



# Esperimentazione di Elettronica

Andrea Boldetti

29 gennaio 2025

## Sommario

Relazione di laboratorio di Elettronica 2024/2025. In questa relazione vengono riassunte le esperienze fatte in laboratorio, dal funzionamento base di un microcontrollore, fino alla lettura di dati analogici presi con un SiPM con trigger.

Questa relazione avrà un focus specifico sulle funzionalità del microcontrollore e sui programmi creati per interagirci.

Alla fine della relazione verranno riportati i dati e i codici in un link a GitHub.

# Indice

<b>1</b>	<b>Microcontrollore</b>	<b>3</b>
1.1	Funzionamento generale . . . . .	3
1.1.1	Struttura interna . . . . .	3
1.1.2	Memoria . . . . .	3
1.1.3	Interrupt . . . . .	4
1.2	Periferiche . . . . .	4
1.3	Programmazione . . . . .	5
1.3.1	Struttura del programma . . . . .	5
1.3.2	Interrupt . . . . .	5
<b>2</b>	<b>Universal Synchronous Asynchronous Receiver Transmitter (USART)</b>	<b>6</b>
2.1	Funzionamento . . . . .	6
2.2	Programmazione . . . . .	6
<b>3</b>	<b>Analog Digital Converter (ADC)</b>	<b>8</b>
3.1	Funzionamento . . . . .	8
3.2	Programmazione . . . . .	8
<b>4</b>	<b>Direct Memory Access (DMA)</b>	<b>11</b>
4.1	Funzionamento . . . . .	11
4.2	Programmazione . . . . .	11
<b>5</b>	<b>Timer</b>	<b>13</b>
5.1	Funzionamento . . . . .	13
5.2	Programmazione . . . . .	13
<b>6</b>	<b>DAC e Comparatore</b>	<b>14</b>
<b>7</b>	<b>Scheda Analogica</b>	<b>15</b>
7.1	OP27 . . . . .	16
7.1.1	Configurazione Non Invertente . . . . .	16
7.1.2	Configurazione Invertente . . . . .	19
7.1.3	Conclusione . . . . .	20
7.2	AD848 . . . . .	21
7.2.1	Oscillazioni a Gain BASSO . . . . .	21
7.2.2	Gain elevato . . . . .	23
7.2.3	Conclusioni . . . . .	23
7.3	SiPm . . . . .	24
7.3.1	Setup . . . . .	24
7.3.2	Dark Count Rate . . . . .	24
<b>8</b>	<b>Risultati</b>	<b>25</b>
<b>9</b>	<b>Conclusioni</b>	<b>25</b>

# 1 Microcontrollore

Il microcontrollore usato è il NUCLEO H743ZI2 ed è stato usato il programma dato dalla fabbrica produttrice, STM.

## 1.1 Funzionamento generale

### 1.1.1 Struttura interna

Il  $\mu$  - controllore usato sfrutta la struttura interna di Harvard, ovvero con 2 memorie separate: una per i programmi e una per i dati.

Questa architettura, unita ad un doppio bus asincrono, permette di eseguire più operazioni in parallelo,

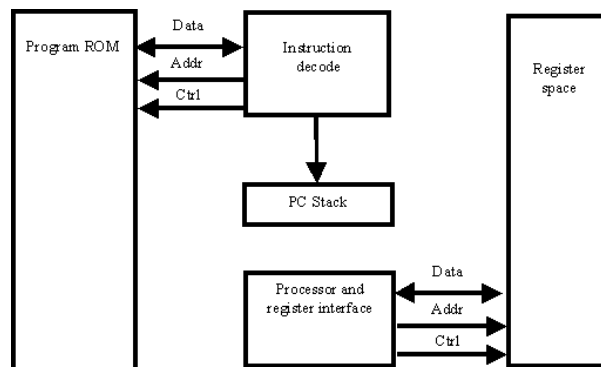


Figure 1-5 Harvard architecture block diagram

Figura 1: Struttura interna del microcontrollore

aumentando la velocità di esecuzione. Un esempio del perché questa architettura sia più veloce deriva dal fatto che in un singolo colpo di clock, il microcontrollore sia in grado di leggere l'istruzione e interagire con il dato da modificare.

Unita al doppio processore Cortex M7 e M4 e il clock da 480 MHz, l'architettura Harvard permette a questa scheda di essere l'ideale per processi molto complessi ma che hanno bisogno di estrema velocità.

### 1.1.2 Memoria

All'interno del microcontrollore esistono 3 tipi di memoria:

- Flash: Una memoria non volatile che sacrifica la velocità di scrittura per aumentare la capacità di lettura. Nei microcontrollori è usata per salvare i programmi.
- RAM: Una memoria volatile con lo scopo di salvare i dati della singola esecuzione del programma. Per stare al passo con la CPU serve velocità di lettura e scrittura
- ROM: Una memoria non volatile di sola lettura. Viene scritta dai produttori e contiene informazioni fondamentali per il corretto funzionamento del microcontrollore, oltre alla definizione di registri e celle di memoria speciale.

Ogni memoria è divisa in celle, ognuna delle dimensioni di un byte e caratterizzata da un indirizzo unico. Per salvare dati più grandi di un byte, si usano più celle contigue, così che sapendo dimensione del dato e indirizzo di partenza, si possa trovare univocamente il dato.

La memoria è connessa alla CPU con dei BUS dedicati, per accedere alle celle, la CPU abilita delle uscite dallo stato di alta impedenza.

Per quanto riguarda la RAM, oltre ad essere divisa in celle, i produttori della scheda dividono la memoria in registri. Dandogli dei ruoli specifici all'interno della CPU. Questa divisione permette di

avere un modo più semplice di collegare le celle di memoria al loro significato e permette alla CPU di lavorare in maniera più efficiente sulle celle di memoria a utilizzi speciali.

Mentre, nella memoria Flash, i programmi sono salvati in pile di celle di memoria, ognuna con un codice operativo che, letto dalla CPU, si trasforma in un'istruzione. L'istruzione in questione non ha accesso solo alla RAM in cui sono contenuti i dati di esecuzione ma anche alla memoria flash stessa; questo permette alla CPU di saltare istruzioni nel caso in cui non siano richiesti.

### 1.1.3 Interrupt

Un'altra caratteristica fondamentale dei computer in generale ma anche dei microcontrollori, sono gli Interrupt. Gestiti da una componente specifica all'interno dei microprocessori, sono segnali che vengono inviati alla CPU per interrompere l'istruzione in corso, dando priorità ad un'altra istruzione. Questo meccanismo permette di rimanere in attesa di un evento senza dover per forza limitarsi a controllare ciclicamente se è avvenuto.

Seppur molto utile, l'uso degli interrupt complica la programmazione e la leggibilità del codice.

## 1.2 Periferiche

Il microcontrollore è dotato di diverse periferiche e funzionalità ma noi siamo andati a guardare solo quelle interessanti allo scopo di trattare dati analogici e digitali. Queste funzionalità sono:

- Timer
- Universal Synchronous Asynchronous Receiver Transmitter (USART)
- Analog Digital Converter (ADC)
- Direct Memory Access (DMA)
- Digital Analog Converter (DAC)
- Comparatore

Queste funzionalità sono quelle utili per essere in grado di leggere dati analogici presi da un sensore e salvarli ad un PC.

## 1.3 Programmazione

Per programmare sul microcontrollore è stato usato il programma STM32CubeIDE, un IDE fornito dalla STMicroelectronics. Il linguaggio di programmazione usato è C99. All'interno dell'IDE sono inserite delle librerie che permettono di interfacciarsi con le periferiche del microcontrollore, le quali contengono gli indirizzi dei registri e le maschere per collegare ogni bit dei registri al funzionamento sulla periferica.

I registri prendono il nome dalla periferica che controllano e ogni registro è diviso in sottoregistri a cui si può accedere usando i puntatori.

Per modificare i registri è consigliato usare maschere, quindi dei numeri binari che, se applicate delle operazioni di AND e OR permettono di andare a modificare solo determinati bit.

