UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

"FILTRO PASSA BASSO CON FREQUENZA DI TAGLIO VARIABILE DIGITALMENTE"

Relatore: Prof. / Dott. Buso Simone

Laureando/a: Berzacola Nicola

ANNO ACCADEMICO 2021 – 2022

Data di laurea 22/09/2022

Sommario

1. INTRODUZIONE

Nel seguente elaborato verranno raccolti i ragionamenti, i principi di funzionamento e i passaggi fondamentali per la progettazione e realizzazione di un filtro analogico con guadagno e frequenza di taglio variabile digitalmente.

Nella prima parte verranno illustrate le specifiche tecniche che solitamente deve soddisfare un dispositivo di questo tipo per l'utilizzo proposto, in secondo luogo, ho espresso i ragionamenti volti all'acquisto dell'hardware con le alternative presenti sul mercato con i loro pro e contro.

Una volta scelto il cervello del progetto, il microcontrollore, verranno raccolte le funzioni principali, quelle più utili allo scopo e ne verrà brevemente illustrato il funzionamento e le modalità operative al modo di poter scegliere l'implementazione migliore per la realizzazione del filtro.

Verranno mostrati i programmi utilizzati, le configurazioni di sistema e gran parte del codice scritto. Passo dopo passo verrà spiegato prima il funzionamento e poi la programmazione per la comunicazione tra microcontrollore e potenziometri digitali e degli altri dispositivi utilizzati.

Dopo aver assemblato il prototipo del circuito completo farò una breve analisi sul funzionamento reale del filtro come la frequenza di taglio massima e minima raggiunta così da verificare che le scelte realizzate siano opportune.

Per concludere, mostro un secondo utilizzo del dispositivo diverso da quello pensato in origine mostrando le alternative e le potenzialità dello strumento, con alcuni consigli per poterne migliorare l'uso e le funzionalità.

Questo elaborato non vuole essere una stesura teorica, ma un modo per documentare la realizzazione del prototipo mostrando tutti i passaggi svolti in modo da poter essere replicato ed eventualmente migliorato in futuro.

Ritengo sia anche importante tenere conto che è stato realizzato in un laboratorio amatoriale all'interno della mia abitazione, gli strumenti che utilizzerò per la visualizzazione dei segnali e la misura della resistenza sono un multimetro digitale KDM-6000C e un oscilloscopio digitale Rigol DS1054z.

2. SPECIFICHE DI PROGETTO

Quello che si vuole realizzare è un filtro passa basso del primo ordine con la capacità di variare la frequenza di taglio e il guadagno digitalmente, tramite un microcontrollore. Lo scopo del dispositivo sarà quello di separare le basse frequenze per pilotare un subwoofer all'interno di un home theater o un impianto di alta fedeltà. Una volta accesa la macchina verrà caricata una funzione di setup che porta i parametri del filtro a valori standard preimpostati dal programmatore. Tramite l'utilizzo di un normale telecomando ad infrarossi l'utente potrà scegliere la corretta frequenza di taglio e guadagno del filtro, andando così a modificare l'esperienza di ascolto secondo i propri gusti musicali. Il range di funzionamento è tra i 50 e 120Hz mentre il guadagno può variare da 0,1 fino ad una amplificazione di 10. In questo modo l'utilizzatore potrà scegliere quanti bassi e l'intensità di questi ultimi per un ascolto ottimale all'interno della sua stanza. Il segnale in ingresso al filtro è l'uscita di un preamplificatore musicale con ampiezza massima di 2Vpp. Una caratteristica importante del filtro è che l'ingresso sia in alta impedenza, in questo modo il dispositivo non andrà a influenzare il funzionamento del preamplificatore lasciando il segnale in ingresso inalterato. La realizzazione del progetto ha come scopo primario la semplicità di utilizzo da parte dell'operatore tenendo i costi contenuti.

Il periodo corrente caratterizzato dalla crisi del silicio ha influenzato la scelta dei componenti più adatti, riducendo la lista dei possibili candidati a pochi pezzi disponibili nei magazzini dei principali distributori elettronici.

3. MATERIALE UTILIZZATO

Per la realizzazione di questo particolare tipo di filtro ho utilizzato una scheda dotata di microcontrollore ARM chiamata Blue Pill; non è tra gli ultimi modelli prodotti dall'azienda, ma, proprio per questo, è facilmente reperibile ad un prezzo e tempi di spedizione ragionevoli. Le dimensioni sono molto contenute, ma ha bisogno di un programmatore esterno; sempre per mantenere il prezzo accessibile, ho optato per il ST-LINK V2.

Il nome dei pin è scritto direttamente sulla scheda, ma, per conoscere la loro funzione nel dettaglio, è possibile consultare l'infografica della Figura 1. Sul lato sinistro e destro possiamo vedere tutti i pin di input/output e come essi possono essere impostati. Ad esempio, il pin A0 rappresentato col numero 10, può essere un input a 3.3V, output, ingresso per ADC0, un trigger esterno per il timer 2 oppure il pin per il wake-up del dispositivo dallo stand-by. Oltre al LED che indica lo stato del sistema è presente anche un piccolo LED pilotabile tramite il pin 13.

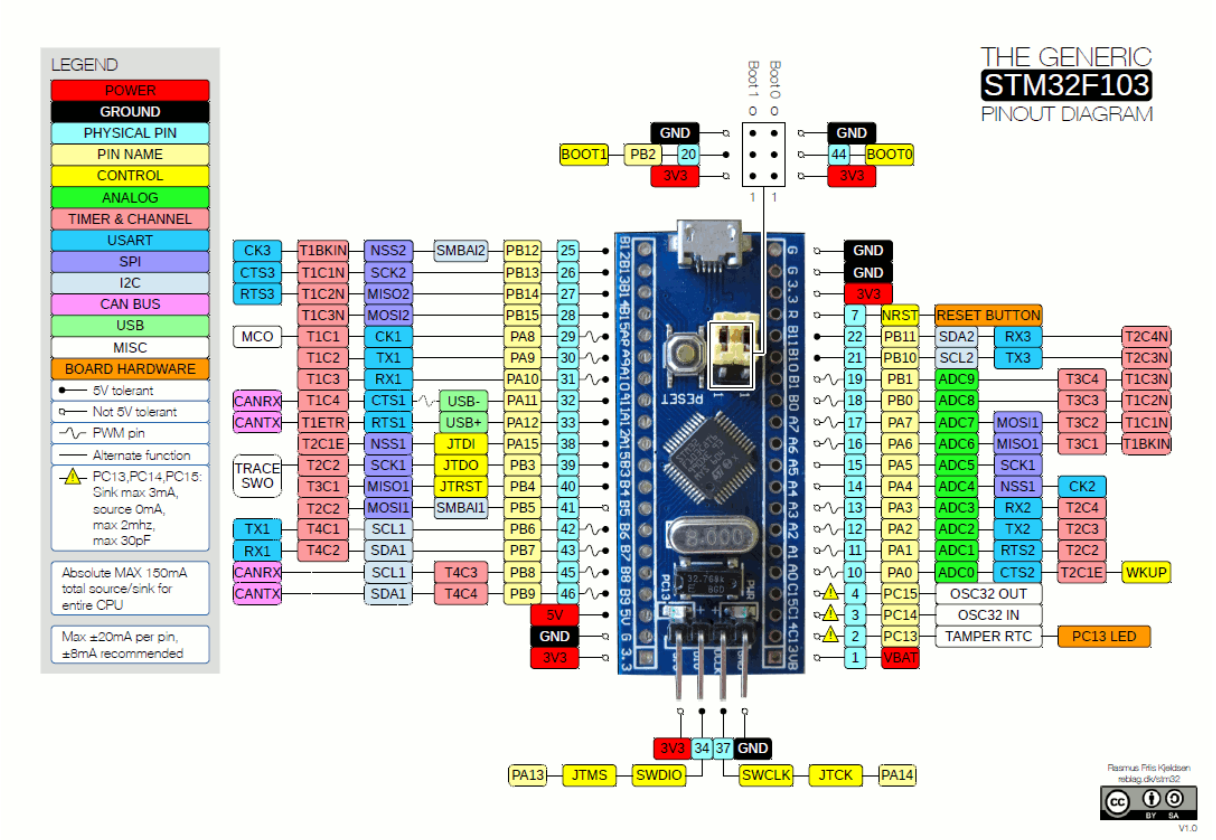


Figura 1: Pinout della scheda Blue Pill

Al centro della scheda verso il lato superiore ci sono due jumper; servono per cambiare le condizioni di avvio della scheda. In condizioni normali ad ogni accensione partirà il caricamento del programma, ma, cambiando il jumper, è possibile rimanere in fase di boot e riprogrammare un boot-loader per usare la scheda, ad esempio, con Arduino IDE.

Sul lato inferiore ci sono quattro pin; sono fondamentali per la programmazione della scheda e andranno collegati con ST-LINK V2 nel seguente modo SWCLK → SWCLK e SWD→ SWDIO.

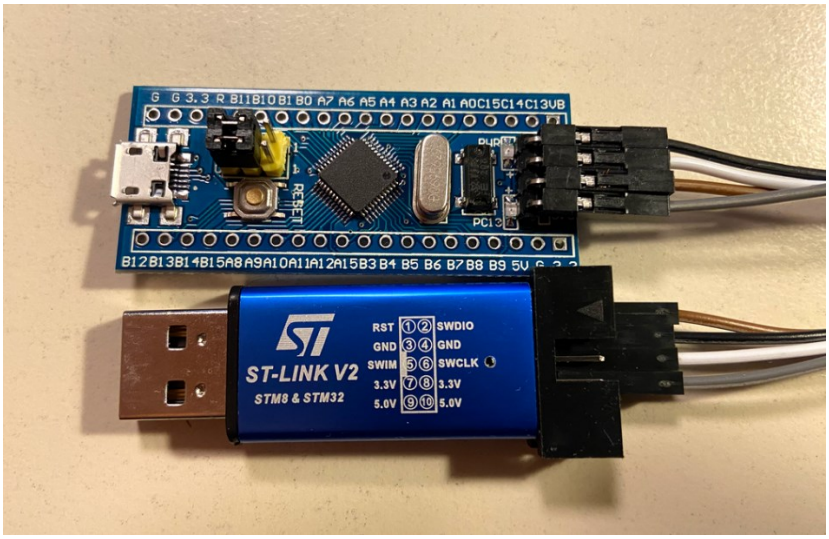


Figura 2: Collegamento ST-LINK e microcontrollore

Per scrivere il programma nella memoria interna del dispositivo ho usato il programma della ARM ST-LINK Utility.

Affinché avvenga correttamente la connessione e il dispositivo venga riconosciuto è opportuno impostare la connessione under hardware reset. In questo modo, ogni volta che si vorrà connettere il dispositivo al computer, bisognerà fermare l'esecuzione del programma e avviare la fase di boot premendo il tasto reset direttamente sulla scheda. Una volta eseguita la connessione, si può premere su Target e poi Program per caricare sulla memoria interna del dispositivo il file esadecimale generato dal compilatore.

4. STM32F103C8T6

La scheda utilizzata per il progetto si trova anche comunemente sotto il nome di Blue Pill, dato dal suo colore e la dimensione contenuta. Nonostante sia una scheda datata e piuttosto economica non manca di interessanti caratteristiche. Si basa infatti sul processore Arm Cortex M3 a 32 bit, progettato con architettura RISC, capace di lavorare con un clock di massimo 72MHz, sufficiente per lo sviluppo di svariate applicazioni. Dispone di una discreta quantità di porte I/O e periferiche, inclusi due ADC da 12-bit, tre timer a 16-bit e uno PWM. Sono presenti i principali sistemi di comunicazione standard come I2C, SPI, USART, USB e CAN. Nonostante ci siano tutte queste connessioni non sempre sono tutte disponibili contemporaneamente; alcune condividono dei pin di ingresso/uscita, costringendo il programmatore a fare delle scelte e arrivare a dei compromessi. Per lo scambio dei dati tra le periferiche e le memorie, il microcontrollore è equipaggiato non solo con un NVIC (Nested Vectored Interrupt Controller), ma anche con un DMA (Digital Memory Access) a memoria circolare. In questo modo, il passaggio dei dati può avvenire in assenza di interrupt lasciando il processore libero di eseguire istruzioni senza interruzioni ogni qual volta ci sia bisogno di spostare dati tra le memorie e le periferiche.

