UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Esperimentazione di Elettronica

Andrea Boldetti

31 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

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 μ - 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,

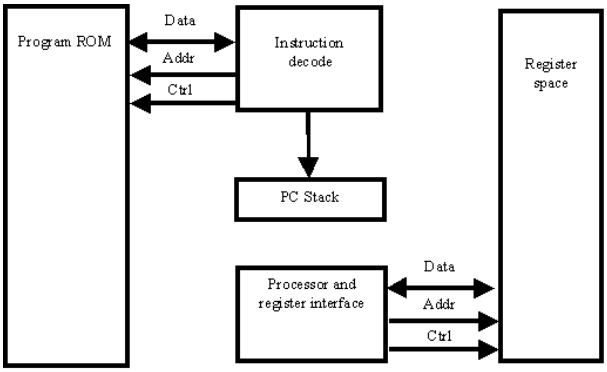


Figure 1-5 Harvard architecture block diagram

Figura 1: Struttura interna del microcontrollore

aumentando la velocità di esecuzione. Un esempio del perchè 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 spceifici 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 del 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 maaschere, 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 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 chiamati 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 sicrono 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

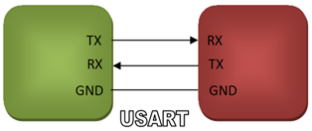


Figura 2: Collegamento tra microcontrollore e PC

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 usare 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.

```
void ESPE_USART_char_start(void){

    if ( USART3 ->ISR & USART_ISR_RXNE_RXFNE){ // Se il buffer è pieno
        if ( USART3 -> RDR == char_trigger){ // Se il carattere è uguale al trigger
            flag_Trigger_EN = 1;

        }
    }
}
```

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.

```
#define len_vec 100
uint_8 vec_dati[len_vec];
uint_8 len = sizeof(uint_8)/sizeof(char)*len_vec;
char *str = vec_dati;
uint_8 indice = 0;


void ESPE_USART_send_data(void){
    if( USART3 ->ISR & USART_ISR_TC){ // Se il buffer è vuoto
                if (indice < len){
                        USART3->TDR = *(str+indice); // Invia il dato
                        indice++;
                }else{
                        if(indice == len){
                                USART3 -> TDR = '\r'; // Invia il carattere di fine riga
                                indice = 0;
                        }
                }
        }
}
```

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.

```
void ESPE_USART_invert_mode(void){
    if(USART3 -> CR1 & USART_CR1_RXNEIE){
        USART3 -> CR1 &= ~USART_CR1_RXNEIE;
        USART3 -> CR1 |= USART_CR1_TCIE;
    }else if(USART3 -> CR1 & USART_CR1_TCIE){
        USART3 -> CR1 |= USART_CR1_RXNEIE;
        USART3 -> CR1 &= ~USART_CR1_TCIE;
    }
}
```

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. Seppur, con la *differential mode*, si possono prendere segnali positivi e negativi, per semplificare il processo, si userà l'ADC in modalità single-ended e quindi si prenderanno solo segnali positivi.

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.

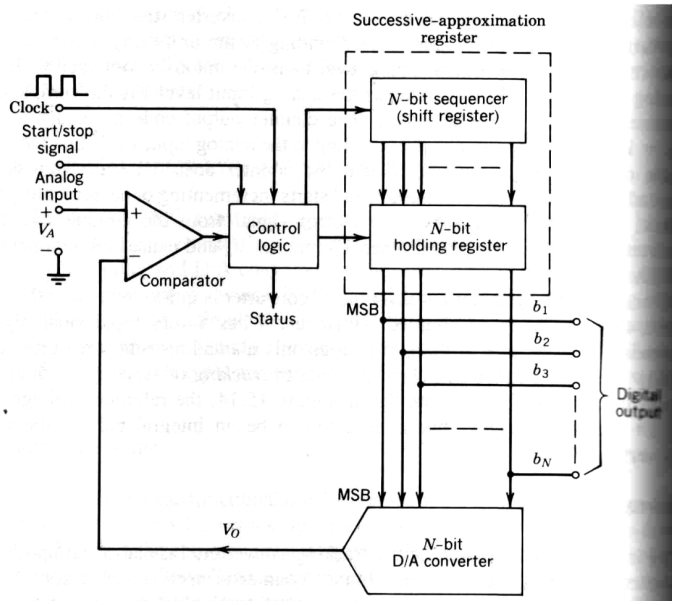


Figura 3: Schema di funzionamento dell'ADC

L'ADC ad approssimazioni successive per funzionare sfrutta un comparatore e un metodo di ricerca binaria. Attraverso una coversione 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 compartore, 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
```

```
        ADC3->SQR1 |= 0 <<ADC_SQR1_L_Pos;// ti dice quante misure deve prendere (n+1)
        ADC3->SQR1 |= 0 << ADC_SQR1_SQ1_Pos; // ti dice qual è la prima misura da fare

        ADC3->PCSEL |= ADC_PCSEL_PCSEL_0;//segna quali sono i canali in lettura per
                                         //velocità massima

        ADC3 -> CR &= ~ADC_CR_DEEPPWD_Pos;//Deep power down state
                                         //(se attivo non overclocka)
        ADC3 -> CR |= 1 << ADC_CR_ADVREGEN_Pos;//Voltage regulator activated

        ADC3 -> CR &= ~ADC_CR_ADCALDIF_Pos;//seleziona modalità differenziata
                                           //di calibrazione (a 0)
        ADC3 -> CR |= 1<< ADC_CR_ADCALLIN_Pos;//seleziona la modalità lineare
                                           //di calibrazione (a 1)
        ADC3 -> CR &= ~ADC_CR_ADEN_Pos;//Controlliamo che l'ADC non sia acceso e che
                                           //il bit sia stato resettato
        ADC3 -> CR |= 1<< ADC_CR_ADCAL_Pos;// Inizia la calibrazione

        while( ADC3->CR & ADC_CR_ADCAL ){
            //Aspetti che la calibrazione sia finita, il bit viene cambiato dall'hardware
        }

        ADC3->ISR &= ~ADC_ISR_ADRDY_Pos;//Controlli che il bit per l'inizio della
                                         //presa dati sia a 0
        ADC3->CR |= 1<<ADC_CR_ADEN_Pos;//Attiviamo l'ADC (non la presa dati)
        while( !(ADC3->ISR & ADC_ISR_ADRDY)){
            //Aspettiamo che sia setuppato correttamente
        }

        ADC3 -> IER |= ADC_IER_EOCIE;//Attiviamo l'interrupt

        ADC3 -> SMPR1 |= 0<<ADC_SMPR1_SMP0_Pos;
        //Possiamo aggiungere un ritardo di (n = 0) cicli prima della lettura

    }
```

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:

```

    #define len_vec 100
    uint_16 vec[len_vec];
    uint_8 counter_ADC = 0;
    uint_8 conversion_flag = 0;


```