### 1.3.1 Struttura del programma

Il programma si divide in diversi file:

- **main.c:** Il file principale del programma, contiene la funzione `main()` e le funzioni di inizializzazione delle periferiche. Il compilatore lancia il programma a partire da questo file.
- **stm32h7xx\_it.c:** Il file contenente tutti gli interrupt abilitati. Ogni interrupt è una funzione diversa che non comunica direttamente con il resto.
- **Librerie create dall'utente:** File contenenti le funzioni create dall'utente per interfacciarsi con le periferiche.

Questi sono solo alcuni dei file presenti nel progetto, ma sono quelli che effettivamente sono modificati dall'utente.

### 1.3.2 Interrupt

Tutte le implementazioni delle funzionalità sono state fatte usando i relativi Interrupt. Un interrupt è un segnale che viene inviato alla CPU per interrompere l'istruzione in corso e dare priorità ad un'altra istruzione. Questo meccanismo permette di rimanere in attesa di un evento senza dover per forza limitarsi a controllare ciclicamente se è avvenuto.

Così facendo si è in grado di fare più cose contemporaneamente e di ottimizzare la velocità di esecuzione del programma.

Ogni periferica ha uno o più interrupt dedicati che vengono attivati con eventi diversi. In particolare:

- **USART:** Ha un interrupt dedicato per la ricezione e uno per la trasmissione, che vengono chiamati quando il buffer in ricezione è pieno o quando il buffer in trasmissione è vuoto.
- **ADC:** Ha un interrupt dedicato per la fine della conversione, che viene chiamato quando il valore analogico è stato convertito in digitale.
- **DMA:** Ha un interrupt dedicato per la fine del trasferimento, che viene chiamato quando il trasferimento di dati è finito.
- **Timer:** Ha un interrupt dedicato per la fine del conteggio, che viene chiamato quando il timer ha finito di contare.

Per ogni interrupt, esiste una funzione creata ad hoc che possiamo riempire con il codice che vogliamo e che possiamo decidere se attivare o disattivare a piacimento.

## 2 Universal Synchronous Asynchronous Receiver Transmitter (USART)

La prima periferica usata è stata la USART. Usart ci permette di comunicare con il PC attraverso la Seriale.

### 2.1 Funzionamento

Seppur la periferica permetta il trasferimento sincrono dei dati, noi la useremo come USART, quindi una versione precedente.

Questa periferica per funzionare ha bisogno di un collegamento diretto con il dispositivo a cui comunicare, questo collegamento nel nostro cavo consiste in 3 cavi: TX, RX e GND.

- TX: Trasmettitore, invia i dati al PC
- RX: Ricevitore, riceve i dati dal PC
- GND: Collegamento a terra, serve per chiudere il circuito e tarare entrambi i dispositivi allo stesso 0

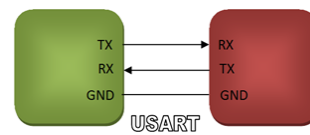


Figura 2: Collegamento tra microcontrollore e PC

I dati sono mandati sottoforma di impulsi elettrici. Dato che i due dispositivi sono collegati con un singolo cavo, i dati sono inviati in ordine e, l'unica cosa di cui i due dispositivi hanno bisogno è la velocità di trasmissione, che deve essere uguale tra i 2 per assicurare la giusta lettura.

La velocità di trasmissione è misurata in baud, ovvero il numero di bit trasmessi in un secondo e per questo si chiama Baud Rate. Di base è a 9600 baud, ma a seconda delle necessità può essere alzata fino al limite strumentale.

Per evitare problemi di lettura, il ricevitore overcampiona i dati per poi farne una media. Così facendo, il rumore dovuto a cavi troppo lunghi o a interferenze elettromagnetiche viene minimizzato.

### 2.2 Programmazione

Usart, nel nostro caso, è stato usato nel seguente modo: Inizialmente abbiamo spedito un carattere specifico dal pc al microcontrollore, questo carattere l'abbiamo usato come trigger per far partire la lettura dei dati, finita la lettura dei dati abbiamo trasmesso tutti i dati al computer.

```
1 void ESPE_USART_char_start(void){
2
3     if ( USART3 ->ISR & USART_ISR_RXNE_RXFNE){ // Se il buffer è pieno
4         if ( USART3 -> RDR == char_trigger){ // Se il carattere è uguale al trigger
5             flag_Trigger_EN = 1;
6
7         }
8     }
9 }
```

Nel codice 2.2, andiamo ad usare 2 registri di USART: ISR e RDR. Il primo, l'Interrupt Status Register, contiene flag per controllare lo status di USART, mentre il secondo, il Receive Data Register, contiene il dato ricevuto.

```
1 #define len_vec 100
2 uint_8 vec_dati[len_vec];
3 uint_8 len = sizeof(uint_8)/sizeof(char)*len_vec;
```

```

4 char *str = vec_dati;
5 uint_8 indice = 0;
6
7
8 void ESPE_USART_send_data(void){
9     if( USART3 ->ISR & USART_ISR_TC){ // Se il buffer è vuoto
10         if (indice < len){
11             USART3->TDR = *(str+indice); // Invia il dato
12             indice++;
13         }else{
14             if(indice == len){
15                 USART3 -> TDR = '\r'; // Invia il carattere di fine riga
16                 indice = 0;
17             }
18         }
19     }
20 }

```

In questo codice, invece andiamo a usare il registro TDR, il Transmission Data Register, per mandare i dati al PC, ma, dato che i dati sono messi in un vettore di uint8 e il TDR ha le dimensioni di un char, il numero di byte da inviare non corrisponde alla dimensione del vettore. Questa discrepanza è risolta dalla variabile \*str che è una variabile puntatore di tipo char puntata all'inizio del vettore.

Oltre a queste funzioni è necessario definire un'altra funzione ausiliaria con la quale si passa dalla lettura dei dati al loro invio.

```

1 void ESPE_USART_invert_mode(void){
2     if(USART3 -> CR1 & USART_CR1_RXNEIE){
3         USART3 -> CR1 &= ~USART_CR1_RXNEIE;
4         USART3 -> CR1 |= USART_CR1_TCIE;
5     }else if(USART3 -> CR1 & USART_CR1_TCIE){
6         USART3 -> CR1 |= USART_CR1_RXNEIE;
7         USART3 -> CR1 &= ~USART_CR1_TCIE;
8     }
9 }

```

Questa funzione cambia il metodo di trigger dell'interrupt di USART, così facendo si evitano problemi con interrupt in lettura mentre l'invio è in corso o viceversa.

## 3 Analog Digital Converter (ADC)

L'ADC è una periferica che permette di convertire un segnale analogico in un segnale digitale.

### 3.1 Funzionamento

Per convertire valori analogici in digitale esistono diversi metodi, il microcontrollore usato usa il metodo di conversione ad approssimazioni successive.

Seppur questo metodo non permette di avere i dati convertiti istantaneamente come può essere un Flash ADC, permette di avere una maggiore precisione e di avere un errore minore.

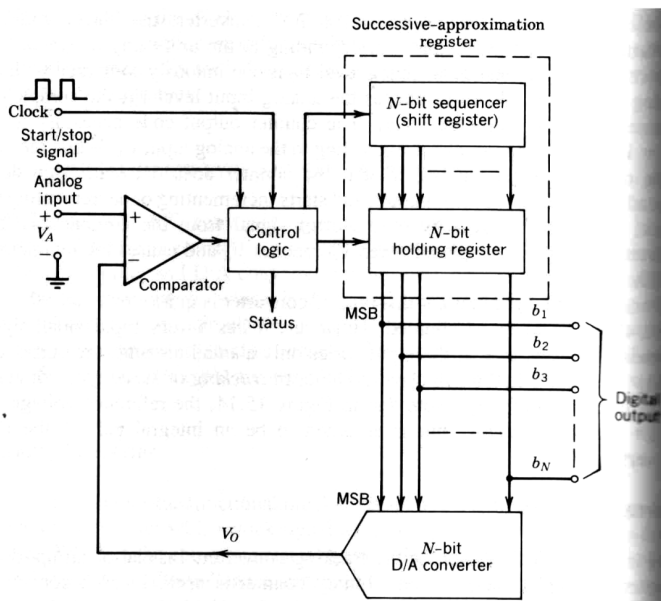


Figura 3: Schema di funzionamento dell'ADC

L'ADC ad approssimazioni successive per funzionare sfrutta un comparatore e un metodo di ricerca binaria. Attraverso una conversione Digitale-Analogica, il microcontrollore invia un segnale analogico al comparatore, il quale ritornerà un segnale positivo se il segnale inviato è maggiore di quello ricevuto, altrimenti ritornerà nullo.