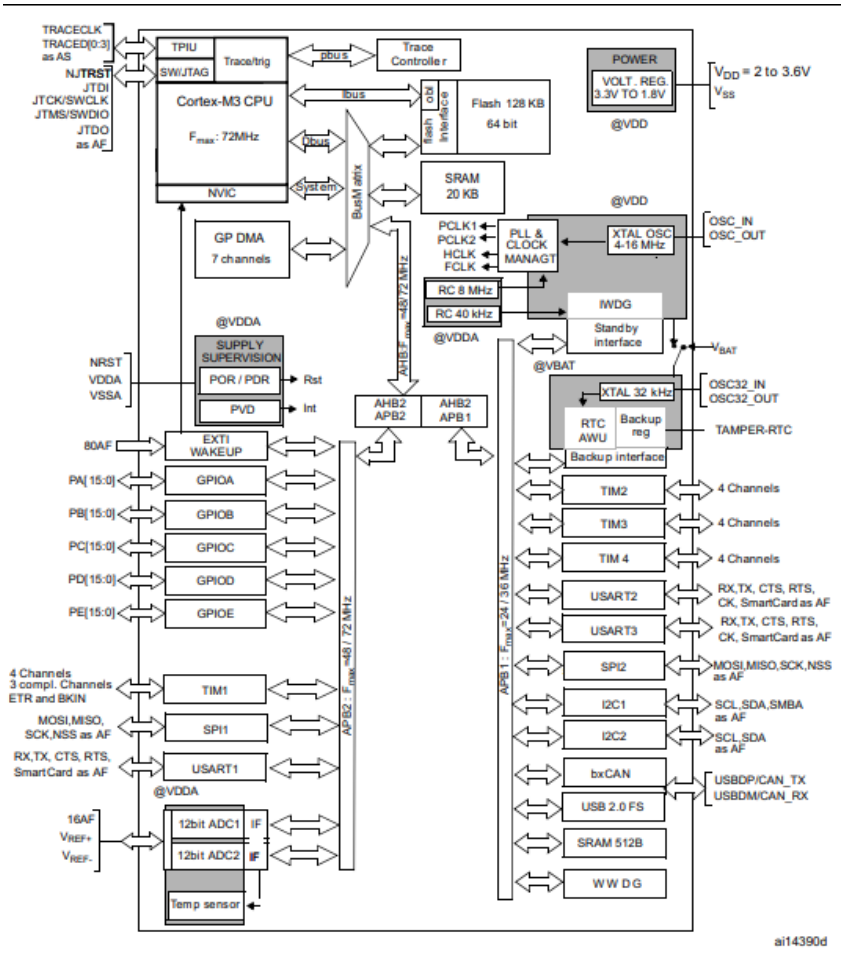


Figura 3: Panoramica connessioni STM32F103

4.1 SPECIFICHE DETTAGLIATE

- ARM® 32-bit Cortex®-M3 CPU Core
  - 72 MHz maximum frequency, 1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
  - Single-cycle multiplication and hardware division
- Memories
  - 64 or 128 Kbytes of Flash memory
  - 20 Kbytes of SRAM
- Clock, reset and supply management
  - 2.0 to 3.6 V application supply and I/Os
  - POR, PDR, and programmable voltage detector (PVD)
  - 4-to-16 MHz crystal oscillator
  - Internal 8 MHz factory-trimmed RC
  - Internal 40 kHz RC
  - PLL for CPU clock
  - 32 kHz oscillator for RTC with calibration
- Low-power
  - Sleep, Stop and Standby modes
  - VBAT supply for RTC and backup registers
- 2 x 12-bit, 1 µs A/D converters (up to 16 channels)
  - Conversion range: 0 to 3.6 V
  - Dual-sample and hold capability
  - Temperature sensor
- DMA
  - 7-channel DMA controller
  - Peripherals supported: timers, ADC, SPIs, I2Cs and USARTs
- Up to 80 fast I/O ports
  - 26/37/51/80 I/Os, all mappable on 16 external interrupt vectors and almost all 5 V-tolerant
- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
- 7 timers
  - Three 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input

    - 16-bit, motor control PWM timer with deadtime generation and emergency stop
    - 2 watchdog timers (Independent and Window)
    - SysTick timer 24-bit downcounter
- Up to 9 communication interfaces
    - Up to 2 x I2C interfaces (SMBus/PMBus)
    - Up to 3 USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)
    - Up to 2 SPIs (18 Mbit/s)
    - CAN interface (2.0B Active)
    - USB 2.0 full-speed interface
- CRC calculation unit, 96-bit unique ID
- Packages are ECOPACK®

4.2 GPIO

La struttura base di un pin di I/O è visibile nella figura 4. I pin di ingresso uscita sono controllati tramite due registri per la configurazione (32 bit) e due registri per i dati (32 bit). Il programmatore può settare i bit per ottenere diverse funzioni come:

- **Input flottante**: ingresso ad alta impedenza
- **Input pull-up**: ingresso forzato alto
- **Input pull down**: ingresso forzato basso
- **Analog**: ingresso di un segnale analogico per ADC
- **Output open drain**: il PMOS collegato al livello alto viene staccato, di conseguenza il pin di uscita può essere solo o float o collegato a massa. Questa modalità è sensata se usata con una resistenza pull up esterna.
- **Output push pull**: l'uscita oscilla tra GND o Vcc.
- **Alternate function push pull**: la funzione del pin può cambiare ed essere usato dalle varie periferiche all'interno del microcontrollore.
- **Alternate function open drain**.

Ci sono poi un registro di set/reset (32 bit) e un registro di reset (16 bit) utili per impedire che si svolgano delle IRQ tra la lettura e la modifica della configurazione del pin quando questo è configurato in alternate function. Per ultimo, è presente un registro di locking per bloccare la configurazione dei pin fino al successivo reset.

INPUT MODE

Quando una porta è in modalità ingresso, il buffer di output viene disattivato, si attiva il Trigger di Schmitt e il dato ricevuto viene inserito all'interno dell'input data register ad ogni ciclo di clock.

Due interruttori vanno ad isolare, o meno, le resistenze in base al tipo di input impostato nel configuration register.

Sono inoltre presenti due diodi per la protezione da sovratensioni, sia positive che negative. Nella porta rappresentata nella Figura 4, la tensione massima consentita è Vdd – Vss = 4.0V. Sono presenti anche dei pin individuabili tramite i datasheet che supportano una tensione massima di Vdd + 4.0V collegabili quindi ai 5V senza problemi. Bisogna inoltre fare attenzione alla corrente massima sia in ingresso che in uscita, la quale non deve superare i 25mA per entrambi i tipi di pin.

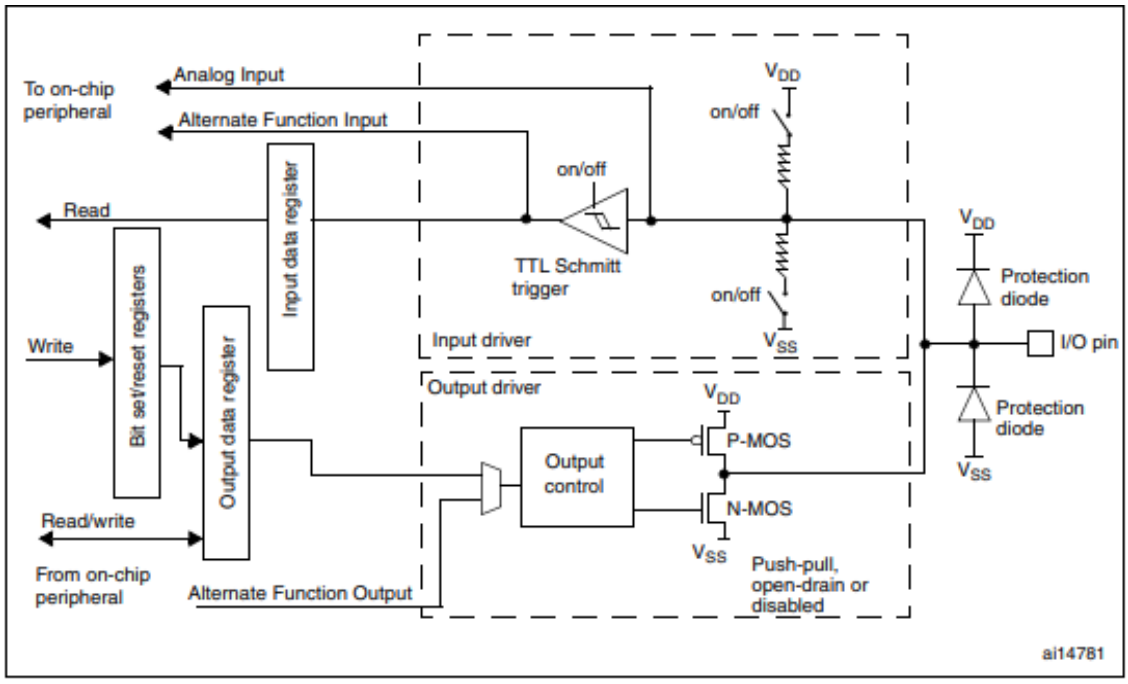


Figura 4. Struttura base pin I/O

OUTPUT MODE

Nel momento in cui il pin viene settato come output, l'output buffer si attiva e i due MOSFET vengono pilotati in base alla modalità selezionata. In caso di open drain mode il P-MOS non verrà mai attivato, l'output data register andrà ad operare solo sul dispositivo N-NOS. Se invece si seleziona la modalità push-pull allora saranno attivabili entrambi a seconda del caso. Nella sezione di input viene attivato il Trigger di Schmitt mentre vengono staccate entrambe le

resistenze di pull-up e pull-down. L'uscita viene quindi campionata all'interno del Data register in modo da raccogliere informazioni sullo stato corrente, in caso di modalità open drain, o sull'ultimo valore che è stato emesso nella modalità Push Pull.
Agendo su appositi bit nel control register è anche possibile impostare la velocità massima di output tra le seguenti:
Bassa, fino a 2MHz
Media, fino a 10Mhz
Alta, fino a 50 Mhz

4.3 TIMER

All'interno del STM32F103 sono presenti sette timer; tre timer generici, un timer avanzato, due watchdog e un SysTick timer, più che utili a coprire la maggior parte delle applicazioni che si vogliono realizzare.

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
TIM1	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	Yes
TIM2, TIM3, TIM4	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No

Ogni timer è indipendente e ha il proprio bus; quindi, possono essere usati contemporaneamente senza creare conflitti o rallentamenti. Il timer è composto da un circuito logico in grado di contare i fronti positivi che riceve in ingresso e scriverli all'interno di un registro chiamato Counter register (TIMx_CNT). Nel caso del microcontrollore il segnale in ingresso può variare in base a quello che sceglie il programmatore; può essere il segnale di clock, un segnale proveniente da un pin esterno, oppure un segnale generato internamente. Una volta che il contatore ha raggiunto il massimo valore che può contare (dipende dalla risoluzione, nel nostro caso, visto che è a 16 bit, è pari a 65536) il timer va in overflow e viene generato un Update event.
Ogni timer è dotato di un prescaler per dividere la frequenza in ingresso di un dato valore a 16 bit inserito nel prescaler register (TIMx_PSC), anch'esso a 16 bit.

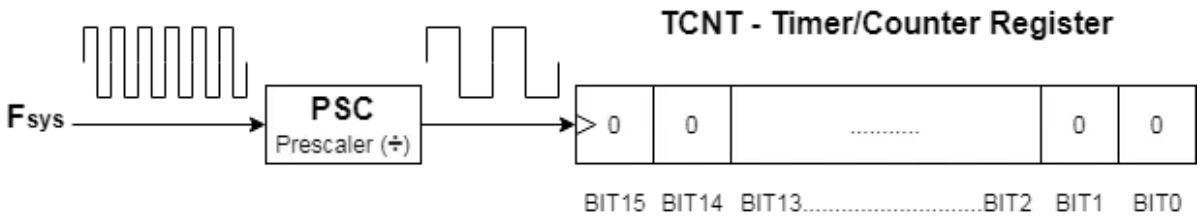


Figura 5. Prescaler con valore 3, in questo modo 6 colpi di clock diventano 2

Ad esempio, se volessimo contare i microsecondi pur avendo un clock di 72MHz possiamo seguire la seguente formula per ricavare il corretto valore del prescaler $T_{out} = \frac{PSC \times Preload}{F_{CLK}}$. $T_{out} = 1 \times 10^{-6} s$, $F_{CLK} = 72 \times 10^{6} Hz$ mentre il $Preload$=1 è facile ricavare 72 come valore da inserire nel prescaler register.

In questo modo, ogni 72000 colpi di clock avrò un incremento del valore del Counter register fino al raggiungimento dell'overflow dopo 65,536 secondi.