```
8     void ESPE_ADC_data(void){
9         if( counter_ADC < len_vec){
10            vec[counter_ADC] = ADC3->DR;
11            counter_ADC++;
12        }else{
13            conversion_flag = 1;
14            ADC3-> CR &= ~ADC_CR_ADSTART;
15        }
16    }
17
```

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.

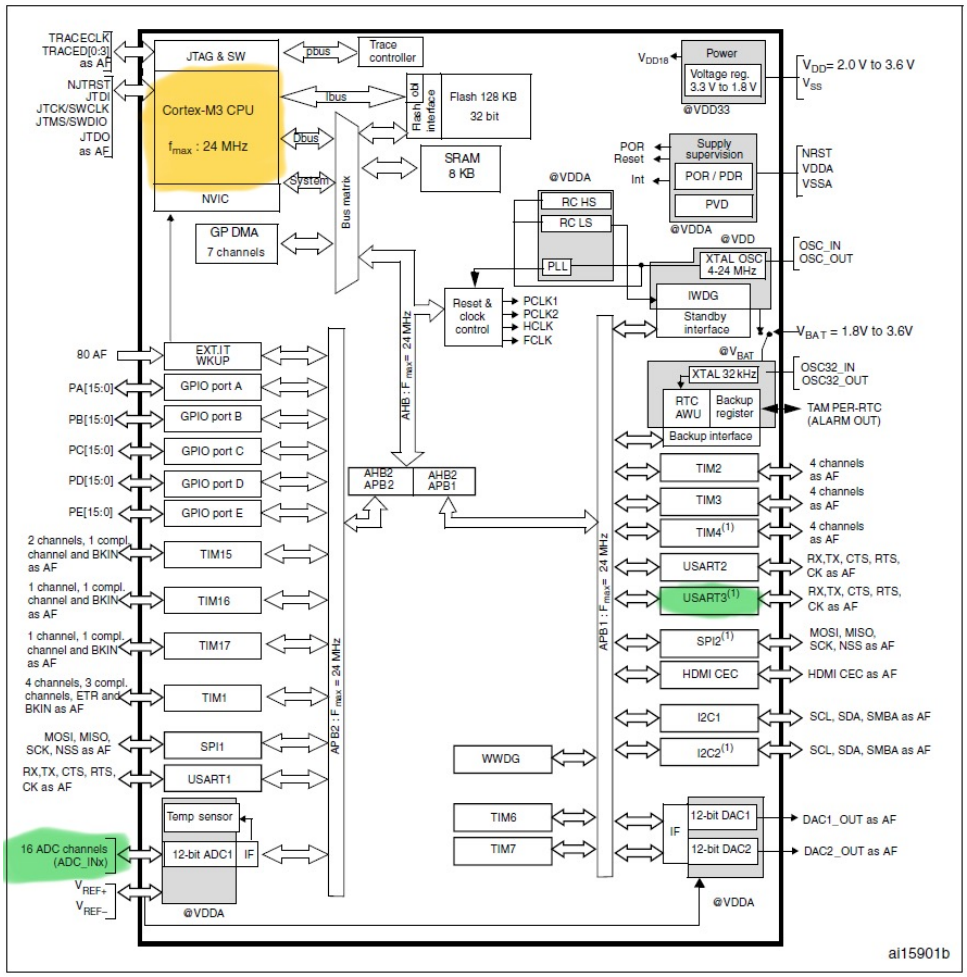


Figura 4: Struttura interna del microcontrollore

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.

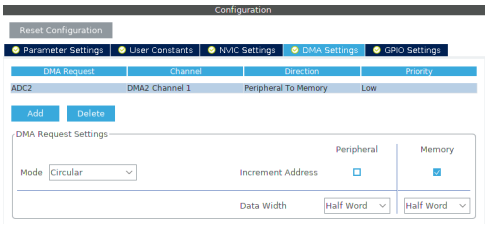


Figura 5: Configurazione del DMA

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

```
1    DMA1_Stream0 -> PAR = (uint32_t) &ADC3->DR; //Collegare il DMA al registro
2                                                //che contiene il dato
3    DMA1_Stream0 -> M0AR = (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
```

Notiamo come, anche se il DMA è attivo, ciò non va a modificare il funzionamento degli interrupt. Anche se il DMA controlla il flusso dei dati, noi possiamo comunque accedere a qualsiasi dato (misurato o trasmesso) in qualsiasi momento.

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.

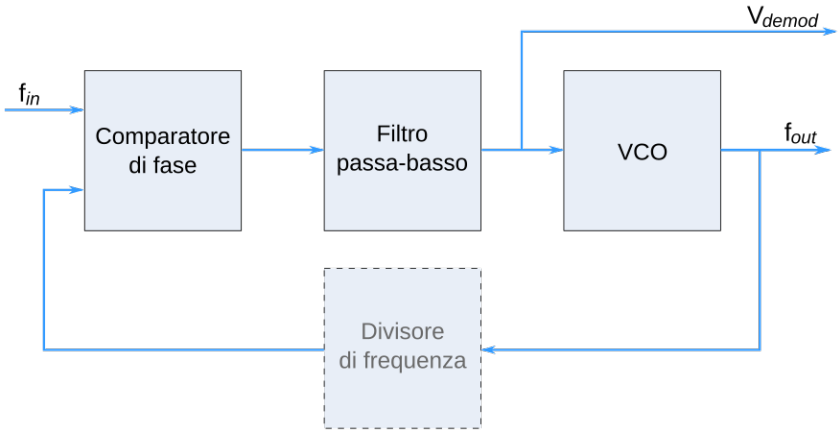


Figura 6: Schema di funzionamento del Timer

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  }
```

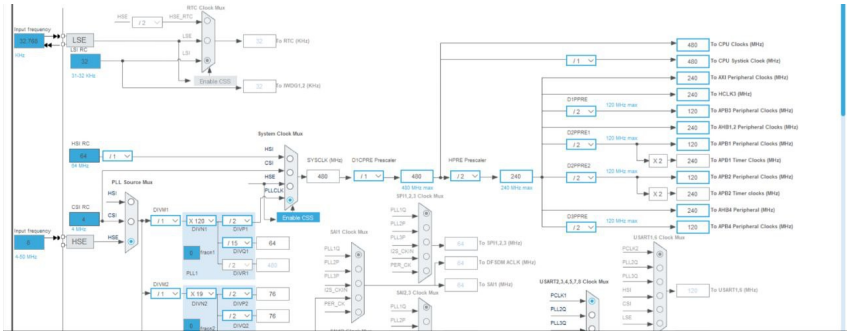


Figura 7: Configurazione del Timer

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 Digital Analog Converter (DAC) e Comparatore

Il DAC e il Comparatore, seppur 2 periferiche diverse, sono trattate insieme in quanto sono usate come trigger: il DAC per generare un segnale analogico e il comparatore per confrontare il segnale generato con un segnale di riferimento, così facendo si può avere un meccanismo di Trigger molto preciso e veloce.