Così facendo, il microcontrollore può ricavare con precisione e con poche iterazioni del metodo la rappresentazione binaria migliore per il valore analogico.

Altri aspetti positivi di questo metodo sono la presenza di un singolo comparatore, il quale permette di diminuire notevolmente l'errore nella misura e la presenza di registri per la ricerca binaria, i quali rendono il processo più veloce di altri metodi.

### 3.2 Programmazione

Parte importante della programmazione con l'ADC è la calibrazione dell'apparato. Infatti, prima dell'utilizzo

```
1 void ESPE_ADC_init(void){
2
3     // azzeriamo per evitare casini di configurazione
4     ADC3 -> SQR1 = 0;
5
6     // ogni numero è collegato ad un pin a sèstante
7     ADC3->SQR1 |= 0 << ADC_SQR1_L_Pos; // ti dice quante misure deve prendere (n+1)
8     ADC3->SQR1 |= 0 << ADC_SQR1_SQ1_Pos; // ti dice qual è la prima misura da fare
9
10    ADC3->PCSEL |= ADC_PCSEL_PCSEL_0; //segna quali sono i canali in lettura per
11                                     //velocità massima
12
13    ADC3 -> CR &= ~ADC_CR_DEEPPWD_Pos; //Deep power down state
14                                     //(se attivo non overcloccka)
```



```

15     ADC3 -> CR |= 1 << ADC_CR_ADVREGEN_Pos; //Voltage regulator activated
16
17     ADC3 -> CR &= ~ADC_CR_ADALDIF_Pos; //seleziona modalità differenziata
18                                     //di calibrazione (a 0)
19     ADC3 -> CR |= 1 << ADC_CR_ADALLIN_Pos; //seleziona la modalità lineare
20                                     //di calibrazione (a 1)
21     ADC3 -> CR &= ~ADC_CR_ADEN_Pos; //Controlliamo che l'ADC non sia acceso e che
22                                     //il bit sia stato resettato
23     ADC3 -> CR |= 1 << ADC_CR_ADICAL_Pos; // Inizia la calibrazione
24
25     while( ADC3->CR & ADC_CR_ADICAL ){
26         //Aspetti che la calibrazione sia finita, il bit viene cambiato dall'hardware
27     }
28
29     ADC3->ISR &= ~ADC_ISR_ADDRDY_Pos; //Controlli che il bit per l'inizio della
30                                     //presa dati sia a 0
31     ADC3->CR |= 1 << ADC_CR_ADEN_Pos; //Attiviamo l'ADC (non la presa dati)
32     while( !(ADC3->ISR & ADC_ISR_ADDRDY)){
33         //Aspettiamo che sia setupato correttamente
34     }
35
36     ADC3 -> IER |= ADC_IER_EOCIE; //Attiviamo l'interrupt
37
38     ADC3 -> SMPR1 |= 0 << ADC_SMPR1_SMP0_Pos;
39     //Possiamo aggiungere un ritardo di (n = 0) cicli prima della lettura
40
41 }

```

Oltre a impostare le misure da fare e il numero di misure che ogni ADC deve fare su ogni pin, all'interno della funzione è contenuto il processo di calibrazione dell'ADC. La calibrazione è fatta automaticamente dal microcontrollore usando dei valori di riferimento ma questo processo occupa tempo e, se non si aspettasse la fine della calibrazione per prendere la misura si avrebbero valori errati. Quindi c'è bisogno di un ciclo *while()* per aspettare la fine della calibrazione.

La calibrazione è fatta accedendo a 2 Registri in particolare: il registro CR e il registro ISR. Il primo contiene i bit per la calibrazione e l'attivazione dell'ADC, mentre il secondo contiene i bit per controllare lo stato dell'ADC.

Una volta finita la calibrazione, scrivendo

```
ADC3-> CR |= ADC_CR_ADSTART
```

l'ADC inizia a prendere le misure.

Per leggere i dati, invece, si può inserire nell'interrupt una funzione dedicata del tipo:

```

1
2     #define len_vec 100
3     uint_16 vec[len_vec];
4     uint_8 counter_ADC = 0;
5     uint_8 conversion_flag = 0;
6
7     void ESPE_ADC_data(void){
8         if( counter_ADC < len_vec){
9             vec[counter_ADC] = ADC3->DR;
10            counter_ADC++;
11        }else{
12            conversion_flag = 1;
13            ADC3-> CR &= ~ADC_CR_ADSTART;
14        }

```

15  
16

```
}
```

In questo codice, il registro DR, il Data Register, contiene il valore misurato dall'ADC in Volt. Se il valore che si vuole non è in volt, è necessario fare una conversione. La conversione dipende da fattori come la tensione di riferimento dell'apparecchio che fa la misura e il valore massimo misurabile dall'ADC.

L'ultima riga, invece è necessaria per fermare l'ADC una volta che ha finito di prendere le misure.

## 4 Direct Memory Access (DMA)

Usando ADC e USART per fare misure e trasferire dati, una delle più grandi limitazioni in velocità è il tempo di trasferimento dei dati dalla periferica al microcontrollore. Questo tempo è dovuto al fatto che la CPU deve leggere i dati dalla periferica e salvarli in memoria.

### 4.1 Funzionamento

Il processo per trasferire dati dalla periferica alla memoria presume che il dato, prima di essere salvato nella memoria RAM, sia salvato nella memoria interna alla CPU in modo che la CPU ci possa lavorare. Questo passaggio si può rimuovere usando il DMA.

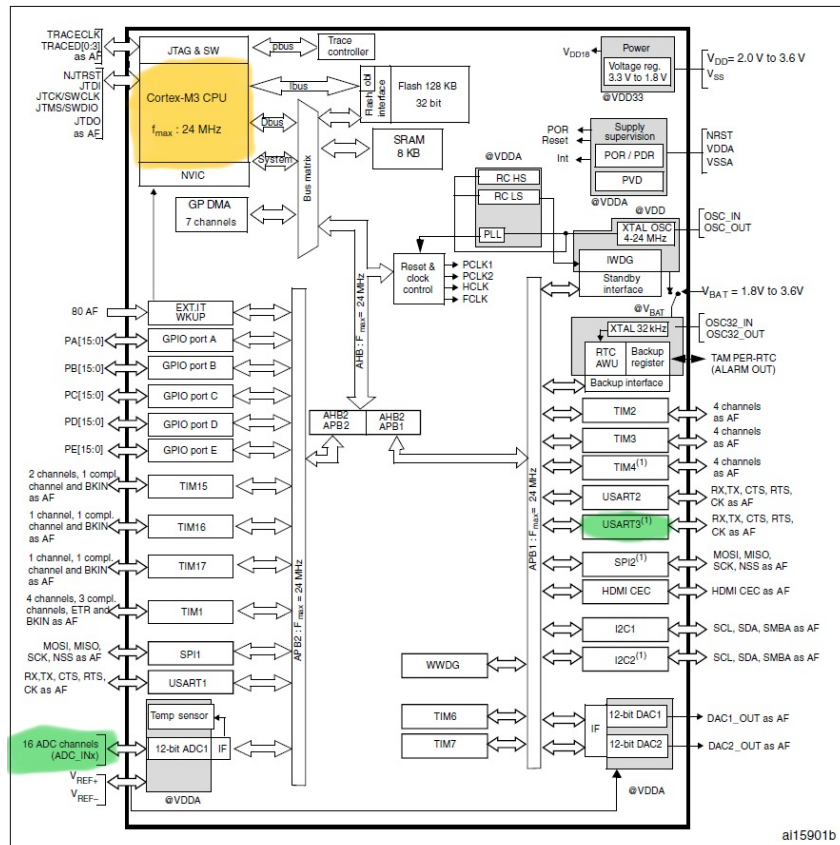


Figura 4: Struttura interna del microcontrollore

Una volta configurato, il DMA permette di evitare il carico computazionale della CPU per il trasferimento dei dati dando l'accesso diretto alla periferica di accedere alla memoria. Così facendo, oltre alla CPU più libera, i dati saranno salvati direttamente dalle periferiche nella memoria nella maniera più veloce possibile.