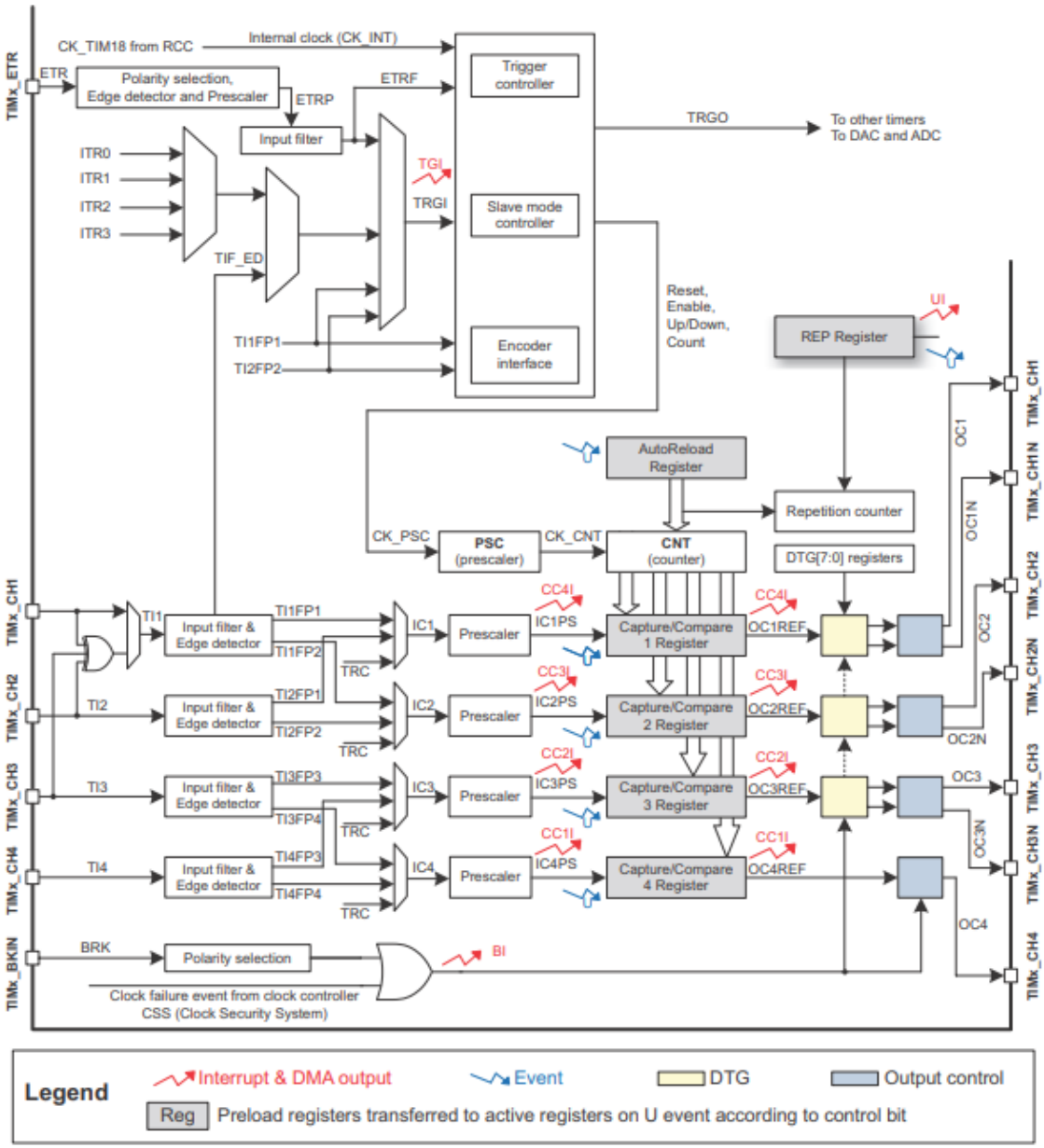


Figura 6: Rappresentazione dei timer del dispositivo

TIMER AVANZATO (TIM1)

Il timer avanzato dispone di 4 canali indipendenti, un prescaler programmabile e un contatore a 16 bit con funzione di auto reload; quindi, aggiornabile anche mentre questo sta contando. Ogni canale può eseguire diverse funzioni come:

- Input capture
- Output compare
- PWM generation
- One pulse mode output.

Non manca la possibilità di sincronizzarlo con gli altri timer, generare interrupt o DMA dopo un overflow/underflow, un evento di trigger, un input capture o un output compare.

Più avanti verranno descritte le varie funzioni più nel dettaglio.

4.4 FUNZIONI DEL CONTATORE

Input capture

Nel momento in cui si verifica una condizione di attivazione (può essere un fronte positivo o negativo a seconda dell'impostazione) il timer inizia a contare, quando poi, in un secondo momento, si verifica una seconda condizione di attivazione viene generata una richiesta al DMA o un Interrupt passando così al microprocessore il valore presente all'interno del counter register. Bisogna prestare attenzione che sia un valore positivo e che non vi siano generati più di un overflow tra un evento e l'altro per evitare la lettura di valori falsi. Questa funzione può essere utile, ad esempio, nel caso si stia utilizzando un sensore ad ultrasuoni e si vuole calcolare la distanza di un oggetto dal sensore. Viene generata la prima condizione di attivazione quando si emettono gli ultrasuoni mentre si verifica la seconda quando il suono torna al sensore. Il valore registrato nel counter register rappresenta il tempo che il suono ha impiegato per rimbalzare sull'oggetto e tornare al sensore; sarà immediato poi usarlo per ricavarne la distanza.

Output compare

Il timer dispone di un compare register (CCR1) nel quale il programmatore può inserire un valore a 16 bit. Quando il timer raggiungerà il valore contenuto nel CCR1 verrà asserito un bit di uscita (OC1) che generalmente attiva un pin che rimarrà alto fino al verificarsi del prossimo evento. Questa modalità può essere utile per generare impulsi a durata variabile o chiamare determinate funzioni in tempi molto precisi

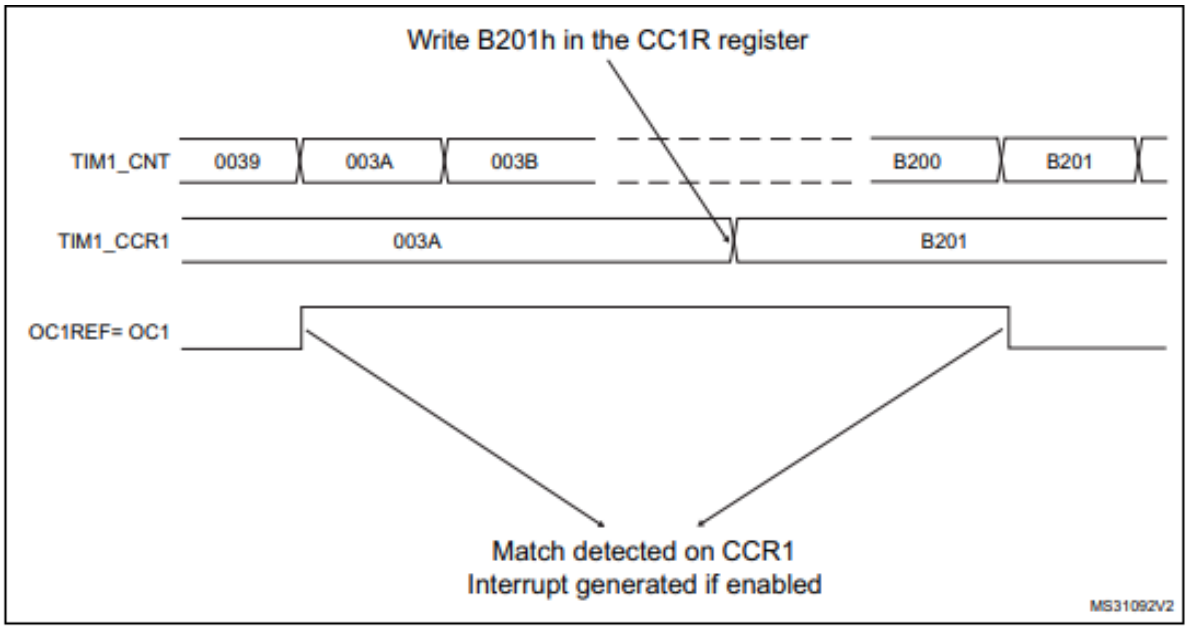


Figura 7:Timing dei segnali del contatore

PWM

La modalità PWM è una semplificazione di Output compare pensata per semplificare la generazione di impulsi con frequenza prefissata e duty cycle variabile. Agendo sul prescaler e il counter period è possibile variare il periodo di ogni impulso; tramite il compare register si può invece impostare il duty cycle. Bisogna tener conto che la risoluzione del PWM diminuisce con l'aumentare della frequenza del periodo di modulazione che viene scelto.

One pulse mode output

Questa modalità è utile se si vuole generare un impulso con una durata definita e un certo delay da una condizione generante. Per ottenere l'effetto desiderato il timer deve essere inizializzato in slave mode. Ogni volta che si verificherà la condizione di trigger il timer inizierà a contare, nel momento in cui il valore del counter register (CNT) sarà uguale a quello impostato nel compare register (CCR), il pin di uscita cambia stato e lo mantiene fino al raggiungimento dell'overflow (ARR). Le condizioni da soddisfare sono quindi CNT <CCR ≤ ARR.

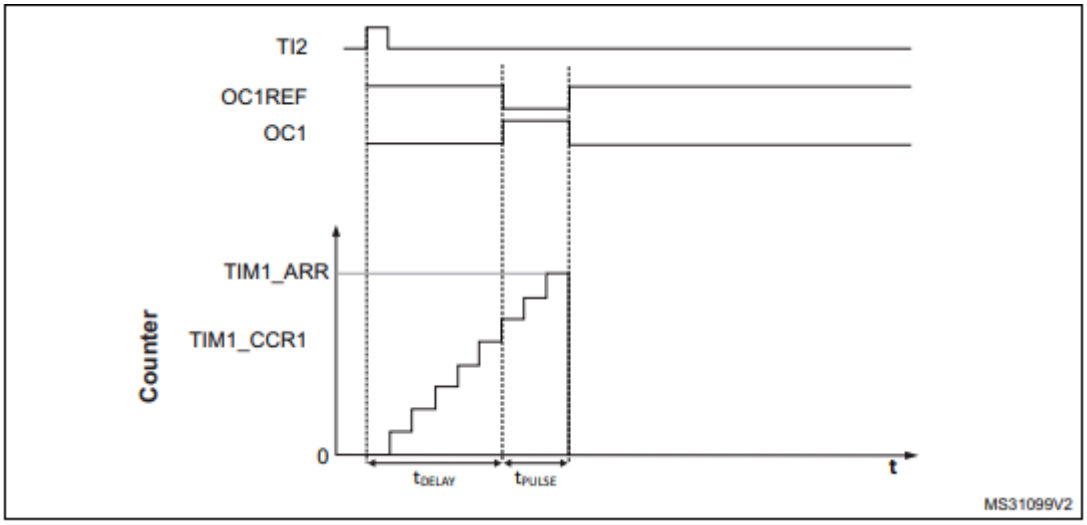


Figura 8: Funzionamento segnali interni timer

TI2: evento di trigger

OC1REF: uscita output compare

OC1: impulso desiderato in uscita

4.5 ADC

Il microcontrollore è provvisto di due convertitori analogico digitale con una risoluzione di 12 bit. Ogni ADC ha fino a 10 ingressi per segnali esterni e 2 per segnali interni, è possibile accedere ad ognuno di essi in maniera quasi contemporanea attraverso un multiplexer. Una volta avviata la conversione, e se verrà abilitata la scan conversion, l'ADC memorizzerà i dati di ogni canale all'interno del data register, un registro a 16 bit con possibilità di scegliere se usare un allineamento a destra o a sinistra.

Il tempo di campionamento è indipendente per entrambi i canali, mentre il clock massimo per la conversione dell'ADC è di 14MHz. Il tempo per eseguire una conversione è di 1 µs a 56 MHz (1.17 µs a 72 MHz, l'aumento è dovuto al fatto che con i prescaler dell'ADC non si riesce a raggiungere la velocità massima di 14MHz ma ci si deve fermare a 12MHz.)