6.1 Funzionamento DAC

Il DAC del microcontrollore è una periferica a 12 bit che permette di generare un segnale analogico a partire da un segnale digitale.
Una grande problematica del DAC è la gestione dell'errore. Infatti, se si vuole generare un segnale analogico ad un determinato voltaggio a partire da un numero binario, a prescindere dalla struttura, ci sarà sempre un errore dovuto alla lunghezza del numero binario. A questo errore si aggiungono tutti gli errori strumentali delle componenti del DAC.

Per questo motivo, la struttura del DAC è stata progettata in modo da minimizzare l'errore. Questa struttura è detta *R-2R* e consiste in una rete di resistenze dal valore di R e $2R$.

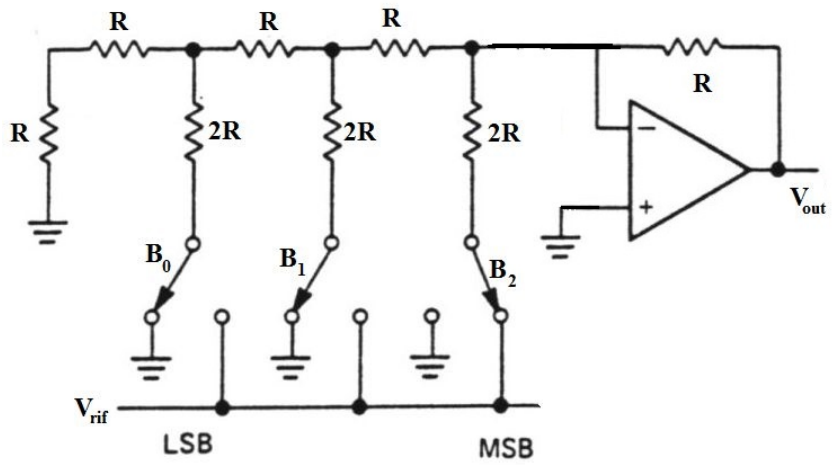


Figura 8: Struttura interna del DAC

La struttura è molto semplice: se il bit è 1, il circuito si chiude e il voltaggio in uscita è V_{ref}, se il bit è 0, il circuito si apre e il voltaggio in uscita è 0. La configurazione delle Resistenze permette di dimezzare il voltaggio in uscita ad ogni iterazione dello schema. E, dato che alla fine c'è un amplificatore operazionale, il voltaggio in uscita sarà la somma dei voltaggi generati da ogni bit, moltiplicato per un fattore dell'Amplificatore.

Così facendo, usando solo 2 tipi di resistenza, un amplificatore operazionale e dei transistore come interruttori, siamo in grado di creare un segnale analogico.

6.2 Funzionamento Comparatore

Il Comparatore non è altro che un amplificatore operazionale con un'uscita che non può essere Negativa. Questo significa che, se il voltaggio in ingresso è maggiore di un voltaggio di riferimento, l'uscita sarà alta, altrimenti sarà bassa.

Un comportamento particolare del comparatore è l'isteresi. L'isteresi è un fenomeno che si verifica quando il segnale in ingresso è vicino al voltaggio di riferimento. In questo caso, l'uscita del comparatore non è stabile e può oscillare tra alto e basso. Per evitare questo fenomeno, si può aggiungere un isteresi al comparatore.
L'isteresi si aggiunge mettendo un partitore di tensione in retroazione positiva. Così facendo ci sarà una differenza di voltaggio tra quando il comparatore sale e quando scende.

Questo comportamento ci permetterà di usare il comparatore non solo come singolo trigger ma come doppio trigger.

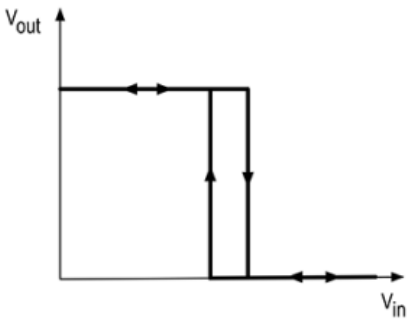


Figura 9: Grafico Voltaggio Amplificatore con Isteresi

6.3 Programmazione

La programmazione del DAC e del Comparatore si fa quasi totalmente da STM32CubeMX. La configurazione consiste nel collegare il DAC al comparatore, selezionre il valore di isteresi del Comparatore. Una volta configurato, nel codice basta aggiungere la funzione che permette di attivare il DAC e il Comparatore.

```
    COMP2->CFGR |= COMP_CFGRx_EN; // Attiva il Comparatore
    DAC1 -> DHR12R1 = 1000;        // soglia di trigger
        DAC1 -> SWTRIGR |= DAC_SWTRIGR_SWTRIG1; // Abilita il software trigger
        DAC1 -> CR |= DAC_CR_EN1; // Attiva il DAC
```

E, per andare a controllare lo stato del comparatore, si può usare la funzione:

```
if( COMP12 -> SR & COMP_SR_C2VAL){ // Se il comparatore è attivo
    //Fai qualcosa
}
```

7 Oscilloscopio

Unendo tutte le funzionalità spiegate in precedenza, siamo in grado di usare il microcontrollore come oscilloscopio. Per farlo, però, serve un meccanismo di trigger che ci permetta di selezionare solo i segnali voluti.

7.1 Trigger

Sono stati sperimentati diversi tipi di trigger con diverse periferiche. In particolare, i trigger utilizzati sono:

- Trigger a singola soglia software: Un trigger che usando il software controlla se il valore misurato è maggiore di una soglia
- Trigger a singola soglia hardware: Un trigger che usando un comparatore hardware controlla se il valore misurato è maggiore di una soglia
- Trigger in salita software: Un trigger che controlla prima se il valore è sotto una soglia e poi se un valore successivo è sopra un'altra soglia. Questo trigger permette di evitare di triggerare in discesa e quindi quando il segnale è già passato.
- Trigger in salita hardware: Un trigger basato sul controllo precedente ma fatto con dei comparatori al posto che dal software.