### 4.2 Programmazione

Dato che il protocollo del DMA ti permette di salvare i dati in autonomia, la parte di programmazione consiste solo nel configurare correttamente il DMA.

Dato che può essere molto complesso, parte della configurazione è stata fatta con il code generator di STM32CubeIDE e consiste nel collegare uno *stream* del DMA alla periferica che si vuole usare e dargli la dimensione del dato e in che tipo di memoria si vuole salvare.

Una volta configurato su STM32CubeMX, c'è bisogno di configurare specificatamente il DMA per la periferica che si vuole usare e aggiungere alla periferica l'istruzione che permette di essere collegata al DMA.

In generale, questo tipo di configurazione può essere riassunta in 4 righe di codice. Usiamo come esempio il DMA per l'ADC:

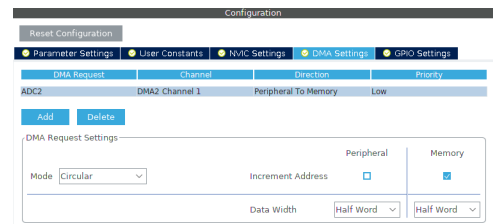


Figura 5: Configurazione del DMA

```

1 DMA1_Stream0 -> PAR = (uint32_t) &ADC3->DR; //Collegare il DMA al registro
2                                     //che contiene il dato
3 DMA1_Stream0 -> MOAR = (uint32_t) &vec_dati; //Dare al DMA l'indirizzo della memoria
4                                     //in cui salvare il dato
5 DMA1_Stream0 -> NDTR = len; // Dimensione del dato
6 ADC3 -> CFGR |= (3<<ADC_CFGR_DMNGT_Pos); // Configura l'ADC per usare il DMA

```

e poi si attiva il DMA con la funzione:

```

1 DMA1_Stream0->CR |= DMA_SxCR_EN // Attiva il DMA

```

## 5 Timer

Parte fondamentale per il funzionamento del microcontrollore è il Timer. In particolare, il timer è fondamentale per scandire il rateo delle operazioni della CPU e per dare un tempo di esecuzione alle periferiche.

Nel microcontrollore che abbiamo usato non c'è un singolo timer e ognuno ha caratteristiche, precisione e funzionalità diverse. Per i nostri scopi, noi abbiamo interagito con il Timer 6 che è un timer PLL

### 5.1 Funzionamento

I clock PLL sono dei clock che sfruttano un clock di riferimento, nel nostro caso di quarzo, per generare un clock a frequenze più alte. Questo processo avviene attraverso un Voltage Control Oscillator il quale è in grado di generare onde quadre a frequenze molto alte e un Filtro passa Basso che fa da filtro per le armoniche generate dal VCO.

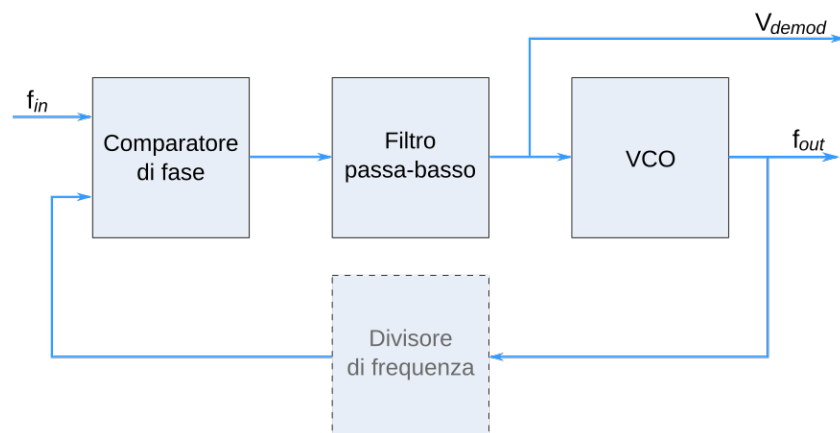


Figura 6: Schema di funzionamento del Timer

Questi 2 componenti, collegati ad un comparatore e al cristallo di riferimento riescono a generare onde quadre ad un ritmo costante usando il quarzo, molto stabile, come riferimento per non perdere la periodicità.

A questo processo si aggiungono altri componenti come il prescaler e il contatore che permettono di avere un controllo più preciso sulla frequenza del clock generato.

Per avere la massima velocità di esecuzione del programma, vanno modificate le frequenze dei clock. Così facendo, si possono velocizzare i processi più lenti a discapito di altri e di un maggior consumo di energia.

### 5.2 Programmazione

Per configurare il Timer 6, abbiamo usato STM32CubeMX. Questo programma permette di configurare ogni singolo timer a piacere, a patto di non uscire dai limiti del microcontrollore. Questo processo si fa attraverso un'interfaccia grafica dedicata

Una volta configurato, si può passare all'inizializzazione del Timer.

```
1 void ESPE_TIM6_init(void){
2     TIM6->CNT = 0; //Resetta il contatore
3     TIM6->ARR = 10; //Setta il valore di auto reload
4     TIM6->PSC = 24; //Setta il prescaler
5 }
```

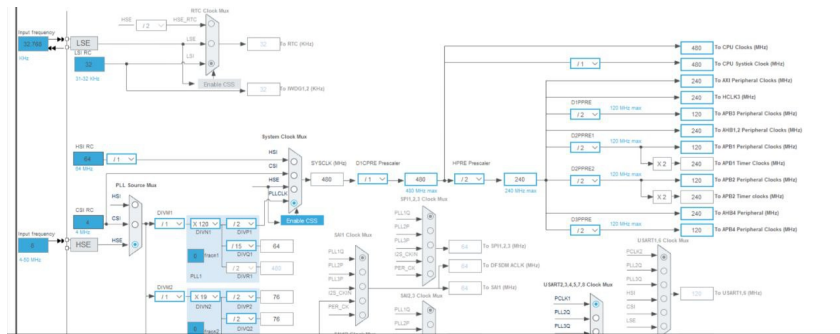


Figura 7: Configurazione del Timer

Così facendo si può ulteriormente modificare la frequenza del timer e quindi la velocità delle periferiche ad essere collegate.

Per attivare il timer, si usa la funzione:

```
1 TIM6->CR1 |= TIM_CR1_CEN;
```

Nel caso ce ne sia necessità il timer ha anche un interrupt che viene chiamato quando il contatore arriva al valore di auto reload, ma l'utilizzo del timer è stato quello di collegarlo all'ADC per far sì di avere misure equamente distanziate nel tempo, cosa non scontata a causa del metodo di conversione dell'ADC.

## 6 DAC e Comparatore

## 7 Scheda Analogica

**Lavoro di Gruppo:** Yehan Edirisinghe, Andrea Boldetti, Elisa Minelli

In questa sezione viene discussa la caratterizzazione degli Amplificatori Operazionali e il loro utilizzo all'interno della scheda analogica fornita:

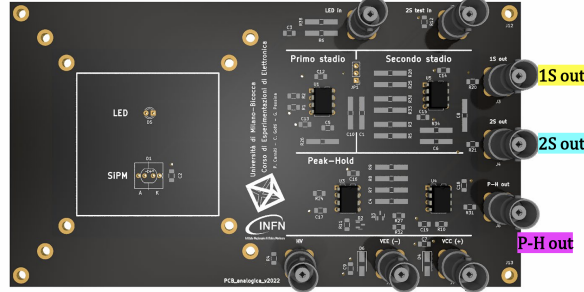
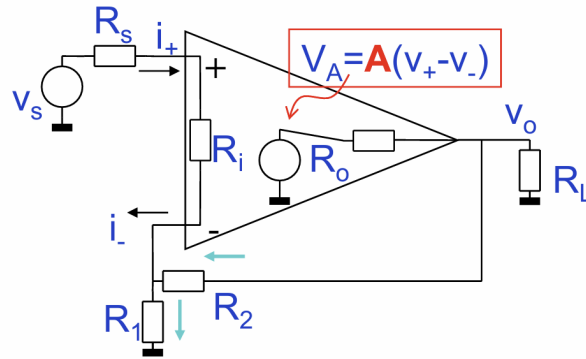


Figura 8: Scheda Analogica

Il nostro obiettivo è rendere il segnale uscente da un SiPm a stato solido leggibile dall'ADC del microcontrollore. Siccome la risposta del sensore ad un fotone è a tensioni troppo basse per la risoluzione dell'ADC, è necessario implementare degli stadi di amplificazione.