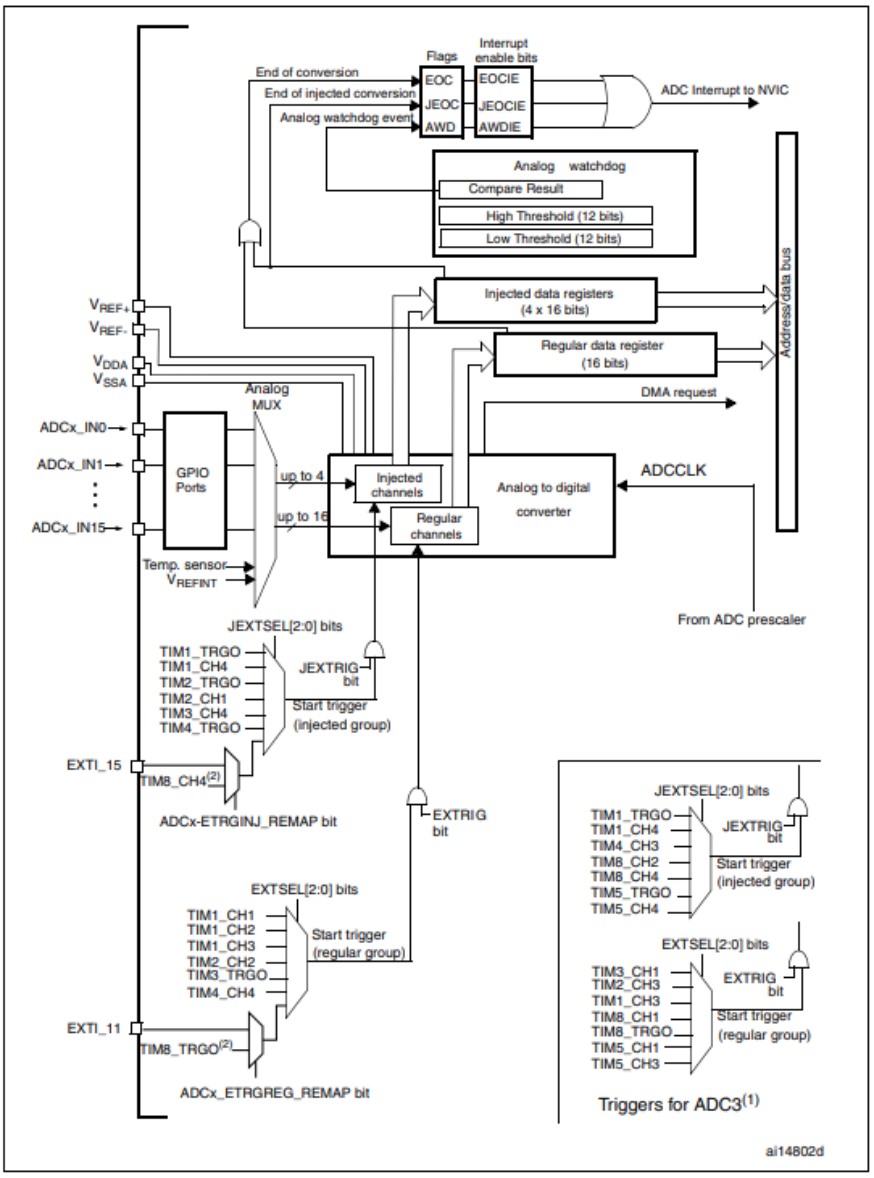


Figura 9 Diagramma a blocchi ADC

Il convertitore dispone anche di una funzione di auto calibrazione; è fortemente consigliato avviarla ad ogni accensione del dispositivo. Si può decidere di avviare una conversione attraverso un segnale di trigger software esterno oppure di tipo injected. Per capire quale dei tre sia il corretto si può fare riferimento alla figura 6, partendo dalla periferica che lo genera fino ad arrivare all'ADC interessato.

Come ultima cosa è importante tenere conto che il segnale da campionare deve rientrare nei limiti di VREF- ≤ VIN ≤ VREF+ oppure possiamo riscontrare errori di saturazione.

Modalità conversione singola

Il convertitore analogico digitale può essere impostato in conversione singola settando il bit CON a 0. In questo modo, solo dopo aver ricevuto un segnale di partenza, attraverso il bit ADON per le conversioni regolari o attraverso un trigger esterno per le injected, inizia la conversione.

Nel momento in cui la conversione è completa:

- Il dato viene memorizzato nel registro ADC_DR (ADC_DRJ1 in caso di injected)
- Viene attivato il flag EOC (JEOC nel caso injected)
- Nel caso fosse specificato viene generata una interruzione EOCIE (JEOCIE nel caso injected)

Affinché la conversione vada a buon fine e il convertitore si fermi è importante che venga attivato il flag di fine conversione. Nel caso venisse triggerato l'ADC durante una conversione quel trigger andrà perso.

Continuos conversion mode

Impostando il bit CON a 1 si attiva la conversione continua, il convertitore esegue la prossima scansione appena terminata la precedente. In questa modalità la frequenza di campionamento è selezionabile solo modificando la frequenza di clock, ad esempio, agendo sul prescaler dell'ADC. Una volta che la conversione è terminata viene eseguito lo stesso procedimento della conversione singola.

Analog watchdog

La funzione watchdog è molto utile per monitorare un segnale e generare un interrupt se questo è superiore o inferiore ad una determinata soglia. La soglia viene definita dal programmatore scrivendo nel registro ADC_HTR e ADC_LTR. Nonostante siano entrambi a 16 bit il valore

viene rappresentato dai 12 bit meno significativi avendo di conseguenza un limite massimo di 4095 livelli. Il valore della soglia impostata è indipendente dal tipo di allineamento dei dati scelto per il control register.

Scan mode

Per la conversione di canali multipli connessi allo stesso ADC si fa uso del multiplexer. Per abilitare questo tipo di conversione si setta lo SCAN bit nel rispettivo registro. Visto il considerevole numero di conversioni e la velocità con la quale vengono eseguite è obbligatorio l'utilizzo di un DMA quando questa funzione è attiva.

5. PROGRAMMAZIONE DELLA SCHEDA

Per la programmazione vengono utilizzati tre software, STM32CubeMX per la configurazione delle periferiche, Keil µVision 5 per la stesura del codice e per ultimo ST-Link utility per il caricamento del file binario sulla scheda.

5.1 IMPOSTAZIONE DELLE PERIFERICHE

STM32CubeMX è un tool grafico che permette la configurazione delle periferiche di un microcontrollore STM32 senza il bisogno di scrivere codice. Non è fondamentale, ma il suo utilizzo semplifica fase iniziale della programmazione. Il programma è compatibile con una vasta gamma di microcontrollori, microprocessori e schede di sviluppo. Iniziando a scrivere il codice identificativo del dispositivo di interesse il software mostrerà tutte le periferiche supportate e le relative specifiche tecniche con una breve descrizione. In questo modo si può avere una conferma che i dati inseriti siano corretti.

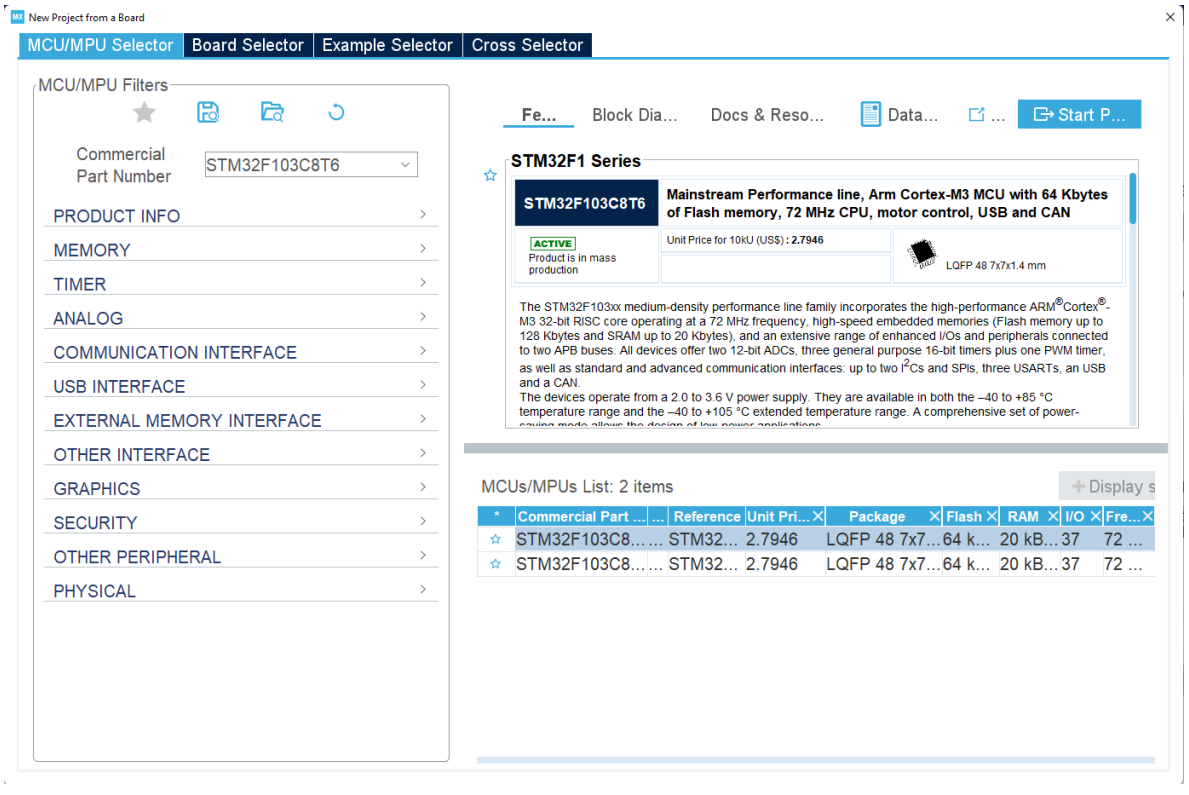


Figura 10: Pagina iniziale STMCubeMX

Nella parte superiore dello schermo verranno inoltre forniti degli utili collegamenti a datasheet, shop e altre risorse utili al programmatore.

Andando a selezionare la scheda corretta si può poi premere su start project e iniziare la vera e propria configurazione iniziale. Tramite CubeMX è possibile, per esempio, impostare il clock

dei timer, scegliere i vari pin di input output, configurare i timer o gli ADC, se disponibili all'interno del dispositivo, impostare un sistema di comunicazione e gestire i vari interrupt.
Una volta che vengono impostate le opzioni desiderate il programma andrà a generare il progetto per µVision (o altri compilatori compatibili) contenente il linguaggio in C++ per l'utilizzo e la configurazione delle periferiche. Una semplificazione notevole per il programmatore, al costo di un codice non perfettamente ottimizzato che potrà comunque essere "ripulito" in un secondo momento in caso di necessità.

Una volta iniziato il progetto si presenterà la seguente schermata:

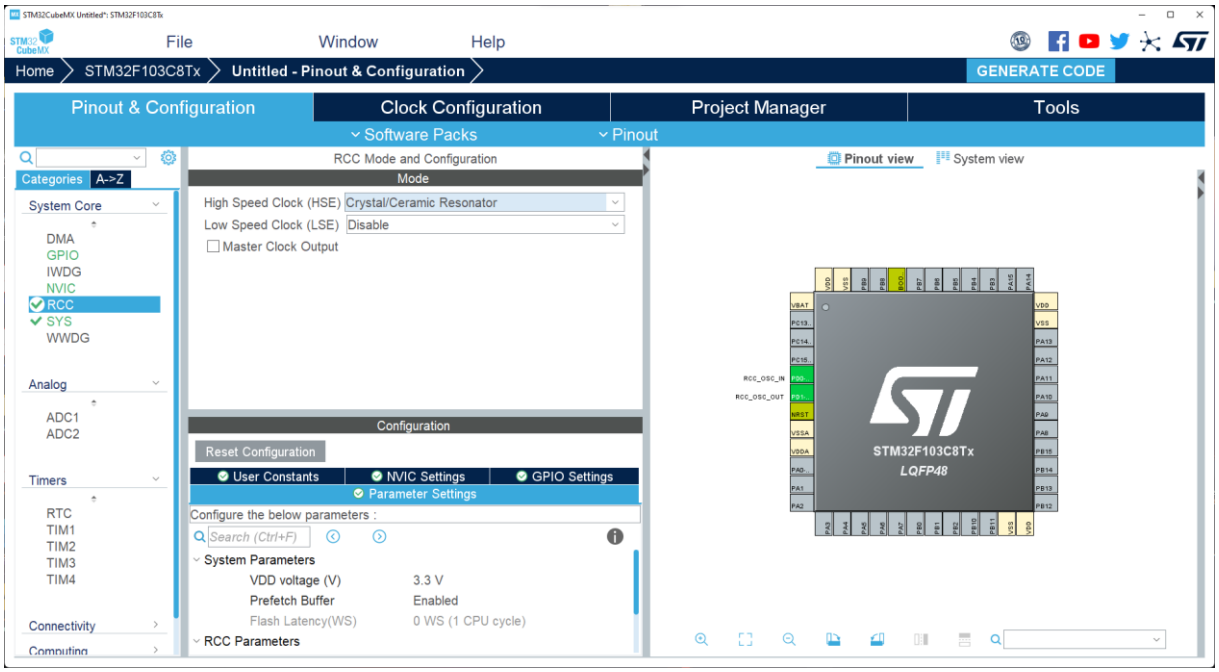


Figura 11: Schermata principale del progetto

Sulla sinistra si possono visualizzare le periferiche divise per categoria, nel centro ci sono le impostazioni che è possibile modificare utilizzando dei menù a tendina o scrivendo un valore da tastiera mentre sulla destra viene mostrato il microcontrollore con i pin utilizzati e quelli liberi. Nell'esempio in figura 11, ho configurato il clock. Per farlo ho selezionato RCC (Reset and clock control) e ho settato l'High speed clock su Crystal Ceramic Resonator. In questo modo il microprocessore utilizzerà l'oscillatore al quarzo esterno da 8MHz per generare il clock interno a velocità massima di 72MHz. Una volta fatta questa operazione il programma ci segnala che due pin del microcontrollore verranno usati come input, i pin PD0 e PD1 diventano verdi, segnalando che sono attualmente in uso e non potranno essere utilizzati da altre periferiche.