7.1.1 Trigger a singola soglia software

Questo tipo di trigger corrisponde alla versione più semplice ed intuitiva di trigger. Il microcontrollore per controllare il valore deve fare diverse operazioni e questo rallenta notevolmente il codice. Inoltre, questo tipo di trigger non permette di selezionare con precisione il segnale desiderato, in quanto per ogni soglia scelta diversa dal picco massimo del segnale, il segnale potrebbe triggerare in 2 momenti diversi.

```
uint8_t flag_Trigger_EN = 0;
uint8_t flag_Triggered = 0;
void ESPE_DMA_Trigger(void){
    if(!flag_Triggered && flag_Trigger_EN){ // Se non è già stato triggerato
                                            //e è arrivato il segnale di attivazione
                                            //del trigger
        if( ADC3 -> DR > Trigger_Value){ // Se il valore è maggiore della soglia
            flag_Triggered = 1; // Triggerato
            flag_Trigger_EN = 0; //Reset Flag
        }
    }
}
```

7.1.2 Trigger in salita software

Il trigger in salita è una diretta evoluzione del trigger a singola soglia. Infatti, la presenza di 2 soglie di riferimento che agiscono in momenti diversi del segnale, permette di avere un trigger più preciso e affidabile. Questo trigger, però, è molto più complesso da implementare e richiede un controllo molto più preciso del segnale. Inoltre, seppur di poco rallenta il codice in quanto il numero di operazioni aumenta.

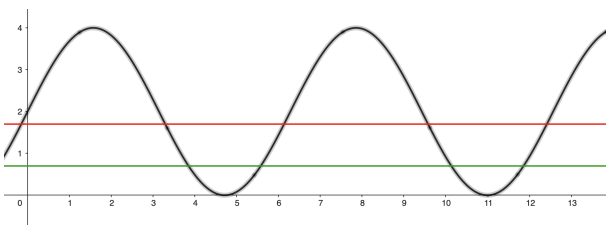


Figura 10: Trigger in salita

Come si vede dalla figura 10, se si controlla prima se il segnale è sotto la linea verde, o *Soglia di Pretrigger* e poi si controlla se il segnale è sopra la linea rossa, o *Soglia di Trigger*, si può risolvere il problema di ambiguità che c'era nel trigger a singola soglia. Questo processo rende molto più costante il trigger e permette di selezionare con precisione il segnale desiderato. Invertendo il processo, si ottiene il trigger in discesa, il quale permette di selezionare il punto che è stato scartato

in questo caso.

A livello di programmazione, questo codice è leggermente più complesso del precedente, ma ci permette di gestire i dati con una precisione migliore. Infatti con questo codice possiamo sceglere anche di prenderer i dati prima del trigger, scegliendo con precisione il punto in cui iniziare a prendere i dati.

```
uint8_t flag_Trigger_EN = 0;
uint8_t flag_Triggered = 0;
uint8_t flag_Pretriggered = 0;
int index = 0;
int index_stop = 0;


void ESPE_DMA_Trigger_Pretrigger(void){
    if(!flag_Triggered && flag_Trigger_EN){ // Se non è già stato
                                            //triggerato e è arrivato
                                            //il segnale di attivazione
                                            //del trigger
        if( flag_Pretriggered){ // Se il segnale è stato pretriggerato
            if( ADC3 -> DR > Trigger_Value){ // Se il valore è maggiore della
                                             //soglia di trigger
                flag_Triggered = 1; // Triggerato
                flag_Trigger_EN = 0; //Resetta i Flag
                flag_Pretriggered = 0;
                index = vec_len - DMA1_Stream0 -> NDTR + data_len +1
                index_stop = vec_len - (index)%vec_len; //Calcola l'indice di stop

            }
            return;
        }
        if( ADC3 -> DR < Pretrigger_Value){ // Se il valore è minore della soglia
                                            //di pretrigger
            flag_Pretriggered = 1; // Pretriggerato
        }
    }
}
```

7.1.3 Trigger hardware

Entrambi i trigger Hardware, non sono altro che variazioni dei trigger software a cui, al posto di controllare la soglia con un if, si usa un valore all'interno del registro del microcontrollore. Questo passaggio permette di saltare la conversione del valore fatta dall'ADC e di avere solo un valore booleano.

Per quanto riguarda il trigger hadware a singola soglia:

```
uint8_t flag_Trigger_EN = 0;
uint8_t flag_Triggered = 0;

void ESPE_DMA_COMP_Trigger(void){
if(!flag_Triggered && flag_Trigger_EN ){// Se non è già stato triggerato e è arrivato
                                        //il segnale di attivazione del trigger
        if( COMP12 -> SR & COMP_SR_C2VAL ){ // Se il comparatore ha triggerato
            flag_Triggered = 1; // Triggerato
            flag_Trigger_EN = 0; //Reset Flag
        }
    }
}
```

Il trigger in salita Hardware, è possibile farlo con un singolo comparatore al posto di 2 usando l'Isteresi. Questo comportamento anomalo dei comparatori reali, permette una sottile divisione dal voltaggio di quando il comparatore è triggerato e quando non lo è. Usando questo comportamento come pretrigger,

si evita di aggiungere cavi per un altro comparatore. Cosa che si deve comunque fare nel caso in cui si voglia un pretrigger notevolmente più basso del trigger.

```
void ESPE_DMA_COMP_Trigger_Pretrigger(void){
if(!flag_Triggered && flag_Trigger_EN){
    if( flag_Pretriggered){
        if( COMP12 -> SR & COMP_SR_C2VAL){
            flag_Triggered = 1;
            flag_Trigger_EN = 0;
            flag_Pretriggered = 0;

            //index_stop = (DMA1_Stream0 ->NDTR + vec_len - data_len)%vec_len;
            index_stop = vec_len - (vec_len - DMA1_Stream0 -> NDTR + data_len +1)%vec_len;

        }
        return;
    }
    if( !(COMP12-> SR & COMP_SR_C2VAL) ){
        flag_Pretriggered = 1;
    }
}
}
```

Il lato negativo del trigger hardware è la quantità di cavi da collegare al microcontrollore. Infatti, per ogni comparatore che si vuole usare, c'è bisogno di 2 cavi aggiuntivi in parallelo con quelli collegati all'ADC. Se si sta lavorando ad alta frequenza o con tanti cavi, questo può causare una deformazione dell'onda.

Velocizzando tutti i processi al massimo, si ottiene un oscilloscopio in grado di misurare segnali fino a 30MHz.

8 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:

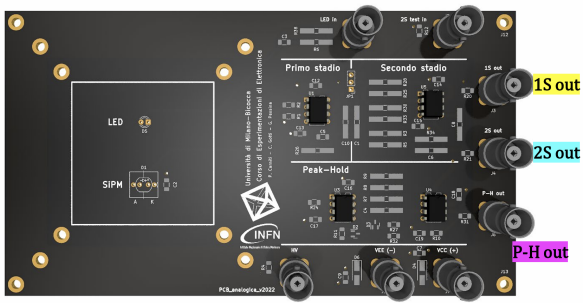
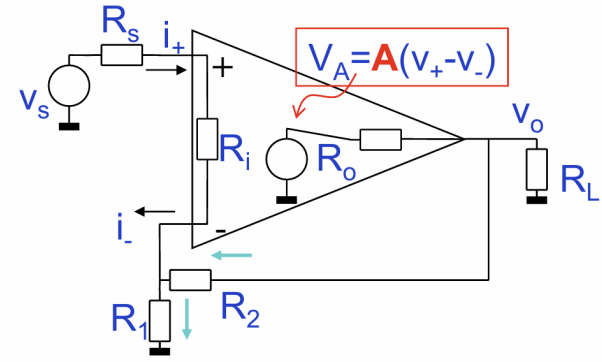


Figura 11: Scheda Analogica

Il nostro obbiettivo 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 $R_L \sim 1M\Omega$.

