//clock asincrono

    //Abbiamo attivato solo 2 canali per prova
    ADC1->CHSELR |= ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL1; /*| ADC_CHSELR_CHSEL4 |
            ADC_CHSELR_CHSEL5 | ADC_CHSELR_CHSEL6 | ADC_CHSELR_CHSEL7 | ADC_CHSELR_CHSEL8 |ADC_CHSELR_CHSEL9;*/

    ADC1->SMPR &= !ADC_SMPR_SMP;//frequenza massima di sampling

    ADC1->CFGR1 &= !(ADC_CFGR1_CONT);
    ADC1->CFGR1 |= ADC_CFGR1_DISCEN;
    //risoluzione 12 bit (default), modalità single mode

}
```

Codice STM32: handler interrupt esterno (bottone)

```
void EXTI4_15_IRQHandler(void){
    if(state == 0){
        if(cnt >= (T_STEP0-1)){
            state = 1;
            cnt=0;
        }
    }
    else {
        if(cnt >= T_STEP1-1){
            state = 0;
            cnt=0;
        }
    }

    while ((GPIOC->IDR & GPIO_IDR_7) != 0); /*Finché manteniamo il pulsante premuto la prova
    non parte (soluzione per problemi di caduta di tensione sui led che si verifica solamente
    durante la chiusura del circuito del bottone)*/

    //per aumentare la robustezza aspettiamo un altro seocondo prima di riprendere o iniziare la prova
    TIM15->CR1 |= TIM_CR1_CEN;
    TIM15->DIER |= TIM_DIER_UIE;

    hold_on_signal = 1;
    while (hold_on_signal != 0);

    //abilito il timer
    TIM14->CR1 |= TIM_CR1_CEN;
    TIM14->DIER |= TIM_DIER_UIE;

    //riattacca l'interruttore generale
    GPIOC->ODR |= 0x1;

    EXTI->PR |= EXTI_PR_PIF7;
}
```

Codice STM32: acquisizione sequenza canali ADC

```
void acquisisciSequenza(void){
    int i = 0;
    //Acquisisce tutti i canali

    for (i=0; i < N_CH; i++)
    {
        ADC1->CR |= ADC_CR_ADSTART; /* Start the ADC conversion */
        while ((ADC1->ISR & ADC_ISR_EOC) == 0) /* Wait end of conversion */
        {
            /* For robust implementation, add here time-out management */
        }
        dati[i] = ADC1->DR; /* Store the ADC conversion result */
    }
}
```

Codice STM32: controllo LED durante lo step 0

```
void TIM14_IRQHandler(void){
    int j = 0;

    acquisisciSequenza();//aggiornamento dati[]

    if(state == 0){
        if(cnt < T_STEP0-1){//conta 4 ore
            cnt++;

            for(j=0; j < N_CH; j++){
                if(guasti[j] == 0){//se entra vuol dire che nel LED j non si è ancora verificato un guasto, altrimenti passa al led successivo
                    if(dati[j] > SOGLIA_0){
                        //controllo se ho un guasto non pertinente
                        cnt_soprasoglia[j]++;
                        if(cnt_soprasoglia[j] > MAX_SOPRASOGLIA){
                            //guasto non pertinente
                            guasti[j] = 2;
                            sendAlarm(j, 1);
                        }
                    }
                }
            }
        }
        else{
            //mando l'avviso
            //stacco l'interruttore generale
            sendAlarm(0, 2);//scrivo al posto dell'indice del LED lo step che si è appena concluso

            //disabilito il timer
            TIM14->CR1 &= !TIM_CR1_CEN;
            TIM14->DIER &= !TIM_DIER_UIE;
            cnt = 0;
        }
    }
```

Codice STM32: controllo LED durante lo step 1