Per usufruire della velocità massima di clock non basta limitarsi ad usare un quarzo esterno, è importante configurare il Clock tramite l'apposita sezione.

Cliccando su Clock Configuration si aprirà uno schema a blocchi che mostra tutti i multiplexer, prescaler e multiscaler che è possibile utilizzare per raggiungere la frequenza desiderata non solo nel processore, ma anche nelle varie periferiche. Se durante la configurazione si impostasse un valore troppo alto il software lo segnalerà colorando di rosso la casella col valore errato e impedendo all'utente di procedere. Nel caso si facesse confusione o non si riuscisse ad avere la frequenza desiderata è possibile usare l'opzione "Resolve clock issues" che in automatico andrà a modificare i prescaler e multiscaler in modo da non eccedere con le frequenze massime. Nonostante sembri la soluzione migliore, è comunque sconsigliata perché non sempre trova la configurazione più ottimizzata.

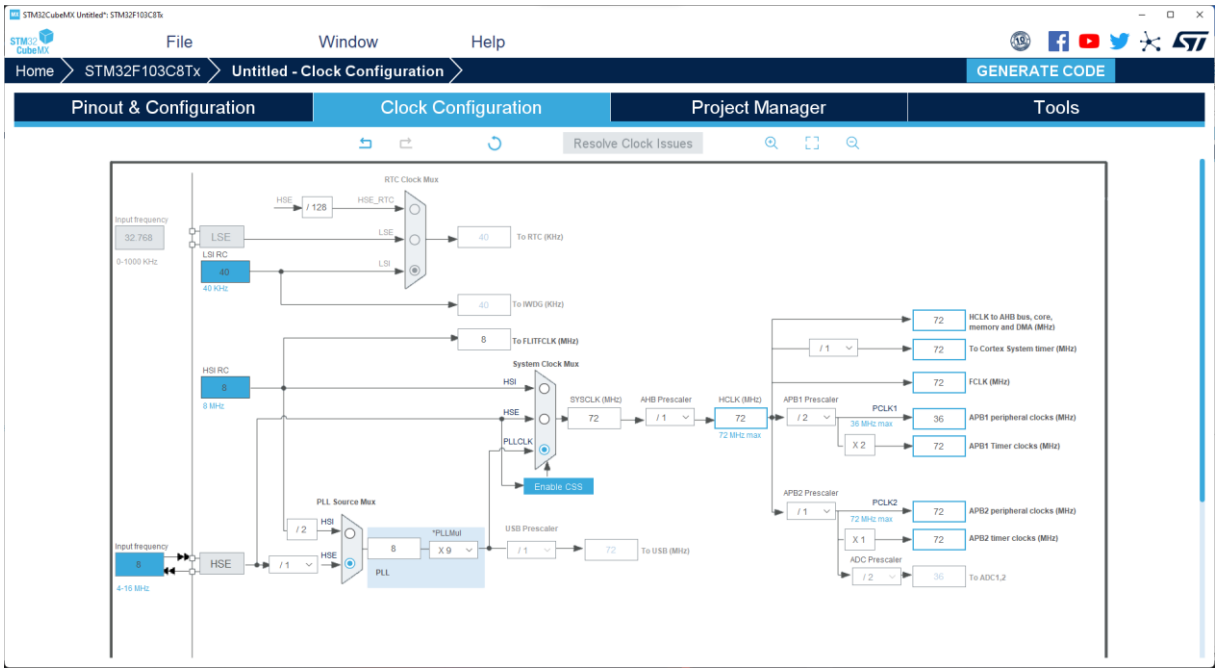


Figura 12: Impostazioni di clock

Per impostare il clock del processore alla massima velocità, dopo aver attivato il quarzo, ho eseguito il seguente procedimento: spostare il toggle del multiplexer PLL su HSE, modificare il PLL multiplier a ×9, spostare il toggle su PLLCLK sul System clock multiplexer. A questo punto il programma mi segna errore di clock troppo elevato sulla beriferica ADB1, per risolvere il problema basta ridurre la frequenza delle periferiche ADB1 impostando il prescaler a /2. Il risultato finale si può vedere nella figura 12.

La configurazione della frequenza del clock è giunta al termine; posso tornare alla pagina precedente e proseguire con la configurazione delle periferiche. Basta fare un click sui pin desiderati per scegliere la configurazione. Nel mio caso ho deciso di usare i pin PA1,2,3,4,5,6

come output push-pull per comandare i potenziometri digitali, mentre ho impostato il pin PA9 come ingresso no pull-up no pull-down per la ricezione del segnale dal sensore ad infrarossi. Come è possibile vedere dalla figura 13, sono comparsi dei triangoli arancioni che rappresentano dei warning in prossimità di ADC1,2 e TIM1.

STM32CubeMX informa l'utente che si sono creati dei conflitti con le periferiche selezionate, andando a indagare nelle relative opzioni è immediato notare che alcune impostazioni vengono evidenziate in rosso, segno che non sono più disponibili perché per funzionare utilizzano gli stessi pin impostati prima come uscite

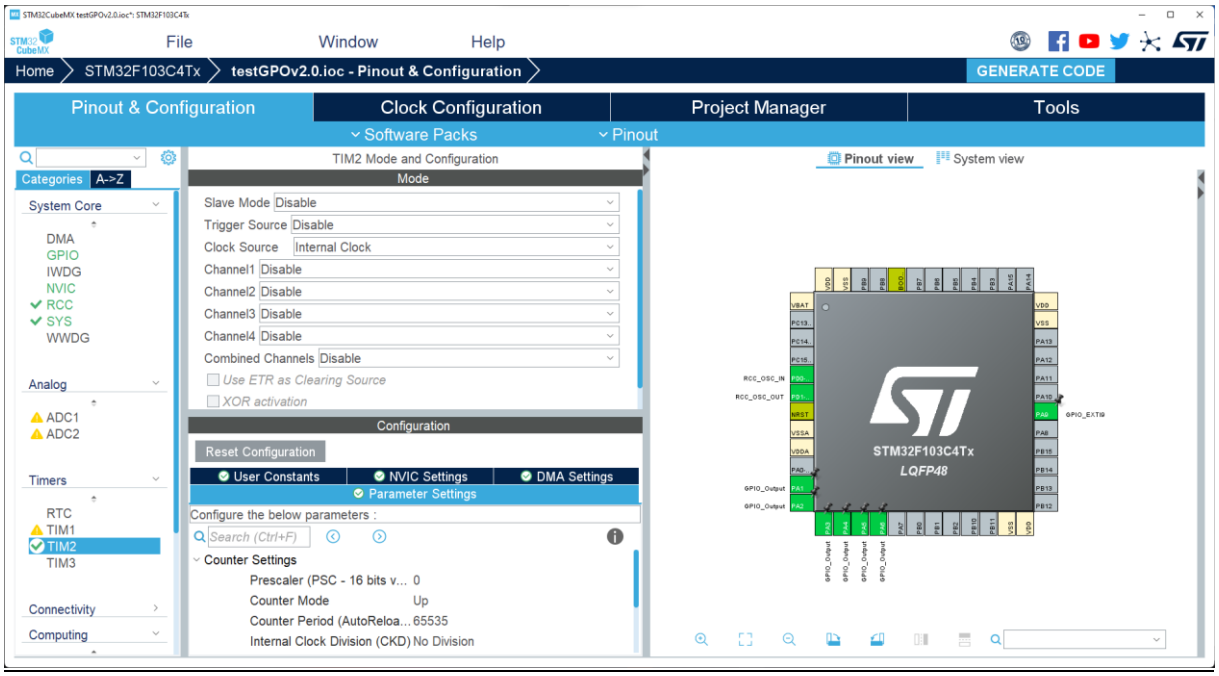


Figura 13: Impostazioni del timer

Per esempio, posso vedere che il TIM1 utilizza PA9 per l'output compare del canale 2. Visto che questa funzione non mi interessa posso lasciare tutto invariato senza preoccuparmi di modificare nulla. In caso contrario, sarei costretto a modificare il pinout scelto precedentemente. Risulta quindi evidente che per progetti più complicati ci sia il bisogno di scendere a compromessi.

Terminata la configurazione dei GPIO posso passare al setup del timer. Per evitare problemi di conflitti e considerato che non ho alcun bisogno di funzioni avanzate ho deciso di utilizzare un timer base come il TIM2. Il timer verrà usato per la temporizzazione dei segnali; non ho bisogno di generare interrupt, ma ho bisogno di cambiare il valore del contatore mentre questo è ancora in corsa. Per attivare questa funzione basta spostare la tendina relativa all'auto reload preload su Enable. Per la gestione delle temporizzazioni utilizzerò il timer all'interno di due funzioni che scriverò successivamente in µVision, una che conta i microsecondi e la seconda si avvale di una variabile ausiliaria per contare i millisecondi.

I valori accettati dal programma per la configurazione del prescaler vanno da 0 a 65525, quindi, se vogliamo dividere la frequenza di clock di 72 volte bisogna impostare un valore di prescaler pari a 72-1, ottenendo così un incremento ogni microsecondo. Lasciando il valore di overflow al massimo disponibile di 65525 ottengo un massimo di secondi contabili dal timer pari a 0,065536s, risulta quindi importante scrivere una funzione apposita per contare i ms per periodi di tempo più lunghi.

Una volta che sono terminate le configurazioni delle periferiche è possibile generare il progetto in modo che passa essere utilizzato tramite µVision, per farlo bisogna selezionare il tab Project Manager e impostare le ultime specifiche di progetto prima dell'esportazione.

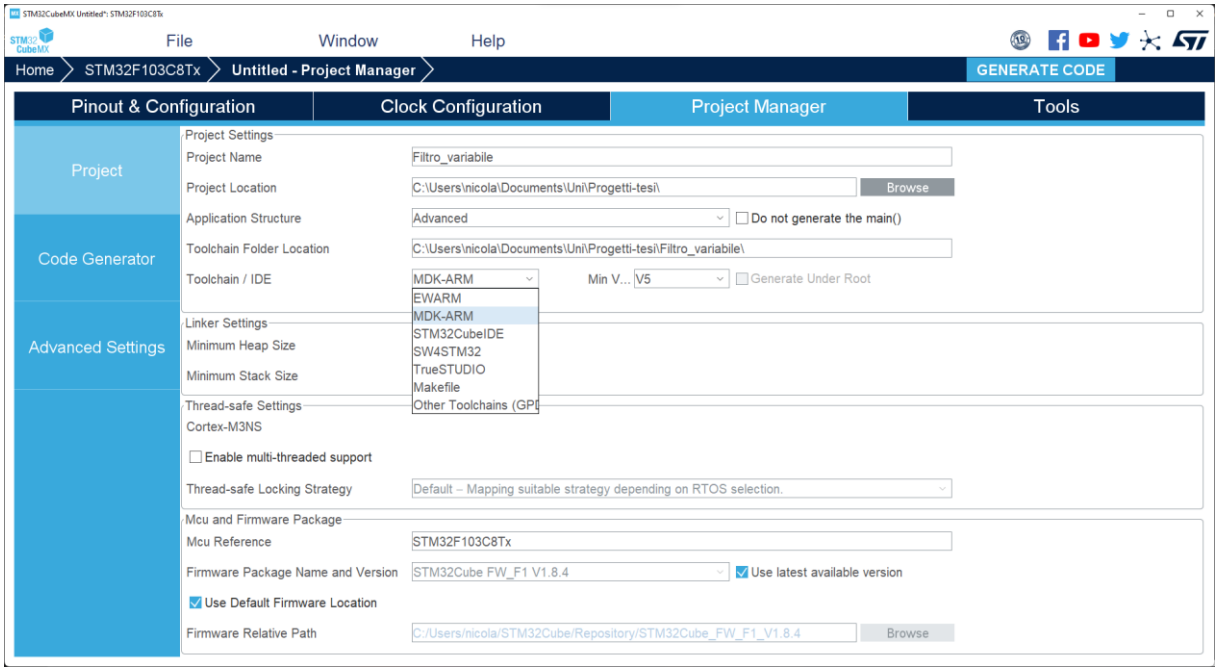


Figura 14: schermata conclusiva

Oltre al nome del progetto e al relativo percorso è importante scegliere il corretto Toolchain/IDE, nel mio caso usando µVision seleziono MDK-ARM. Premendo poi su GENERATE CODE in pochi istanti verrà generato il codice del progetto, nella destinazione impostata, pronto per essere implementato dall'utente.

5.2 PROGRAMMAZIONE DEL TIMER

Quando STM32CubeMX genera del codice inserisce all'interno dei commenti segnalando l'area riservata al codice dell'utente. Scrivendo codice fuori dall'area designata non ci saranno problemi di funzionamento, ma nel caso l'utente voglia cambiare qualche configurazione con CubeMX il codice utente non verrà cambiato mentre il resto sarà completamente sovrascritto.