8.1 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.

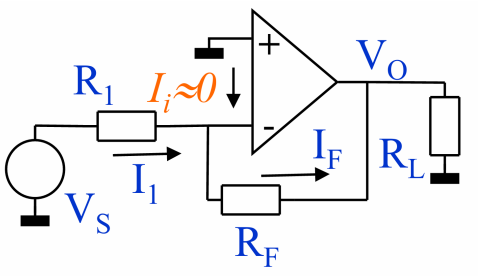
In particolare è segnata una **Bandwidth** di circa **8Mhz** che risulterà essere il dato di nostro interesse.

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

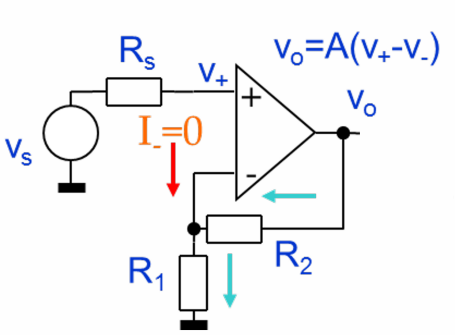
FEATURES

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 V_{OS}: 10 µV
CMRR: 126 dB at VCM of ±11 V
High open-loop gain: 1.8 million
Available in die form

Figura 12: Feature OP27



(a) configurazione invertente



(b) configurazione non invertente

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.

8.1.1 Configurazione Non Invertente

In questa configurazione abbiamo un guadagno di Gain $\frac{1}{\beta} = \frac{R_1+R_2}{R_1}$ nel caso di amplificatiore **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(ω) denota la dipendenza dalla frequenza. Per poter caratterizzare questo comportamento in frequenza vogliamo trovare la **banda** dell'amplificatore.

É 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: $\boldsymbol{Banda = \frac{\omega_F}{\beta} = 7.8Mhz}$.

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.

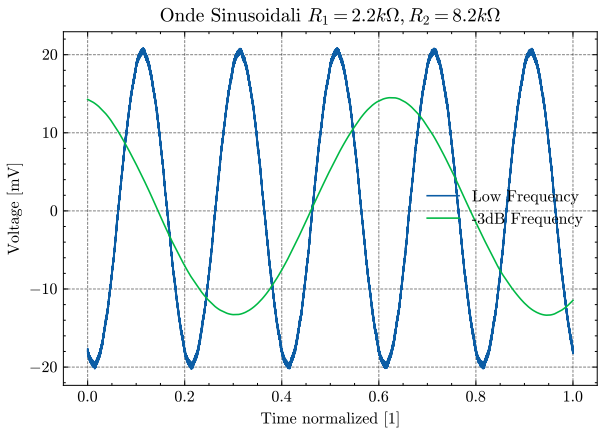


Figura 14: Segnali di uscita dall'amplificatore per frequenza a risposta piatta e a -3dB, (link dati)

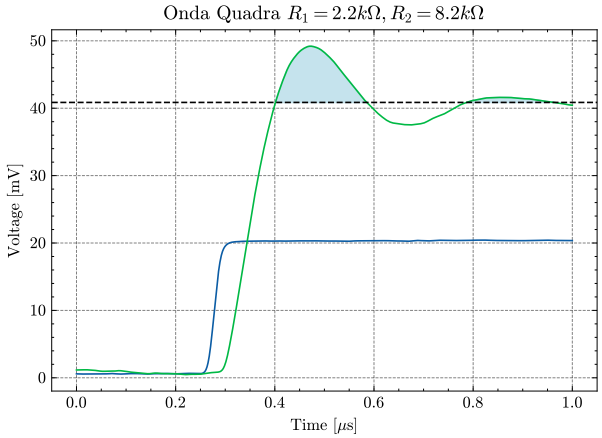


Figura 15: Risposta ad onda quadra, (link dati)

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

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

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

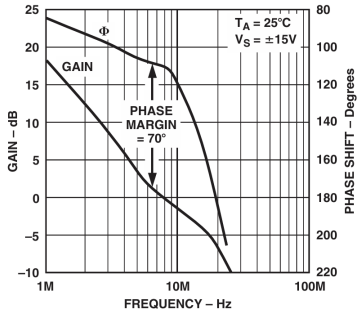
Notiamo come in risposta all'onda quadra, l'amplificatore presenti un'overshoot non trascurabile: circa il 20% rispetto al valore aspettato. Questo può essere dovuto a diversi fattori:

- 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

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

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



TPC 18. Gain, Phase Shift vs. Frequency

Figura 16: 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 18 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.

É 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**.

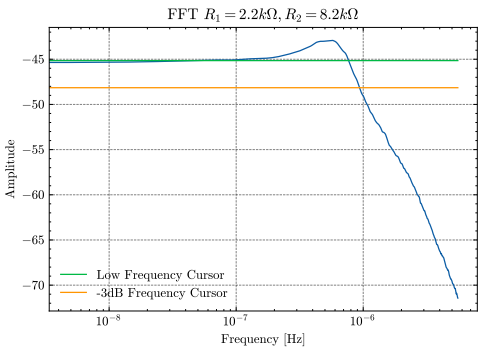


Figura 17: (link dati)

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:

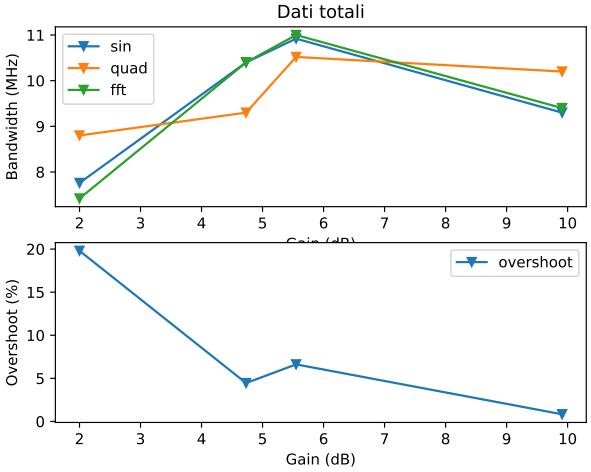


Figura 18: Rappresentazione grafica dell'andamento per configurazione Non-invertente

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 É 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** ~ 9.6**Mhz** che risulta in accordo entro un 20% con il valore dei datasheet.