```
else{
    if(cnt < T_STEP1-1){//conto 2 secondi
        cnt++;
        for(j=0; j < N_CH; j++){
            if(guasti[j] == 0){//se entra vuol dire che nel LED j non si è ancora verificato un guasto, altrimenti passa al led successivo
                if(dati[j] > SOGLIA_1){
                    //controllo se ho un guasto pertinente
                    cnt_soprasoglia[j]++;
                    if(cnt_soprasoglia[j] > MAX_SOPRASOGLIA){
                        //guasto pertinente
                        guasti[j] = 1;
                        sendAlarm(j, 0);
                    }
                }else{
                    //entra SOLO SE all'interrupt precedente era sopra la soglia
                    //Tutto ciò funziona nell'ipotesi che le oscillazioni siano visibili a occhio nudo (circa 60Hz)
                    if(cnt_soprasoglia[j] != 0){
                        cnt_flutt[j]++;
                        cnt_soprasoglia[j] = 0;
                    }
                }

                if(cnt_flutt[j] > MAX_FLUTT){
                    //guasto NON pertinente
                    guasti[j] = 2;
                    sendAlarm(j, 1);
                }
            }
        }
    }
```

Codice STM32: verifica conclusione prova

```
        else{
            //mando l'avviso
            //stacco l'interruttore generale
            sendAlarm(1, 2);//scrivo al posto dell'indice del LED lo step che si è appena concluso

            //disabilito il timer
            TIM14->CR1 &= !TIM_CR1_CEN;
            TIM14->DIER &= !TIM_DIER_UIE;
            cnt = 0;
        }
    }


    int sum = 0;
    for(int i=0; i < N_CH; i++){
        if(guasti[i]!=0)
            sum++;
    }
    if(sum == N_CH){
        //Prova finita
        //Aspetto 1 secondo per non avere problemi di mancata ricezione del messaggio troppi vicini tra
        //l'ultimo led che si rompe e il messaggio di fine prova
        TIM15->CR1 |= TIM_CR1_CEN;
        TIM15->DIER |= TIM_DIER_UIE;
        hold_on_signal = 1;
        while (hold_on_signal != 0);

        sendAlarm(0,3); /* Il primo valore verrà ignorato dal modulo wifi ma mettiamo 0 perchè
        si verifica che il microcontrollore non riesce a fornire abbastanza potenza per tenere tutte le uscite alte */
    }

    TIM14->SR &= !TIM_SR_UIF;
}
```

Codice STM32: invio segnale di alert per ESP8266

```
void sendAlarm(int j, int mex_code){
    //Disabilito contatore del timer e l'enable dell'interrupt
    TIM14->CR1 &= !TIM_CR1_CEN;
    TIM14->DIER &= !TIM_DIER_UIE;

    GPIOC->ODR &= !0x1;//disabilito interuttore generale scrivendo 0 nel primo bit di GPIOC (che arriverà al gate dell'NMOS)
    /* Manda messaggio (implemento semplicemente come un pin di output a 1 che sarà un alert per il modulo wi-fi
    altri 3 pin di output li uso per capire in quale led si è verificato il guasto (o nel caso di fine step in quale dei due siamo)
    e ultimi due pin li uso per capire se il guasto è pertinente o no, se la prova o uno dei due step si è concluso */
    GPIOC->ODR |= ((0b111 & j) << 1) | ((0b11 & mex_code) << 4); //costruisco la stringa di bit che il modulo wifi deve leggere
    /*identificativo LED | mex_code*/
    GPIOC->ODR |= (0x1 << 6); //metto alto il segnale di alert dopo aver costruito tutta la stringa di bit

    /*Attivo TIM15 che conta fino a 1 sec per mantenere i segnali per il modulo Wi-Fi attivi
    la scheda Wi-Fi farà il polling ogni 0.1 sec e appena vede alto il segnale di alert
    manda il messaggio su Telegram e conta fino a 1 sec in modo da non inviare due volte lo stesso messaggio.*/
    TIM15->CR1 |= TIM_CR1_CEN;
    TIM15->DIER |= TIM_DIER_UIE;

    hold_on_signal = 1;
    while (hold_on_signal != 0);
    GPIOC->ODR &= !(0xFFFF);//rimetto a 0 tutti i bit dopo che sono sicuro che il modulo Wi-Fi abbiamo ricevuto il messaggio
    hold_on_signal = 1;
    while (hold_on_signal != 0);
    hold_on_signal = 1;
}
```

Codice ESP8266: connessione WiFi