La funzione del timer verrà scritta aprendo il progetto con μVision e andando a modificare la funzione void MX_TIM2_Init(void) generata in automatico da STM32CubeMX all'interno del file tim.c. Questa funzione verrà chiamata ad ogni accensione del dispositivo e serve per configurare la periferica con i parametri precedentemente impostati tramite CubeMX.

```
void MX_TIM2_Init(void)
{

 /* USER CODE BEGIN TIM2_Init 0 */
 /* USER CODE END TIM2_Init 0 */

 TIM_ClockConfigTypeDef sClockSourceConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};

 /* USER CODE BEGIN TIM2_Init 1 */
 /* USER CODE END TIM2_Init 1 */

 htim2.Instance = TIM2;
 htim2.Init.Prescaler =72-1 ;
 htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim2.Init.Period = 65535;
 htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
 if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
 {
  Error_Handler();
 }
 sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
 if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
 {
  Error_Handler();
```

```
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /* USER CODE BEGIN TIM2_Init 2 */
  HAL_TIM_Base_Start(&htim2);

  /* USER CODE END TIM2_Init 2 */
}
```

All'interno dell'area riservata al codice dell'utente ho inserito il comando HAL_TIM_Base_Start(&htim2) fondamentale per far partire il conteggio del timer alla fine della configurazione dello stesso. In questo modo il timer partirà in modalità Free Run contando in maniera continua, anche quando non sarà utilizzato. Se mi fossi dimenticato di inserire questa riga il timer sarebbe completamente inutilizzabile.
Per il conteggio dei microsecondi e dei millisecondi ho scritto due funzioni distinte all'interno di un nuovo file chiamato Driver_digipot.c inserito all'interno del progetto. In questo file inserirò la maggior parte delle funzioni usate per il funzionamento dei potenziometri digitali.
Prima ho scritto la funzione per il conteggio dei microsecondi:

```
void delay_us(uint16_t num_us)
{
   htim2.Instance->CNT = 0;
   while (htim2.Instance->CNT < num_us);
}
```

In caso di chiamata della funzione delay_us, tramite il comando htim2.Instance->CNT = 0 si va a scrivere uno zero all'interno del registro CNT del timer 2, in questo modo si va ad azzerare il contatore mentre questo è ancora in corsa senza il bisogno di fermarlo. In secondo luogo, si usa la funzione while per fermare il programma finché il contatore non ha raggiunto il valore prestabilito. Il codice funziona solo se i valori del prescaler e la frequenza di clock sono stati

impostati correttamente, cambiando anche uno dei due il codice non è detto che funzioni correttamente.

La finzione delay_ms conta invece i millisecondi usando i microsecondi andando così a eliminare il rischio di overflow del timer

```
void delay_ms(uint16_t num_ms){
   while(num_ms > 0)
   {
   htim2.Instance->CNT = 0;
   while (htim2.Instance->CNT < 1000);
   num_ms--;
   }
}
```

Il limite massimo di millisecondi è dato solamente dalla lunghezza della variabile time, essendo 16 bit sarà 65,536 s, più che sufficienti per la nostra applicazione.

Per verificare la precisione di funzionamento del codice, ho scritto un piccolo programma che genera un impulso positivo della durata di 1ms ogni 10 ms. Andando successivamente a visualizzare su oscilloscopio l'uscita del pin PA1 posso osservare come si comporta il sistema

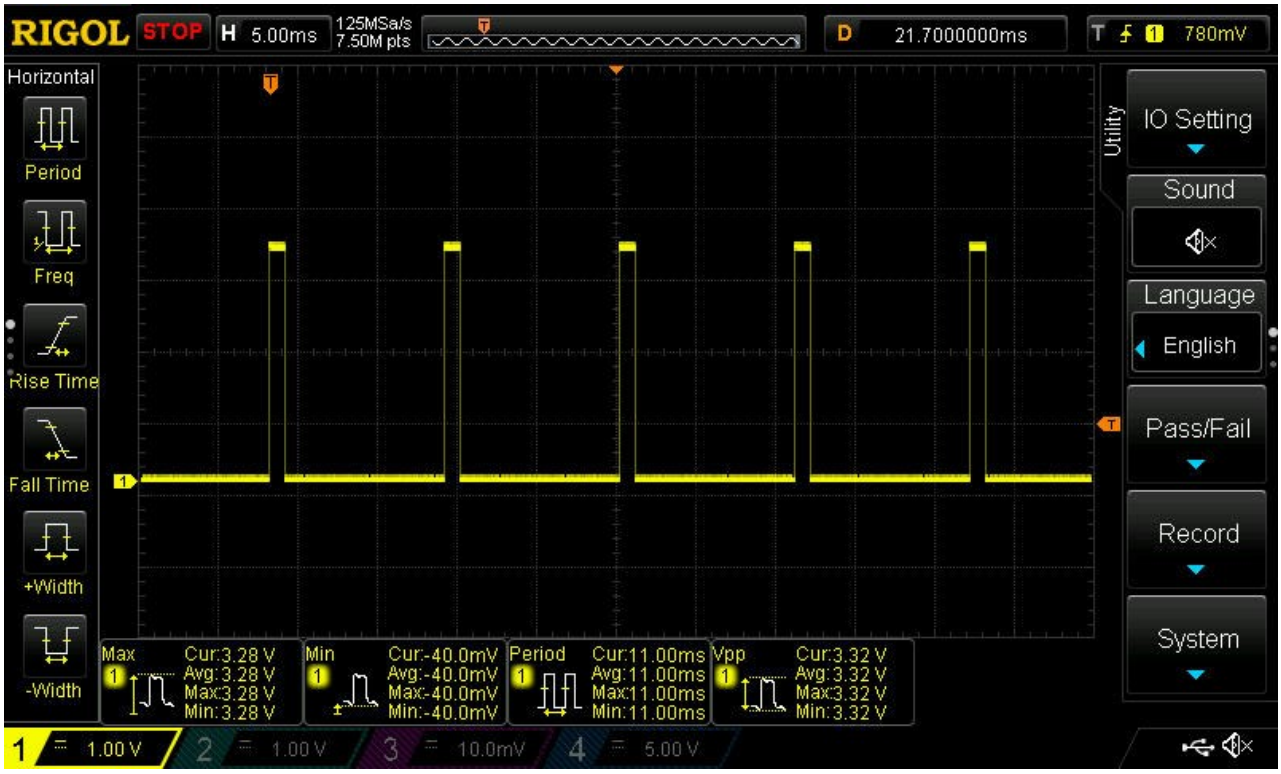


Figura 15: segnale di test

Il periodo risulta di 11.00ms ed è perfetto, per sicurezza misuro la durata dell'impulso:

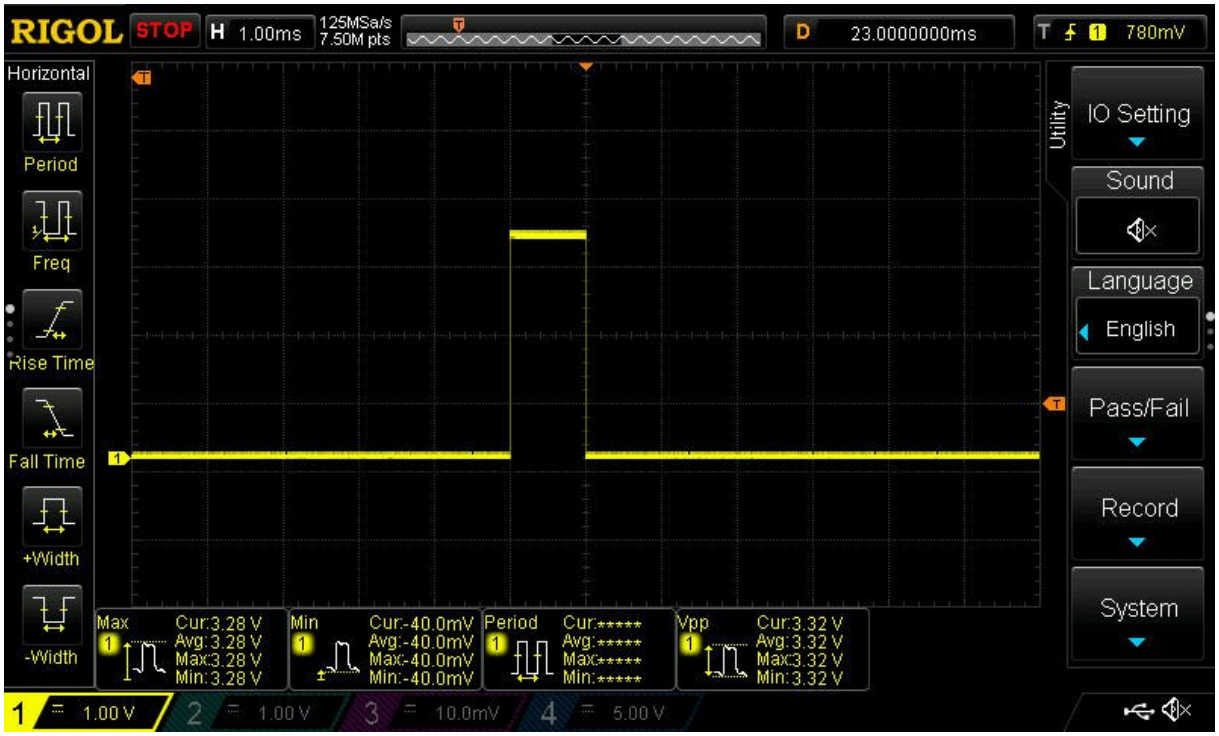


Figura 16: zoom segnale di test

Anche in questo caso la durata è di 1ms preciso. Come si può osservare la precisione è ben oltre la richiesta di progetto.

6. POTENZIOMETRO DIGITALE

Esistono tre macrocategorie di potenziometri digitali, differenti principalmente nel modo di comunicare. La prima si basa sulla comunicazione I2C per ricevere il valore di resistenza da impostare dal microcontrollore direttamente in forma numerica; la seconda usa la comunicazione tramite SPI mentre la terza, dal funzionamento più semplice ma meno immediato, utilizza tre diversi pin che in base a come verranno settati dal microcontrollore andranno ad incrementare o diminuire il valore della resistenza. Per lo scopo del progetto e data la scarsa disponibilità con conseguente elevato prezzo del modello con comunicazione I2C ho deciso di utilizzare il secondo, chiamato anche E2POT, potenziometro digitale con memoria non volatile.

6.1 X9C104

Il modello preciso di potenziometro è il X9C104 (dove 104 va ad indicare il valore di resistenza lineare di valore massimo pari a $10\times 10^4\Omega$) non è altro che un array di 99 resistenze che possono essere collegate o scollegate mediante un contatore binario a 7 bit e una memoria non volatile per la memorizzazione dell'ultimo valore di resistenza impostato.

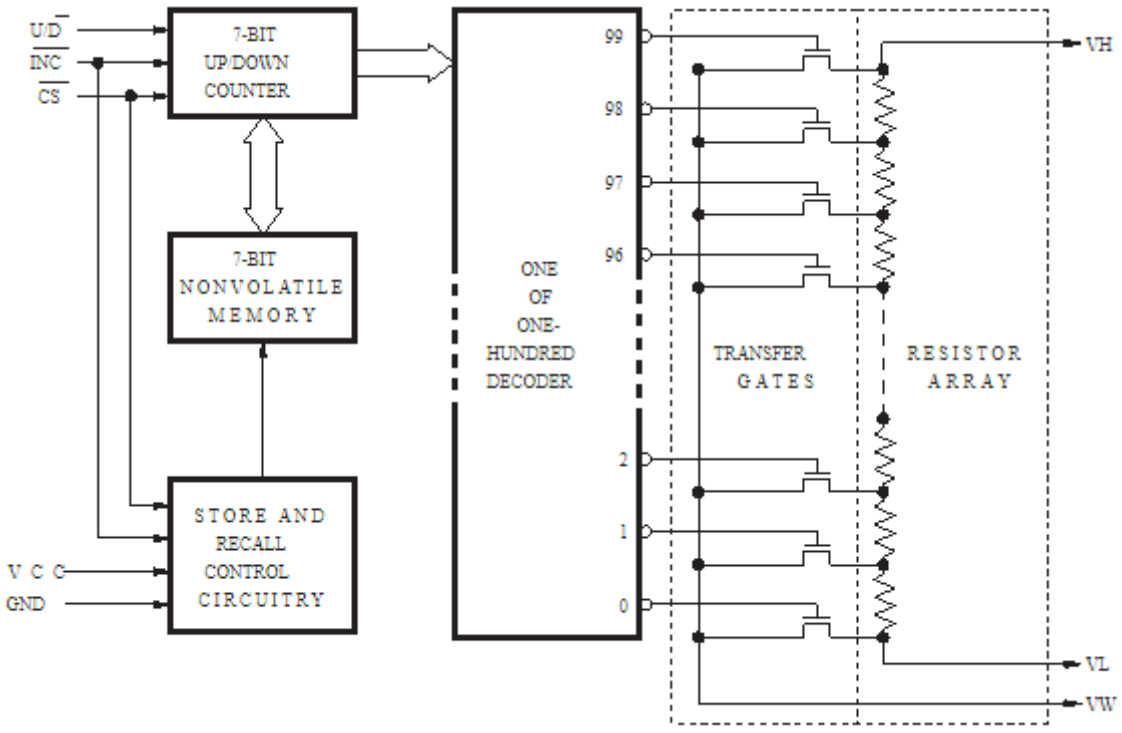


Figura 17:schema funzionamento interno potenziometro digitale

L'integrato è dotato di 3 pin di ingresso per pilotare il contatore, 2 pin per l'alimentazione e 3 pin (VH, VL, VW) per la resistenza variabile. Questi si comporteranno in maniera molto simile

ai 3 pin di un normale potenziometro. All'interno del circuito integrato c'è anche una memoria non volatile a 7 bit, utile per la memorizzazione dell'ultimo valore settato e per richiamarlo poi alla prima accensione. A differenza di un normale potenziometro, ha ovviamente bisogno di alimentazione per funzionare; in caso questa fosse assente tra i terminali resistivi si verifica una condizione di alta impedenza.