8.1.2 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 $R_1 = R_2 = 2.2k\Omega$

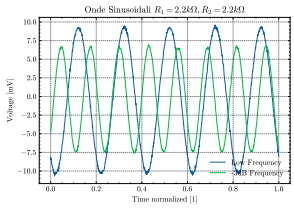


Figura 19: Analisi con segnale sinusiodale

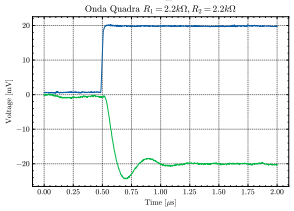


Figura 20: Analisi con onda quadra (overshoot $< 20\%$)

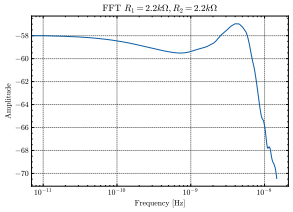


Figura 21: Analisi nel dominio delle frequenze

Da notare che l'uscita è invertita rispetto al segnale. Il resti dei dati è al link: (link dati)

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:

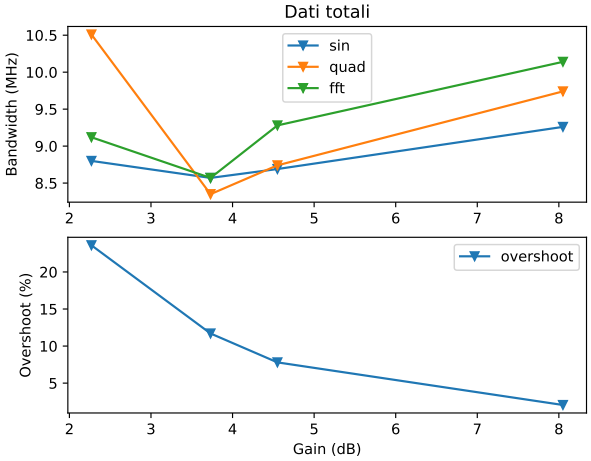


Figura 22: Rappresentazione grafica dell'andamento per configurazione Non-invertente

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 diminuizione dell'overshoot all'aumentare del gain nel circuito.

8.1.3 Conclusione

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

É 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. É 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 σ dalla stima.

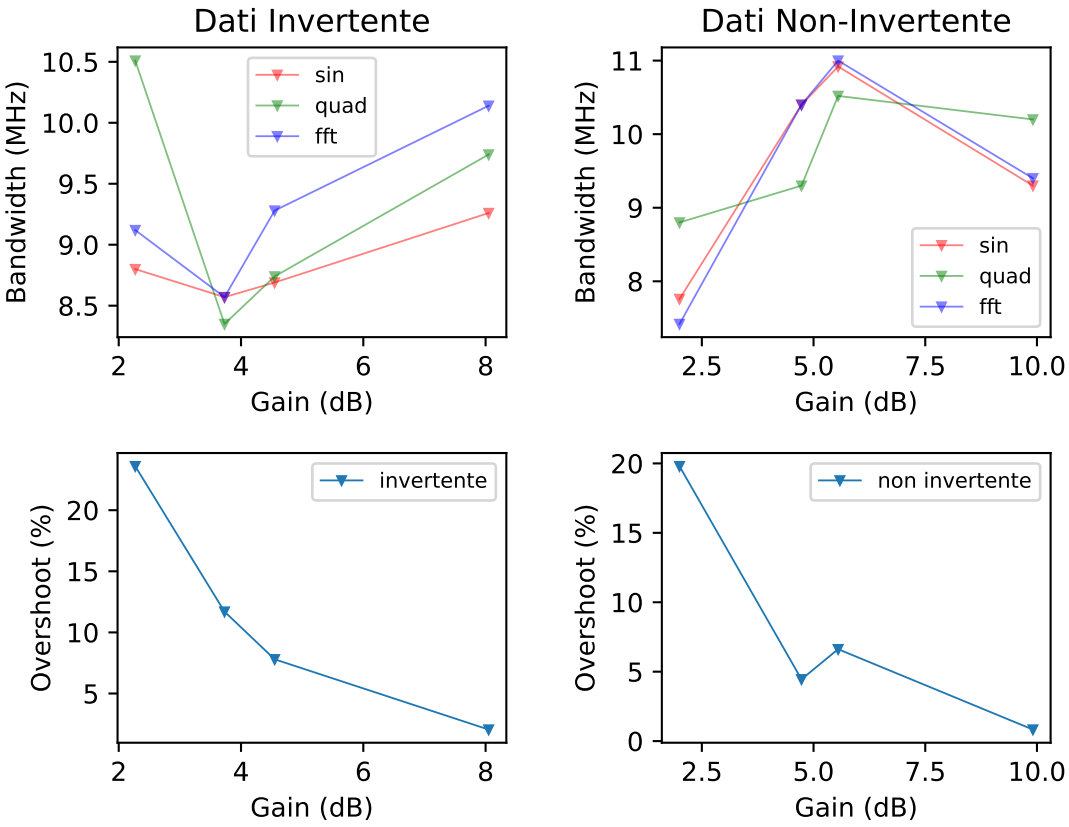


Figura 23: Valori di banda calcolati tramite le tre configurazioni per Invertente e Non-inv, (link dati)

8.2 AD848

Il nostro obbiettivo 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.

8.2.1 Oscillazioni a Gain BASSO

Montando l'amplificatore nella configurazione non invertende, 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° 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 ottnere un gain di -1 con il seguente andamento:

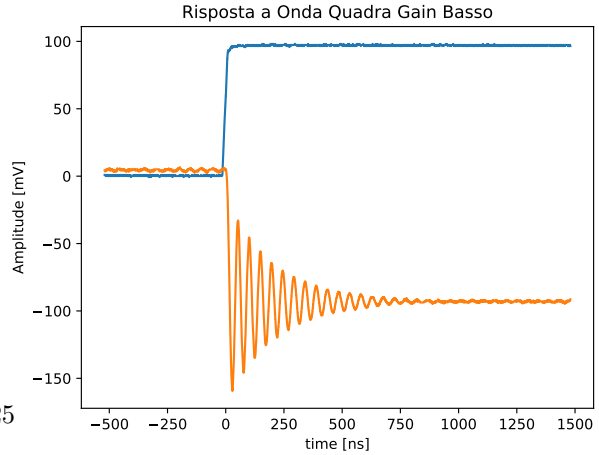


Figura 26: AD848 Configurazione invertente a gain basso, (link dati)

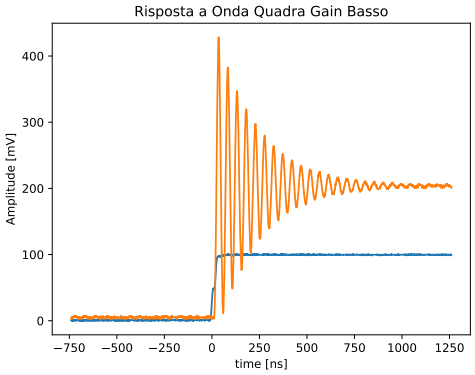


Figura 24: Andamento oscillante a gain basso, (link dati)