**Nota** Per le funzioni di trasferimento degli amplificatori è stata fatta la seguente semplificazione:



$$T = -\frac{R_i}{R_s + R_i} \frac{AR_1}{R_1 + R_2} \frac{R_L}{R_o + R_L} \sim -A\beta$$

Questo è accettabile siccome la resistenza del generatore è stata impostata a  $R_s \sim 50\Omega$  e  $R_L \sim 1M\Omega$ .

## 7.1 OP27

Il primo amplificatore che vogliamo studiare è l' OP27 che dai datasheet risulta essere un amplificatore a singolo polo dominante ovvero della forma:

$$A(s) = \frac{A_o}{1 + s\tau_A}$$

Le cui caratteristiche sono descritte in dettaglio nei datasheet.

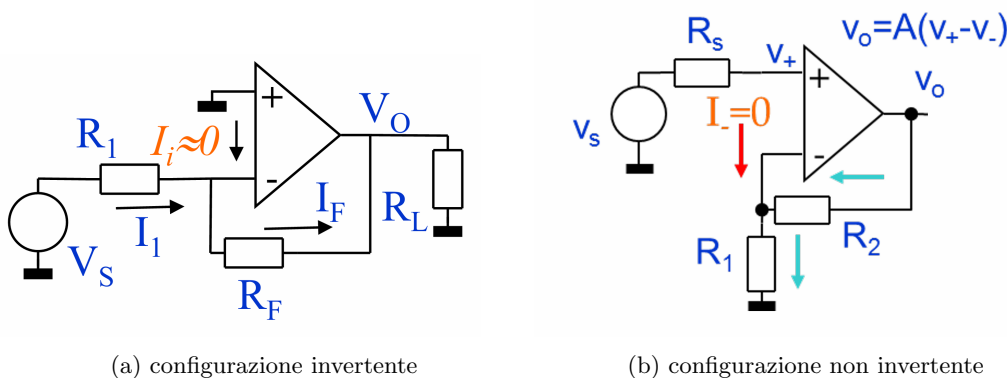
In particolare è segnata una **Bandwidth** di circa **8Mhz** che risulterà essere il dato di nostro interesse.

L'amplificatore può essere configurato in due modalità, ovvero: configurazione invertente e configurazione non invertente come mostrato in figura:

### FEATURES

**Low noise:** 80 nV p-p (0.1 Hz to 10 Hz), 3 nV/√Hz  
**Low drift:** 0.2 μV/°C  
**High speed:** 2.8 V/μs slew rate, 8 MHz gain bandwidth  
**Low Vos:** 10 μV  
**CMRR:** 126 dB at VCM of ±11 V  
**High open-loop gain:** 1.8 million  
**Available in die form**

Figura 9: Feature OP27



Queste risultano entrambe utili a seconda dello stadio di amplificazione. La configurazione invertente permette un guadagno maggiore a costo di invertire il segnale. La configurazione non invertente invece non altera il segnale ma applica semplicemente un'amplificazione.

### 7.1.1 Configurazione Non Invertente

In questa configurazione abbiamo un guadagno di Gain  $\frac{1}{\beta} = \frac{R_1 + R_2}{R_1}$  nel caso di amplificatore **ideale**. Collegando nella scheda resistenze  $R_s = 0.5\Omega$ ,  $R_1 = 2.2k\Omega$  e  $R_2 = 2.2k\Omega$  ci aspettiamo un gain di 2.

Siccome l'amplificatore reale dista dal modello ideale di un polo complesso, è necessario considerare il fattore  $\frac{A(\omega)}{\beta}$  al posto di  $\frac{1}{\beta}$  nella sua funzione di trasferimento dove  $A(\omega)$  denota la dipendenza dalla frequenza. Per poter caratterizzare questo comportamento in frequenza vogliamo trovare la **banda** dell'amplificatore.

È possibile utilizzare tre modalità diverse:

1. Analisi di onde sinusoidali
2. Analisi su un'onda quadra
3. Analisi dello spettro

**Analisi sinusoidale con  $R_1 = 2.2k\Omega$  e  $R_2 = 2.2k\Omega$ :** Per questa modalità è stato impostato un segnale sinusoidale all'ingresso non invertente dell'OP27 ad una ampiezza nota e osservato tramite oscilloscopio la risposta del circuito nel dominio del tempo alle varie frequenze. Per il calcolo della banda è sufficiente trovare la frequenza per cui il gain diminuisce di 3dB, ovvero perde il 30% della sua ampiezza.

Come frequenza bassa è stato usato un segnale sinusoidale di circa 1KHz mentre la frequenza per cui il segnale perde 3dB è di circa 3.9Mhz. Per la banda si trova: **Banda**  $= \frac{\omega_F}{\beta} = 7.8Mhz$ .

**Analisi in onda quadra a  $R_1 = 2.2k\Omega$  e  $R_2 = 2.2k\Omega$ :** Per questa modalità è stata impostata nel generatore un'onda quadra e tramite la modalità di misura dell'oscilloscopio sempre nel dominio del tempo, è stato acquisito il valore di **rise time** della risposta dell'amplificatore. Con rise time si intende il tempo che impiega il segnale a passare dal 10% al 90% della sua ampiezza.



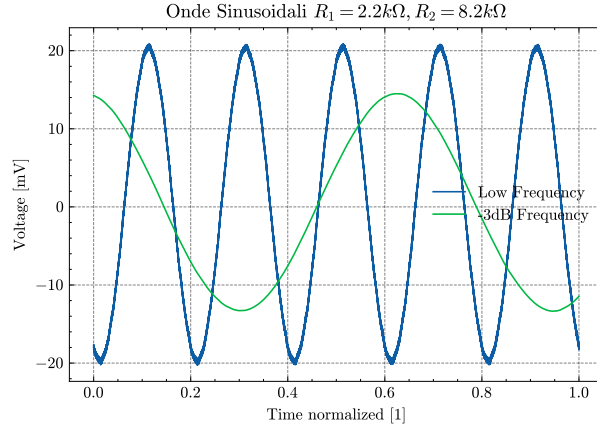


Figura 11: Segnali di uscita dall'amplificatore per frequenza a risposta piatta e a -3dB, ([link dati](#))

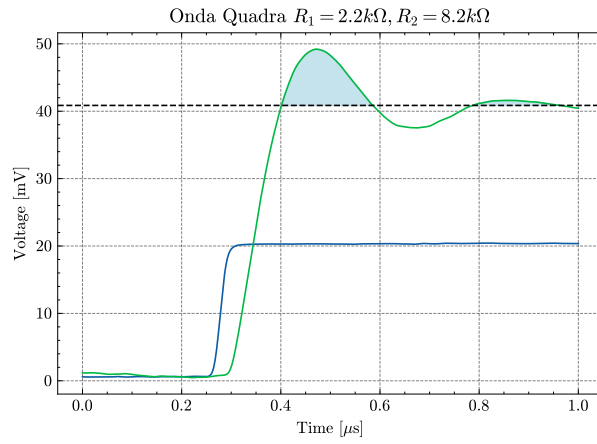


Figura 12: Risposta ad onda quadra, ([link dati](#))

Il rise time misurato dall'oscilloscopio è di 79.5 ns che corrispondono ad una frequenza a -3dB di:

$$f_{-3dB} = \frac{0.35}{t_r} = 4.4 \text{ Mhz}$$

e dunque una **banda di 8.8Mhz**.