```
void wifiConnection(){
  // WiFi impostato in station mode e disconnetti da altri router(solo se precedentemente connesso)
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
  // connessione al WiFi:
  Serial.print("Connecting Wifi: ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) { //si esce dal while quando la connessione è avvunuta con successo
    digitalWrite(wled, HIGH); //led status WiFi spento
    Serial.print(".");
    delay(500);
  }
  digitalWrite(wled, LOW);//led status WiFi acceso
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP()); //stampa l'ip locale
  client.setInsecure();

  bot.longPoll = 60;// costante utilizzata per migliorare i tempi di risposta del bot telegram
  }
```

Codice ESP8266: invia messaggio telegram

```
void sendAlertTelegram(String id){//funzione che interpreta la combinazione e manda il messaggio a id
    int guasto = (warning & 0x18)>>3; // filtra il codice di guasto (& 00011000), e lo shifta a destra di 3 pos
    int idLED = (warning & 0x07); // filtra i 3 bit utilizzati per l'indice del led o per il codice dello STEP
    switch(guasto){
        case(0):
        {String txt = "GUASTO PERTINENTE nel LED n° "+ String(idLED+1);
        bot.sendMessage(String(id), txt);}
        break;
        case(1):
        {String txt = "GUASTO NON PERTINENTE nel LED n° " + String(idLED+1);
        bot.sendMessage(String(id), txt);}
        break;
        case(2):
        {String txt = "FINE STEP " + String(idLED) + "\n1)Premi il punsalte \n2)Cambia la corrente su i LED \n3)Premi nuovamente il pulsante";
        bot.sendMessage(String(id), txt);}
        // (usiamo idLED per indicare in che STEP siamo)
        break;
        case(3):
        bot.sendMessage(String(id), "FINE PROVA (tutti i LED hanno manifestato un guasto).");
        break;
        default:
        bot.sendMessage(String(id), "ERRORE COMBINAZIONE warning");
        break;
    }
}
```

Codice ESP8266: setup pinout e connessione

```
void setup()
{
  Serial.begin(115200); // apri il monitor seriale

  // set Input e Output
  pinMode(wled, OUTPUT);
  digitalWrite(wled, HIGH);

  pinMode(alertPin, INPUT); //ALERT PIN = D5
  pinMode(wrn0, INPUT);
  pinMode(wrn1, INPUT);
  pinMode(wrn2, INPUT);
  pinMode(wrn3, INPUT);

  ids[0] = "291655246" ; // roberto (usato in debug)
  ids[1] = "-1001219067722"; // canale per gli allarmi sulle prove
  ids[2] = "000000000" ; // zeri (usato in debug)
  idallarm = ids[1];    // settare qui a quale id bisogna mandare gli allarmi
  wifiConnection(); //connessione al WiFI
  bot.sendMessage(String(idallarm), "BOT ATTIVO."); //primo messaggio che indica che il bot è in funzione
}
```

Codice ESP8266: lettura pin e polling

```
void readPin(){ //Funzione che crea la variabile warning leggendo i valori in input
    bitWrite(warning, 0, digitalRead(wrn0));
    bitWrite(warning, 1, digitalRead(wrn1));
    bitWrite(warning, 2, digitalRead(wrn2));
    bitWrite(warning, 3, digitalRead(wrn3));
    bitWrite(warning, 4, digitalRead(wrn4));
}
```

```
void loop()
{
  delay(100); // 100 millisecondi
  polling = digitalRead(alertPin);

  if (polling) {
    readPin();
    sendAlertTelegram(String(idallarm));
    delay(1000); // 1 secondo per evitare di leggere più volte lo stesso alert dall'STM32
    }
}
```

Conclusioni

- È stato realizzato un esempio di Sistema di monitoraggio per prove di vita accelerata sui LED
- Misura automatica dello stato del LED
- Log dei risultati della prova salvato sul cloud e accessibile in qualunque momento da qualunque piattaforma
- Notifiche su smartphone e PC
- Un eventuale upgrade potrebbe essere quello di comandare 2 transistor per ogni LED, che deviano la corrente dal LED guasto automatizzando ulteriormente le prove.