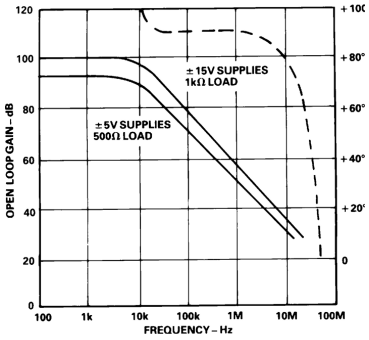


Figure 14. Open Loop Gain and Phase Margin vs. Frequency (AD849)

Figura 25: Margine di fase AD848

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

8.2.2 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 $R_2 = \{14.7k\Omega, 31.5k\Omega, 48.5k\Omega\}$

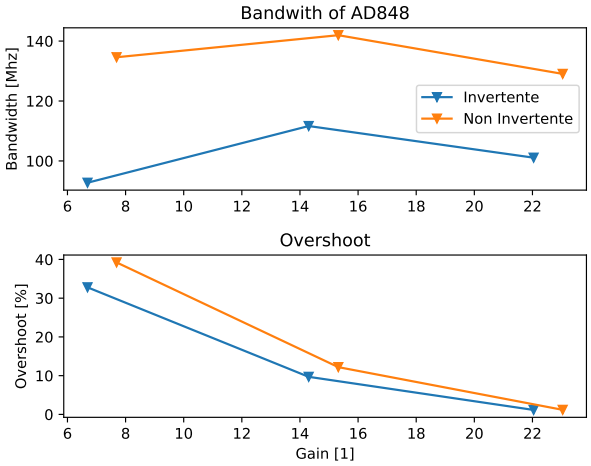


Figura 27: Andamento finale AD848

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

8.2.3 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 $\mathbf{BW = 118.5 \pm 17.9\ MHz}$ 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.

9 Fotomoltiplicatore

Una volta studiata la risposta in frequenza degli OpAmp nella scheda analogica, possiamo passare a studiare il funzionamento del fotomoltiplicatore e prendere una misura del voltaggio in uscita quando un led si accende.

9.1 Funzionamento LED

Un led è un diodo che emette luce quando attraversato da una corrente. Il diodo è in grado di fare ciò grazie ad una giunzione PN.
Una giunzione PN è una giunzione tra 2 materiali semicontrollori drogati in maniera opposta. Da un lato il materiale è drogato con atomi che hanno un elettrone debolmente legato, mentre l'altra parte è drogata con atomi che hanno un elettrone in meno.
Questa asimmetria della componente permette al diodo di avere un comportamento diverso a seconda della direzione della corrente.

Quindi, se la corrente è diretta in un senso, detta *polarizzazione diretta*, il diodo si comporta come un corto circuito. Se la corrente è diretta nell'altro senso, detta *polarizzazione inversa*, il diodo si comporta come un circuito aperto.

In particolare, se polarizzato direttamente, può capitare che degli elettroni si mischino con le lacune cancellandosi a vicenda. L'energia mancante viene emessa come fotone con la stessa energia dell'elettrone.
L'energia dell'elettrone, e quindi la frequenza del fotone emesso, dipendono dalla giunzione PN e dai materiali che la compongono. Questo significa che, cambiando i materiali, si può cambiare la frequenza della luce emessa.

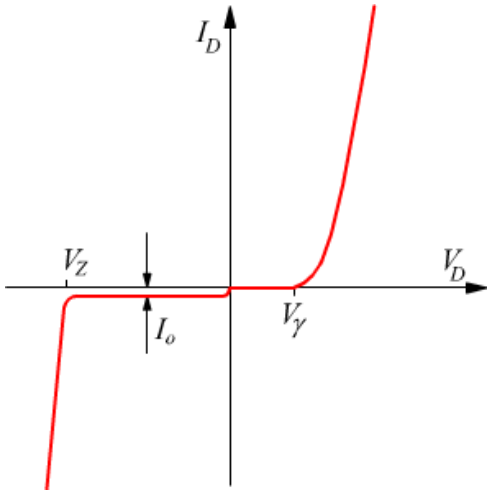


Figura 28: Legge di Shockley per il comportamento ideale del diodo

9.2 Funzionamento Fotomoltiplicatore

Con lo stesso principio del LED, ma in polarizzazione inversa, siamo in grado di creare un fotomoltiplicatore. Ovvero uno strumento che, se colpito da un fotone, emette un elettrone e quindi crea una corrente nel circuito.

Perchè questo succeda però, l'elettrone (e quindi la lacuna) creata con il fotone, deve avere abbastanza energia per uscire dal diodo e iniziare a scorrere nel circuito. Per questo serve aumentare il voltaggio. Una volta aumentato il voltaggio, però gli elettroni possono avere abbastanza energia per colpire altri atomi e creare altri elettroni, a creare una valanga. La tensione necessaria per creare questo fenomeno è detta *tensione di breakdown*.

Così facendo, possiamo collegare una tensione all'interno del circuito con l'urto di un fotone con il fotomoltiplicatore.

9.3 SiPm

Il rilevatore 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 rilevatore fino ad ottenere una risoluzione al singolo fotone.

9.3.1 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:

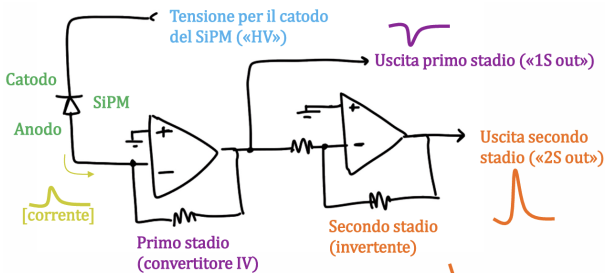


Figura 29: Stadi di amplificazione

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 300ns. Per questo è stato usato un sistema di Peak Hold:

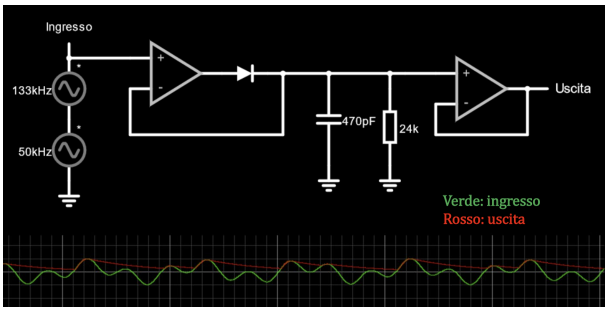


Figura 30: Sistema di Peak Hold

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

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 rilevatore. Successivamente, la tensione fornita al SiPm è stata portata a +3V di Overvoltage per permettere l'avvento della valanga di elettroni nello SPAD.