Notiamo come in risposta all'onda quadra, l'amplificatore presenti un'overshoot non trascurabile: circa il 20% rispetto al valore aspettato. Questo può essere dovuto a diversi fattori:

- la presenza di capacità parassite all'interno dell'amplificatore
- la presenza di un secondo polo nell'amplificatore che abbia un'influenza inaspettata anche a frequenze basse

Prendendo in considerazione l'ipotesi di capacità parassite all'interno dell'amplificatore, si può modificare la formula della funzione di trasferimento come segue

$$T(s) = \frac{R_1 + R_2}{R_1} \frac{1 + s\tau_2}{(1 + s\tau_1)(1 + s\tau_A)} A_o \quad (1)$$

Il quale motiva la presenza del doppio polo. Sfortunatamente, questa ipotesi va scartata in quanto dai dati risulta che le capacità dovrebbero essere molto maggiori del pico Farad, il quale risulta insolito.

Risulta quindi la seconda ipotesi, ovvero che l'OP27 non sia accettabile come amplificatore a polo singolo ma è necessario considerare anche i poli di ordine superiore per avere delle misure coerenti con la teoria

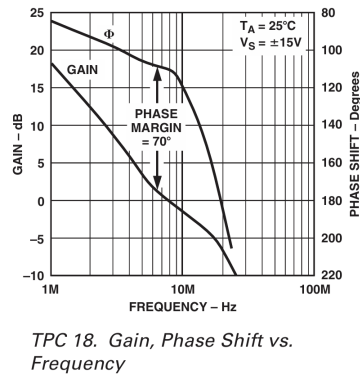


Figura 13: Matgine di fase OP27

Guardando nei datasheet la fase dell'amplificatore si nota che a gain bassi ci si può ridurre nella condizione con margine di fase ridotto. In questa zona possiamo dedurre che si abbiano effetti non previsti come l'andamento instabile osservato nelle misure sperimentali.

Inoltre questa ipotesi a differenza delle altre è consistente con l'andamento osservato per l'overshoot in figura 15 che diminuisce all'aumentare del gain assegnato.

**Nota 2** Siccome in alcuni casi il generatore di onda quadra non risulta perfetto è necessario considerare il rise time di quest'ultimo:

$$t_r = \sqrt{t_m^2 + t_g^2}$$

dove  $t_m$  è il rise time misurato e  $t_g$  il rise time del generatore da solo.

**Analisi Spettrale per  $R_1 = 2.2k\Omega$  e  $R_2 = 2.2k\Omega$ :** L'ultima misura sfrutta il rumore bianco, ovvero rumore che ha potenza spettrale piatta e dunque uguale a ogni frequenza. Avendo selezionato nel generatore un rumore bianco, è stata utilizzata la funzione FFT dell'oscilloscopio per ottenere lo spettro dell'amplificatore.

È possibile in questo caso utilizzare manualmente due cursori per individuare il gain a risposta piatta e dato questo, individuare la frequenza a -3dB da quest'ultima.

Nella misura otteniamo una risposta piatta a circa -45dB e una  $f_{-3dB} = 3.8MHz$  da cui una **banda di 7.6Mhz**.

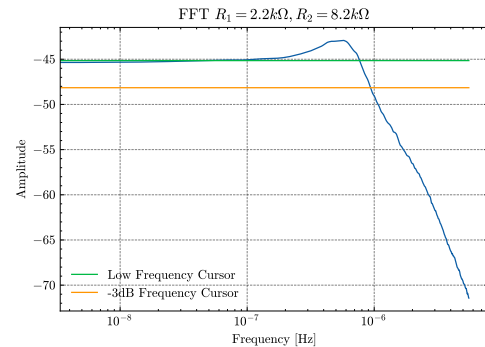


Figura 14: ([link dati](#))

**Cambio Resistenze** Variando la resistenza  $R_2$  è possibile cambiare il gain del circuito e studiare l'andamento della banda. Ripetendo il processo eseguito prima per valori di  $R_2 = \{2.2k\Omega, 8.2k\Omega, 10k\Omega, 17.2k\Omega\}$  è possibile notare il seguente andamento:

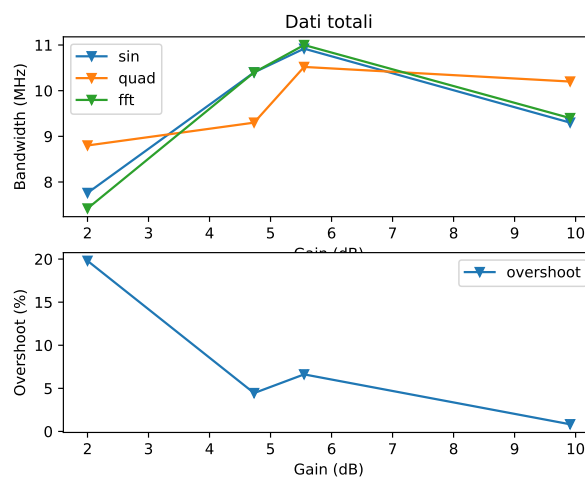


Figura 15: Rappresentazione grafica dell'andamento per configurazione Non-invertente

Gain [1]	Sin [Mhz]	Square [MHz]	Fft [MHz]	Overshoot [MHz]
2	7.76	8.8	7.42	19.8
4.73	10.4	9.3	10.4	4.45
5.55	10.92	10.52	11	6.62
9.91	9.3	10.2	9.4	0.83

Tabella 1: Rappresentazione a tabella dei dati per configurazione Non-invertente

Il valor medio di banda ricavato è di **9.6MHz**.

**Conclusioni** È stato possibile con tre metodi diversi eseguire un'analisi della risposta del circuito alle frequenze in ingresso. In particolare è stato riscontrato un polo complesso che porta ad una larghezza di **banda**  $\sim 9.6Mhz$  che risulta in accordo entro un 20% con il valore dei datasheet.

### 7.1.2 Configurazione Invertente

L'altra configurazione interessante per un amplificatore è quella invertente in cui il segnale entra nell'ingresso negativo  $V^-$ . In questa configurazione si ha un guadagno di

$$Gain = -\frac{R_2}{R_1}$$

Il procedimento per caratterizzare la banda risulta identico ed è possibile nelle tre modalità citate prima.

Questo è un esempio di risposta osservato per resistenze di  $R_1 = R_2 = 2.2k\Omega$

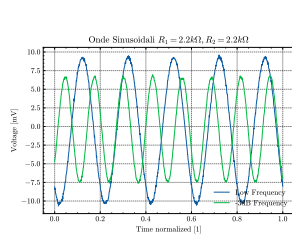


Figura 16: Analisi con segnale sinusoidale

Da notare che l'uscita è invertita rispetto al segnale. Il resto dei dati è al link: ([link dati](#))

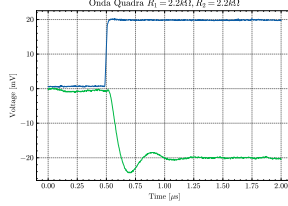


Figura 17: Analisi con onda quadra (overshoot < 20%)

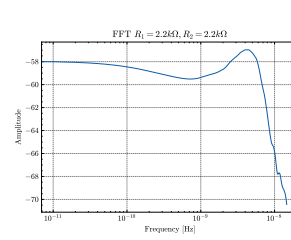


Figura 18: Analisi nel dominio delle frequenze

**Risultato:** Applicando il processo di misura per varie resistenze  $R_2 = \{2.2k\Omega, 8.2k\Omega, 10k\Omega, 17.8k\Omega\}$  è possibile vedere l'andamento seguente:

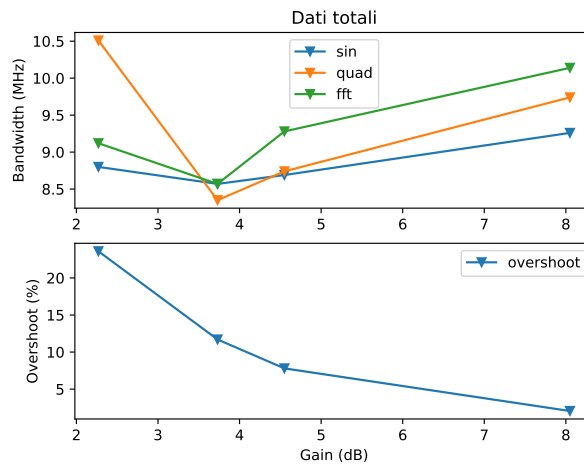


Figura 19: Rappresentazione grafica dell'andamento per configurazione Non-invertente

Gain [1]	Sin [Mhz]	Square [MHz]	Fft [MHz]	Overshoot [MHz]
2.27	8.8	10.51	9.12	23.6
3.73	8.57	8.35	8.57	11.7
4.55	8.69	8.74	9.28	7.8
8.05	9.26	9.74	10.14	2.05

Tabella 2: Rappresentazione a tabella dei dati per configurazione Non-invertente

Come per il caso non invertente si osserva una diminuzione dell'overshoot all'aumentare del gain nel circuito.