La risoluzione è pari alla resistenza massima / 99 quindi nel nostro caso 100K / 99 = 1010Ω.

I pin VH e VL sono rispettivamente il terminale alto e basso equivalenti ai terminali fissi del potenziometro. Il valore alto o basso serve solo a dare un riferimento al programmatore per capire la direzione nella quale si avrà un aumento o diminuzione di resistenza, non si riferisce alla tensione applicabile. La tensione massima applicabile è comunque di ±5V su entrambi i pin.

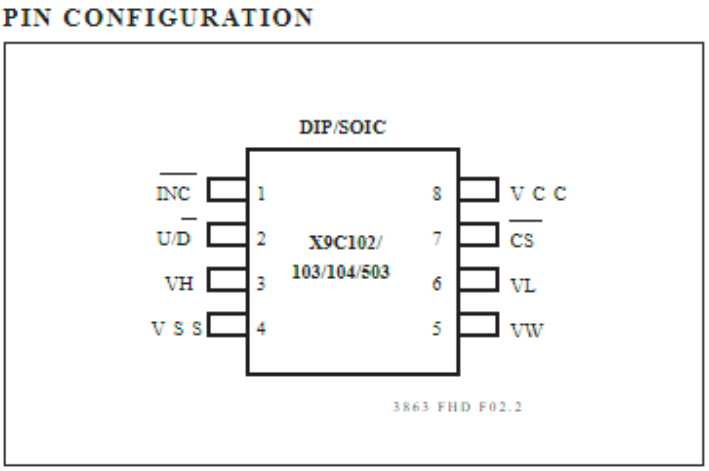


Figura 18: pinout dispositivo X9C104

Una breve spiegazione del funzionamento dei vari pin:

Vw è il wiper terminal, l'equivalente della parte mobile del normale potenziometro analogico. Il valore di resistenza tra questo terminale e gli altri due cambia in base a come viene comandato il contatore interno. Diversamente da un normale potenziometro quando si trova nella posizione minima presenta comunque una resistenza di 40Ω.

Up/Down controlla la salita (pin alto) o la discesa (pin basso) del wiper terminal, per il corretto funzionamento dipende anche dal valore dei restanti pin.

INC quando questo pin viene portato negativo (si attiva sui fronti di discesa) attiva il comando impostato sul terminale up down.

CS il chip è operativo quando questo pin è basso. Se invece si vuole memorizzare il la posizione di VW nella memoria non volatile il valore di CS deve tornare altro mentre INC è già alto.

MODE SELECTION

CS	INC	U/D	Mode
L	⌇	H	Wiper Up
L	⌇	L	Wiper Down
∫	H	X	Store Wiper Position
H	X	X	Standby Current
∫	L	X	No Store, Return to Standby

3863 PGM T06

Figura 19: comandi potenziometro digitale

Gli switch elettronici operano in modalità “make before break”; quando avviene un cambio di posizione, la resistenza totale può temporaneamente calare in maniera drastica se vengono spostati di posizioni multiple in poco tempo.

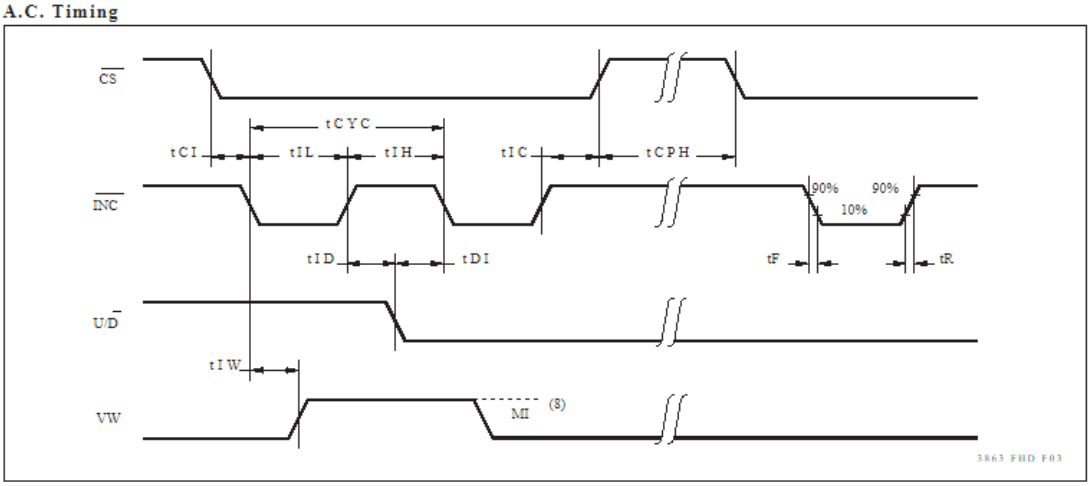
Per la realizzazione della libreria per pilotare il dispositivo sarà opportuno tenere conto del tempo minimo necessario affinché i vari segnali vengano correttamente riconosciuti.

A.C. OPERATING CHARACTERISTICS (Over recommended operating conditions unless otherwise specified)

Symbol	Parameter	Limits			Units
		Min.	Typ.(6)	Max.	
t_{CI}	CS to INC Setup	100			ns
t_{ID}	INC HIGH to U/D Change	100			ns
t_{DI}	U/D to INC Setup	2.9			µs
t_{IL}	INC LOW Period	1			µs
t_{IH}	INC HIGH Period	1			µs
t_{IC}	INC Inactive to CS Inactive	1			µs
t_{CPH}	CS Deselect Time	20			ms
t_{IW}	INC to Vw Change		100	500	µs
t_{CYC}	INC Cycle Time	4			µs
t_R, t_F(7)	INC Input Rise and Fall Time			500	µs
t_{PU}(7)	Power up to Wiper Stable			500	µs
t_R V_{CC}(7)	Vcc Power-up Rate	0.2		50	mV/µs

3863 PGM T07.3

A.C. Timing



Notes: (6) Typical values are for T_A = 25°C and nominal supply voltage.
(7) This parameter is periodically sampled and not 100% tested.
(8) MI in the A.C. timing diagram refers to the minimum incremental change in the Vw output due to a change in the wiper position.

Figura 20: temporizzazione segnali ingresso

Osservando il timing richiesto per operare correttamente il dispositivo, si può notare che per la maggior parte delle operazioni siamo negli ordini dei nanosecondi, con un picco anomalo per la memorizzazione della posizione del terminale che richiede un tempo minimo di 20 ms.
Per la realizzazione del filtro non è necessario avere una velocità elevata, ho deciso di prediligere l'affidabilità e utilizzare un tempo di assestamento di 1ms. In questo modo avrò la certezza che i vari comandi verranno letti correttamente mantenendo comunque una buona velocità generale del sistema.

ANALOG CHARACTERISTICS

Electrical Characteristics

End-to-End Resistance Tolerance	±20%
Power Rating at 25 °C	
X9C102	16mW
X9C103, X9C503, and X9C104	10mW
Wiper Current	±1mA Max.
Typical Wiper Resistance 40	Ω at 1mA
Typical Noise < –120dB/	√Hz Ref: 1V

Figura 21: caratteristiche elettriche

Oltre alla velocità è interessante analizzare le proprietà elettriche del dispositivo. Tra le caratteristiche che saltano subito all'occhio c'è la tolleranza della resistenza tra i morsetti VH e VL sia solo del 20% e la corrente massima che può assorbire il componente resistivo è di soli ±1mA. Valori certamente non da trascurare durante la progettazione circuitale, in caso di applicazioni ad alta precisione o con correnti elevate rendono obbligatoria la scelta di un dispositivo differente. Visto l'elevata escursione di resistenza ho pensato di fare una breve analisi del componente acquistato.
Dopo aver scritto un breve programma che incrementi il potenziometro digitale una volta ogni due secondi ho registrato e successivamente analizzato i dati in un foglio Excel. Il componente in mio possesso ha una resistenza totale di 105KΩ, maggiore del 5% rispetto a quella nominale con step che variano da un minimo di 960Ω a un massimo di 1100Ω l'uno dall'altro, la resistenza minima misurata è di 30 Ω. Un componente che certamente non è adatto per applicazioni di alta precisione, ma che comunque ha prestazioni migliori di quelle descritte nei datasheet.

6.2 PROGRAMMAZIONE POTENZIOMETRO DIGITALE

Per la configurazione del dispositivo ho pensato di scrivere una nuova classe; sarà molto utile per contenere tutte le informazioni per quanto riguarda la connessione dei pin, facilmente modificabile in un secondo momento e di immediata lettura una volta scritto il codice.

```
typedef struct
{
   GPIO_TypeDef * DP_GPIO;
   uint16_t UD_PIN;
   uint16_t INC_PIN;
   uint16_t CS_PIN;
}DP_GIPO;
```

Una volta creta la classe posso assegnare i pin, per il potenziometro del filtro (identificato dal prefisso DP) uso i pin PA1,2,3 mentre per il potenziometro utilizzato per variare il volume (identificato dal prefisso DR) uso i pin PA4,5,6.

```
const    DP_GIPO DP_GIPO_Cfg =
{
         GPIOA,
         GPIO_PIN_1,
         GPIO_PIN_2,
         GPIO_PIN_3,
};
```

In questo modo posso facilmente notare come un pin relativo all'incremento / decremento del potenziometro è collegato al pin PA1 del microcontrollore, il pin INC andrà collegato al pin 2 e così avanti per il resto. Se in futuro farò dei cambiamenti mi basterà modificare DP_GPIO_Cfg e il codice funzionerà senza bisogno di ulteriori modifiche. Per il potenziometro associato al cambio di guadagno ho invece deciso di usare il nome DR_GIPO_Cfg.
La prima funzione che ho scritto è quella per la configurazione iniziale del potenziometro, viene chiamata ogni volta che il dispositivo viene accesso o resettato. Si occupa di portare il potenziometro a metà corsa ed assegna alla variabile DP_pot il valore di 50. Ho pensato di usare questa variabile per tenere traccia del valore degli scatti e quindi al conseguente valore di resistenza al quale si trova il potenziometro digitale.

```
DP_pot = DP_setup();

int DP_setup(void){
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.UD_PIN,GPIO_PIN_SET);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
for(int i = 0 ; i < 100; i++){  //sposto la resistenza al valore massimo
      delay_ms(1);
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_RESET);
      delay_ms(10);
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
}
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.UD_PIN,GPIO_PIN_RESET);
      for(int i = 0 ; i < 50; i++){ //sposto a meta decrementando
         delay_ms(1);
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_RESET);
      delay_ms(10);
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
         }
         delay_ms(1);
         HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_SET);
         return 50;
}
```

Nella prima parte del programma, non avendo modo di sapere a quale valore di resistenza si trova attualmente il potenziometro all'accensione del dispositivo incremento di 100 tap in modo di essere sicuro che sia arrivato a fine corsa. In un secondo momento lo porto a metà corsa decrementando di 50 volte il contatore. Finito il processo di configurazione passo il valore 50 alla variabile DP_pot e abbasso il CS portando il dispositivo in standby. E' molto importante arrivare a fine corsa alto e non basso per evitare che scorra una corrente troppo elevata che possa danneggiare il dispositivo. Dopo aver settato i pin attendo che il segnale si assesti; ho usato come tempo di assestamento 1ms, valore ben sopra le specifiche da datasheet per essere sicuro che nessun segnale venga perso. Inoltre, non c'è necessità di avere una velocità maggiore, il sistema risulta abbastanza reattivo anche con queste tempistiche.