9.3.2 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).

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.

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 tenisoni 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

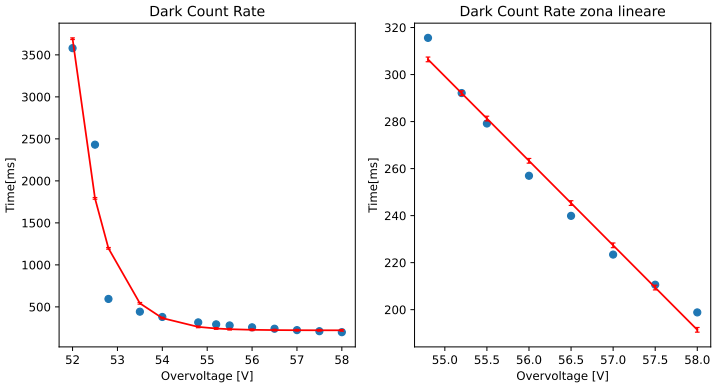


Figura 31: Tempi per conteggio di 40000 eventi, (link dati)

approssimare questo rate ad un valore costante di circa $\mathbf{DCR = 158 \pm 38KHz}$. Questo valore risulta ragionevole con i valori inclusi nel manuale.

10 Misure

Una volta studiato il funzionamento del fotomoltiplicatore, possiamo passare a prendere delle misure.

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

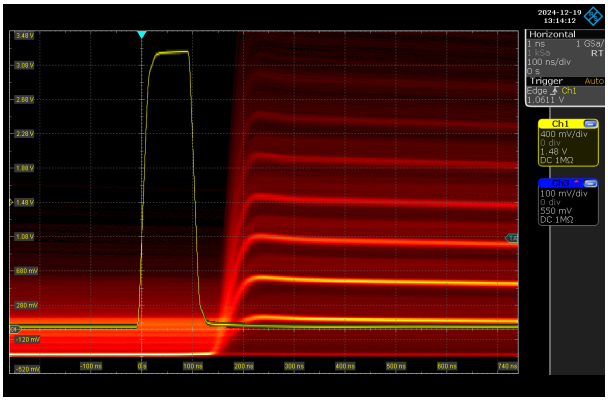


Figura 32: Prima visualizzazione dei segnali con persistance di 2 sec

E evidente la presenza di diverse zone con elevata densità di rilevazioni che corrispondono a un numero di fotoni crescente all'aumentare del voltaggio. Questo fenomeno è causato da 2 fotoni che arrivano in tempi molto ravvicinati e vengono rilevati come un unico evento.
Una volta collegato l'ADC e il comparatore all'uscita del Peak Hold, è stato possibile rilevare le onde:

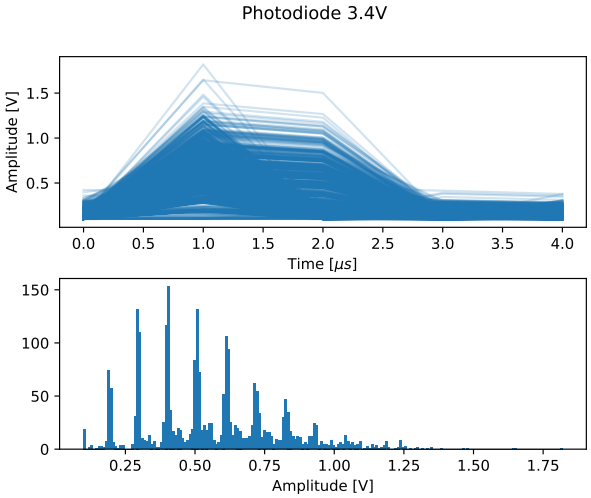
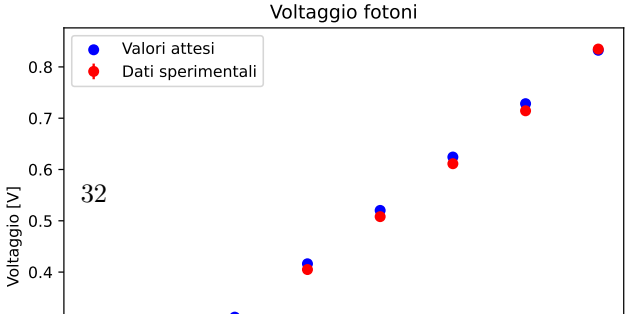


Figura 33: Segnale rilevato dall'ADC

Come si può notare, l'istogramma dei picchi è molto simile alla figura che si ha sull'oscilloscopio e i voltaggi sono molto simili.
Quello che ci aspettiamo è che i fotoni rilevati, siano tutti proporzionali alla misura del singolo fotone. Andando a calcolare i voltaggi che nell'istogramma corrispondevano ai picchi, siamo in grado di confrontare i valori ottenuti con i vari multipli del voltaggio di un singolo fotone.

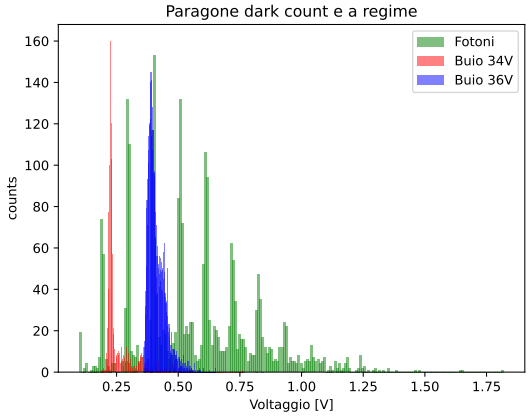
Come si può vedere dall grafico 34, i valori ottenuti sono molto vicini ai valori attesi, e quindi confermare l'ipotesi. Per quanto riguarda il rumore di sottofondo, si è notato che questo va-



ria proporzionalmente a diversi fattori. In particolare, è proporzionale al voltaggio di alimentazione del fotomoltiplicatore. Questo è dovuto al fatto che il rumore di sottofondo è causato da elettroni che vengono emessi dal catodo e che vengono accelerati verso l'anodo. Aumentando il voltaggio di alimentazione, si aumenta la velocità degli elettroni e quindi la probabilità che questi vengano rilevati.

Come si può vedere dal grafico **??**, gli istogrammi di Dark Count sono molto diversi con una differenza di solo $2V$ e questo è ragionevole pensare abbia un'influenza anche con il LED acceso.

Questa condizione motiva la presenza di più fotoni a tensioni più alte rispetto a quello che ci aspettavamo: ovvero di avere più misure di singolo fotone che altro.



11 Bibliografia

- Sito ufficiale di ARM
- Sito ufficiale di STM32CubeIDE
- Sito ufficiale di STM32CubeMX
- Sito ufficiale di STM32H7
- Sito ufficiale di STM32H7 Reference Manual
- Sito ufficiale di STM32H743VI Datasheet
- Github del corso