### 7.1.3 Conclusione

L'unione dei dati presi tramite le due configurazioni porta al seguente andamento:

È possibile notare che tra i due modelli si ottiene lo stesso andamento per l'overshoot che conferma l'ipotesi di dipendenza dai componenti interni all'amplificatore escludendo elementi esterni come capacità parassite. È possibile inoltre notare una dispersione elevata delle misure. Questo si suppone dovuto ad errori di imprecisione nella presa dati (es cursori non perfettamente allineati o resistenze con tolleranze elevate).

Tramite i dati raccolti è possibile ottenere una stima della banda di  $9.4 \pm 0.9$  MHz. Il valore di riferimento di 8Mhz entra in  $1.5 \sigma$  dalla stima.

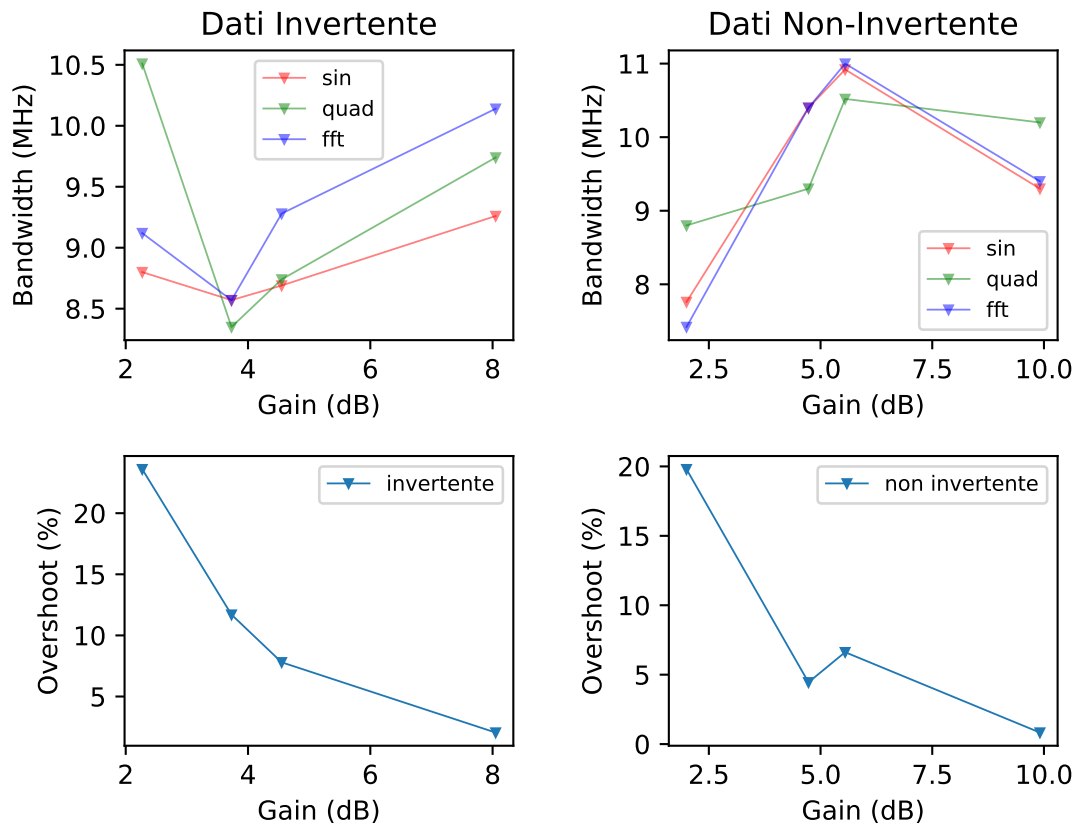


Figura 20: Valori di banda calcolati tramite le tre configurazioni per Invertente e Non-inv, ([link dati](#))

## 7.2 AD848

Il nostro obiettivo finale è la caratterizzazione degli amplificatori AD848 che verranno tenuti nella configurazione finale della scheda analogica. Questo amplificatore a differenza dell'OP27 è garantito nei [datasheet](#) con un gain minimo di 5. In questa sezione vogliamo eseguire una caratterizzazione della sua banda e indagare la zona a basso gain.

### 7.2.1 Oscillazioni a Gain BASSO

Montando l'amplificatore nella configurazione non invertente, come visto in precedenza, abbiamo un gain di

$$\frac{1}{\beta} = \frac{R_1 + R_2}{R_1}$$

per cui utilizzando resistenze  $R_1 = 2.2k\Omega$  e  $R_2 = 2.2k\Omega$  si ottiene un gain di 2 che risulta essere inferiore alla soglia di 5 per il funzionamento prescritto nel manuale.

É possibile facilmente notare che in questa zona l'amplificatore non si comporta in maniera corretta; infatti, guardando il margine di frequenza, si nota che questo tende a  $0^\circ$  nella zona sotto 5 Gain ( $\sim 13dB$ ) rendendo il segnale di uscita totalmente inutilizzabile.

Nella configurazione invertente invece selezionando le stesse resistenze  $R_1 = 2.2k\Omega$  e  $R_2 = 2.2k\Omega$  è possibile ottenere un gain di -1 con il seguente andamento:

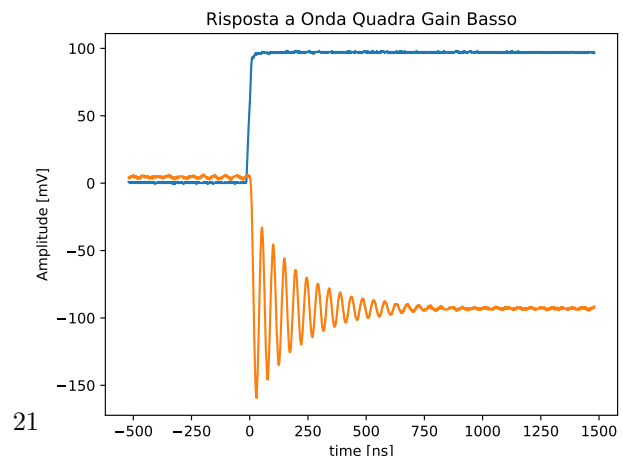


Figura 23: AD848 Configurazione invertente a gain basso, ([link dati](#))

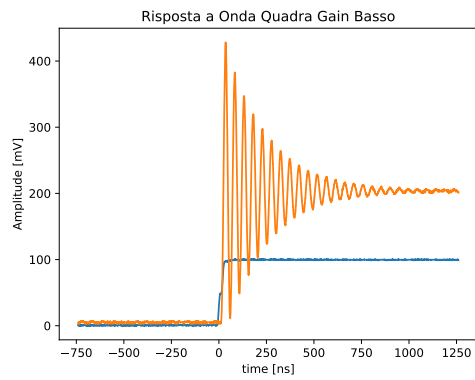


Figura 21: Andamento oscillante a gain basso, ([link dati](#))

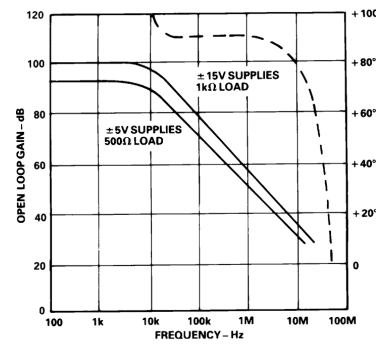


Figure 14. Open Loop Gain and Phase Margin vs. Frequency (AD848)

Figura 22: Margine di fase AD848

Come nel caso di configurazione non invertente il margine di fase risulta essere troppo basso e si ottiene instabilità evidente.

### 7.2.2 Gain elevato

A gain superiore a 5 sono state prese misure sia in configurazione invertente che configurazione non invertente utilizzando un'onda quadra e resistenze  $R_1 = 2.2k\Omega$  e  $R_2 = \{14.7k\Omega, 31.5k\Omega, 48.5k\Omega\}$

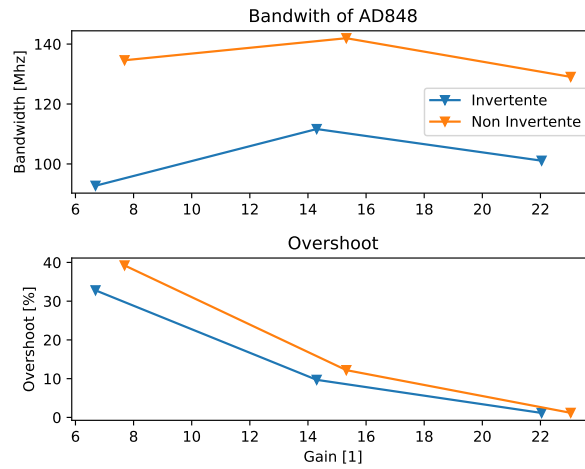


Figura 24: Andamento finale AD848

Il valor medio di bandwidth calcolato è  $\mathbf{BW = 118.5 \pm 17.9 \text{ MHz}}$ .

### 7.2.3 Conclusioni

La presenza del doppio polo porta a un margine di fase molto ridotto a Gain basso ed instabilità del segnale che lo rende inutilizzabile. A Gain superiore alla soglia consigliata si ottiene un andamento consono e una bandwidth di  $\mathbf{BW = 118.5 \pm 17.9 \text{ MHz}}$  in accordo con i datasheet. Questo amplificatore permette guadagni molto più elevati dell' OP27 che permette di amplificare maggiormente il segnale del rilevatore a nostro vantaggio.

## 7.3 SiPm

La parte finale della caratterizzazione della scheda richiede lo studio del rivelatore al suo interno, ovvero un SiPm (Silicon Photo Multiplier).

Questo rivelatore presenta al suo interno il sistema composto da SPAD e resistenze di quenching che permette la creazione di valanghe di elettroni controllate all'interno del rivelatore fino ad ottenere una risoluzione al singolo fotone.

### 7.3.1 Setup

Siccome la tensione generata dal SiPm è troppo bassa per la risoluzione dell'ADC è necessario applicare gli stadi di amplificazione studiati nei punti seguenti come mostrato in figura:

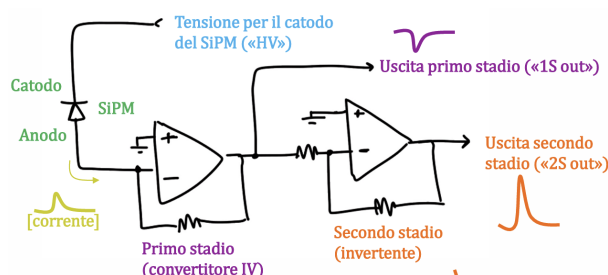


Figura 25: Stadi di amplificazione

Siccome i tempi di esistenza del segnale sono molto bassi, nell'ordine dei 100-200ns, risulta difficile ottenere una misurazione accurata con l'ADC che è limitato a campionamenti nell'ordine dei 200ns. Per questo è stato usato un sistema di Peak Hold:

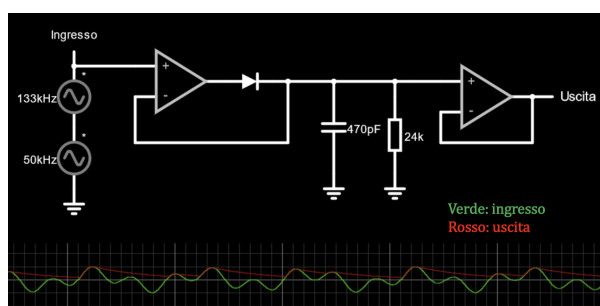


Figura 26: Sistema di Peak Hold

All'uscita del peak hold la tensione viene mantenuta per un tempo sufficiente per l'acquisizione dell'ADC.

Come mostrato in figura 25, il rivelatore che si comporta come diodo è inserito in polarizzazione inversa nella scheda in maniera tale da sfruttarne la tensione di breakdown. Per comprendere la polarizzazione, è stata impostata inizialmente sull'alimentatore una tensione bassa e corrente limitata per controllare la polarizzazione corretta. Avendo inserito nei suoi pin la fonte luminosa a LED, è stata sigillata la zona contenente fonte luminosa e rivelatore. Successivamente, la tensione fornita al SiPm è stata portata a +3V di Overvoltage per permettere l'avvento della valanga di elettroni nello SPAD.

Fornendo al LED una tensione impulsata di 100 ns a circa 3.4 V è stato possibile osservare la rilevazione dei fotoni:

E evidente la presenza di diverse zone con elevata densità di rilevazioni che corrispondono a 1 fotone, 2 fotoni, 3 fotoni etc...

### 7.3.2 Dark Count Rate

Siccome gli elettroni nello spad sono soggetti ad agitazione termica, è possibile che un elettrone riesca casualmente a liberarsi anche senza la presenza di un fotone, portando a una valanga non desiderata. Questo è noto come DCR (Dark Count Rate).



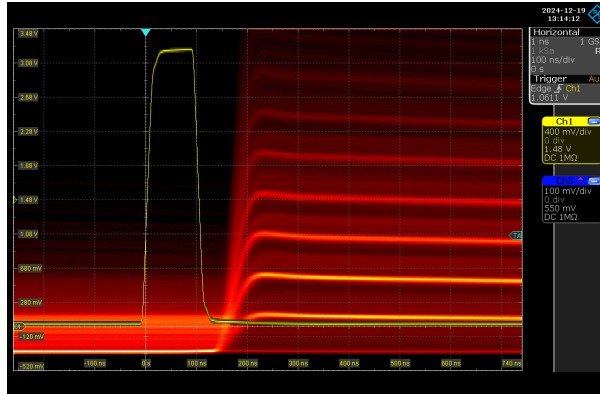


Figura 27: Prima visualizzazione dei segnali con persistence di 2 sec

Per caratterizzare il SiPm è utile avere una stima di questo fenomeno. Per questo è stato impostato l'oscilloscopio in modalità di conteggio dei segnali in arrivo. Impostata una soglia di trigger di circa mezzo fotone è stata fatta partire la misura di tempo per 40000 segnali a tensioni di overvoltage differenti.

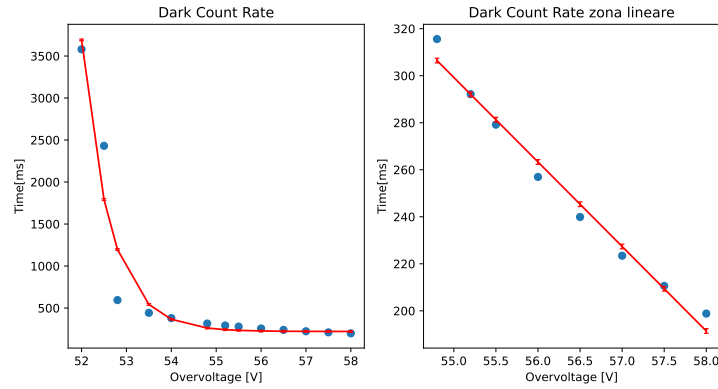


Figura 28: Tempi per conteggio di 40000 eventi, ([link dati](#))

Nella zona prossima alla tensione di *break down* il rate risulta molto basso, si suppone dovuto alla mancanza di campo elettrico fornito agli elettroni che non sono in grado in certi casi di produrre la valanga. Per tensioni prossime a quelle di *overvoltage* consigliate, ovvero  $V_{OV} = 54.9V$ , il rate si comporta in maniera approssimativamente lineare. Siccome il rate cambia di circa 31 ms/V possiamo approssimare questo rate ad un valore costante di circa **DCR =  $158 \pm 38$  KHz**. Questo valore risulta ragionevole con i valori inclusi nel manuale.

## 8 Risultati

Inserisci qui i risultati.

## 9 Conclusioni

Inserisci qui le conclusioni.