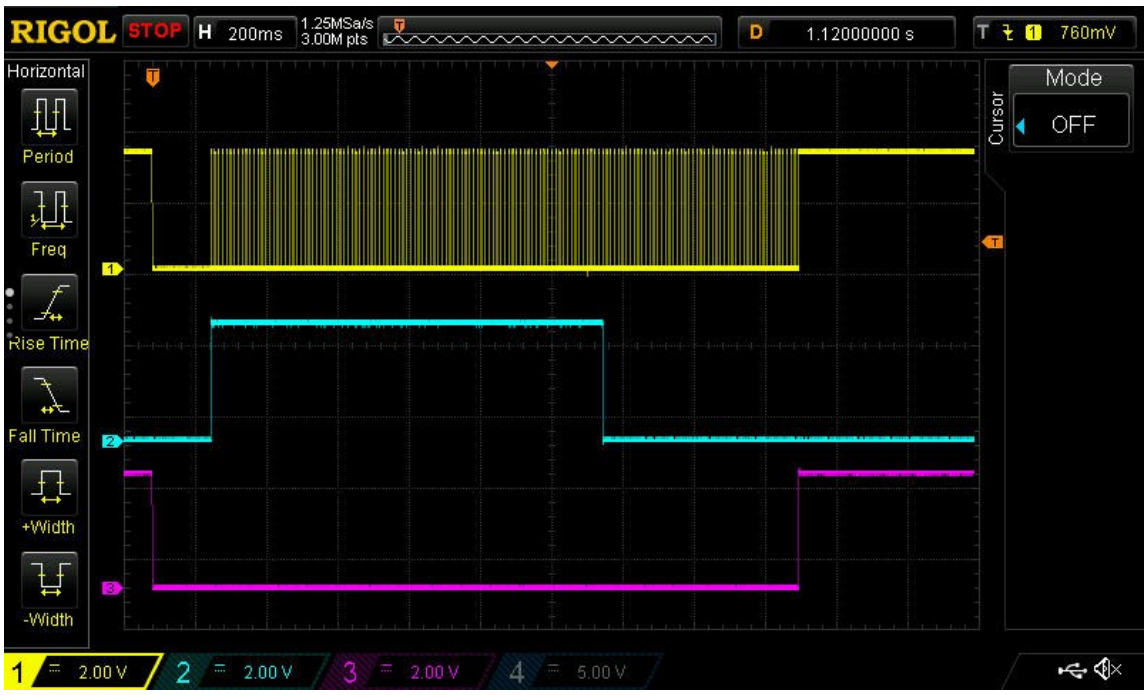


Figura 22: segnali uscita microcontrollore

Analizzando con l'oscilloscopio la fase iniziale di setup posso vedere come il segnale di CS (viola) vada basso, il segnale di UP/Down (blu) vada alto per iniziare un incremento e infine il segnale INC (in giallo) inizia a mandare i 100 impulsi per mandare il potenziamento digitale a fine corsa. Dopo 100 colpi il PIN UP/DOWN torna basso per portare il potenziometro a metà corsa. Alla fine del processo il pin del CS torna alto andando a memorizzare la posizione del potenziometro nella memoria interna ed entrando in modalità stand by.

Per verificare che tutto fosse corretto ho analizzato la temporizzazione dei segnali.

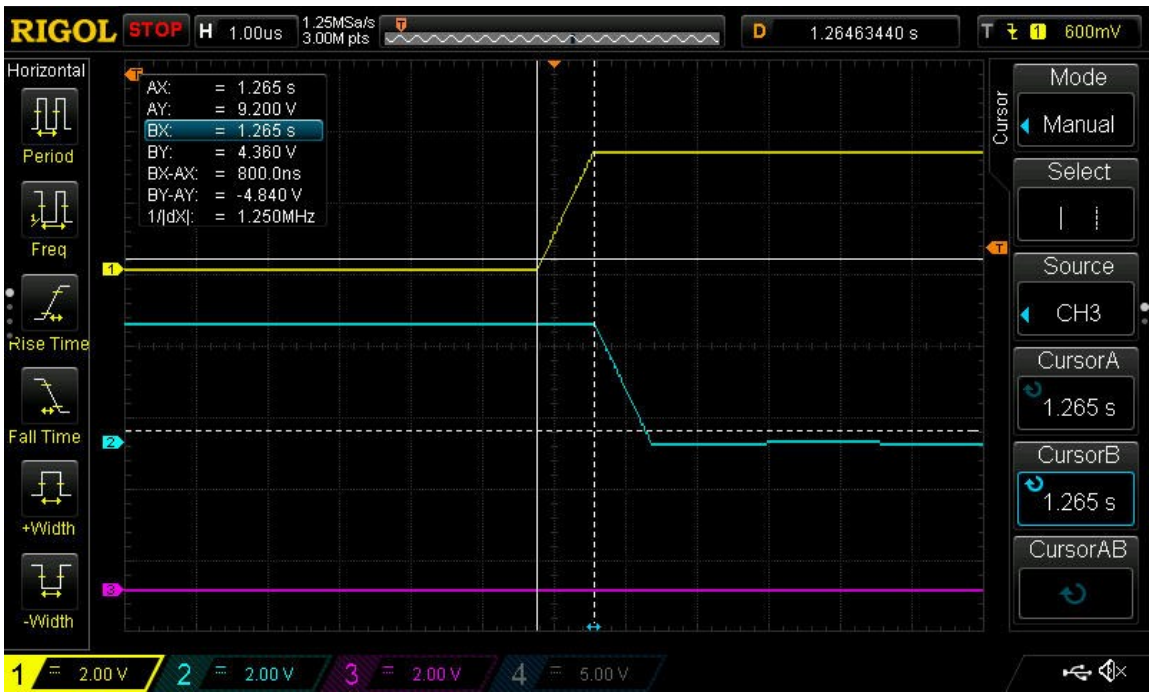


Figura 23: zoom segnali uscita microcontrollore

Facendo uno zoom sul passaggio da positivo a negativo del PIN U/D posso misurare un delay di 800ns, ben sopra le specifiche. Dopo le seguenti misurazioni posso constatare che la funzione scritta risulta efficace per comandare il potenziamento digitale. Affinché avvenga il setup completo ci vogliono 1.8s; non un tempo irrisorio ma comunque accettabile, considerato che la finzione verrà chiamata una volta sola all'accensione del dispositivo.

La seconda funzione che ho scritto non viene utilizzata nel progetto corrente, ma è stata realizzata per completare le funzioni base del dispositivo. Tramite la chiamata di questa funzione è possibile raggiungere il valore di resistenza desiderato. Per il corretto funzionamento bisogna passare il valore in Ω tramite una variabile di tipo int e il valore corrente del potenziometro tramite la variabile DP_pot che verrà passata per indirizzo in modo da poterne aggiornare il valore.

```
void DP_setvalue (int value,int *a){
   int value_step = 0;
   int curr_value = 0;
   curr_value = *a;
   value_step = value /1060;
   if ( value_step > 100 ) return;
   else{
         if (value_step == curr_value) return;
         else{
                  //preparo i pin
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
if (value_step < curr_value ){
  HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.UD_PIN,GPIO_PIN_RESET);
  for(int i = 0 ; i < curr_value - value_step; i++){
      delay_ms(1);
       HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_RESET);
       delay_ms(10);
       HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
}
       *a = value_step;
```

```
      delay_ms(1);
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_SET);
      return;
    }
     else{
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.UD_PIN,GPIO_PIN_SET);
   for(int i = 0 ; i < value_step - curr_value; i++){
      delay_ms(1);
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_RESET);
      delay_ms(10);
      HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
         }
         *a = value_step;
         delay_ms(1);
         HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_SET);
         return;
       }
     }
  }
}
```

Come prima cosa ricavo il numero di step che devo applicare per raggiungere il valore di resistenza impostato, per farlo divido per 1060 che è il valore medio di uno step misurato sul potenziometro in esame. Segue una fase di controllo che il valore inserito non sia fuori scala o che non sia già settato, dopodiché la funzione procede a incrementare o decrementare la resistenza fino al raggiungimento del valore desiderato. Al termine del ciclo viene aggiornato il valore di DP_pot e memorizzato il valore di resistenza corrente nella memoria non volatile del dispositivo.

```
void DP_UP(int *a){ //incrementa il dato
if (*a<100){
  HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.UD_PIN,GPIO_PIN_SET);
  HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_RESET);
  HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
  delay_ms(1);
  HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_RESET);
  delay_ms (10);
```

```
  HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
          *a=*a +1;
 delay_ms(1);
 HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_SET);
          }
else {
        *a = 99;
         return;
        }
}

void DP_DOWN(int *a){ //decrementa il dato
if (*a >= Down_limit ){
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.UD_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
delay_ms(1);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_RESET);
HAL_Delay(10);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.INC_PIN,GPIO_PIN_SET);
*a=*a - 1;
delay_ms(1);
HAL_GPIO_WritePin(DP_GIPO_Cfg.DP_GPIO,DP_GIPO_Cfg.CS_PIN,GPIO_PIN_SET);
}
else {
         *a = Down_limit;
         return;
        }
}
```

Queste ultime funzioni non hanno bisogno di spiegazioni approfondite, la prima incrementa di una la posizione del potenziometro, la seconda invece la diminuisce. Ho pensato di aggiungere un limite inferiore Down_limit nel caso non si volesse oltrepassare certi limiti di resistenza, utili ad esempio per variare la frequenza di taglio minima raggiungibile dal filtro passa basso.

7. AMPLIFICATORE OPERAZIONALE TL074CE

Questo integrato, nonostante non sia tra i più recenti sul mercato dispone di amplificatori operazionali a basso rumore, ottima banda, slew rate veloce e una bassa distorsione armonica rendendolo un'ottima alternativa per applicazioni audio ad alta fedeltà.

- Low Input Noise Voltage: 18 nV/ $\sqrt{Hz}$ Typ•
- Low Harmonic Distortion: 0.01% Typ•
- Low Input Bias and Offset Currents•
- High Input Impedance: 1012Ω Typ•
- High Slew Rate: 13 V/µs Typ•
- Wide Gain Bandwidth: 4.0 MHz Typ•
- Low Supply Current: 1.4 mA per Amp

Si possono trovare in diversi formati, con uno, due o quattro operazionali, io utilizzerò quest'ultimo formato con packaging 646, quindi utilizzabile facilmente su breadboard per la prototipazione e poi successivamente inserito nel circuito tramite l'apposito socket Plastic DIP.

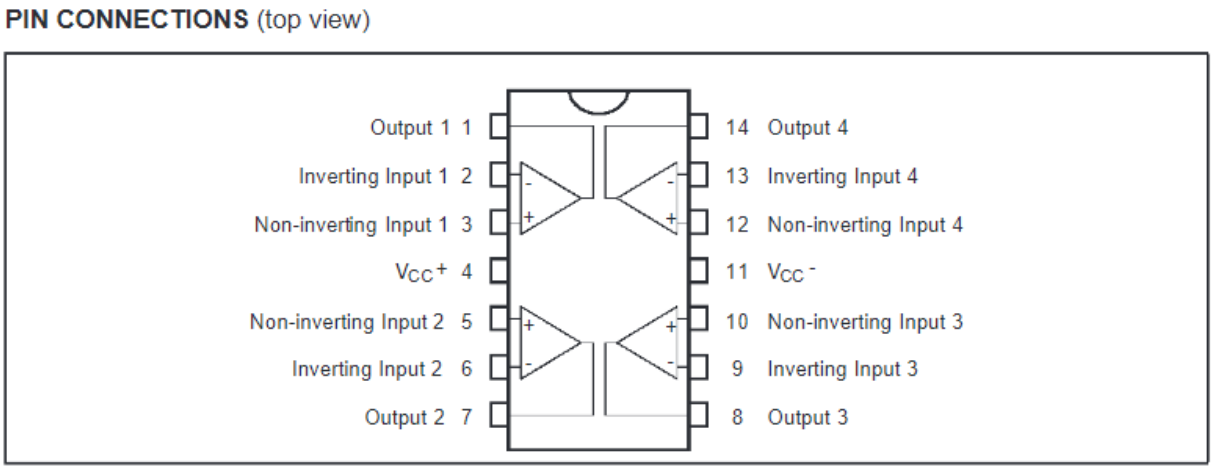


Figura 24: pinout amplificatore operazionale TL074CE

Considerando che il potenziometro digitale funziona a 5V per l'alimentazione, ho pensato di usare ±5V. Nonostante il packaging disponga di 4 operazionali, per la realizzazione del circuito ne userò solamente due. Visto che tutti gli operazionali sono sullo stesso packaging possono interferire tra di loro, per evitare comportamenti anomali gli ingressi inutilizzati sono stati collegati a massa.

Il range di funzionamento del circuito al quale viene utilizzato si trovano ben sotto le specifiche massime del componente riportate nel datasheet.

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	TL074M, AM, BM	TL074I, AI, BI	TL074C, AC, BC	Unit
V_{CC}	Supply voltage - note 1)	±18			V
V_i	Input Voltage - note 2)	±15			V
V_{id}	Differential Input Voltage - note 3)	±30			V
P_{tot}	Power Dissipation	680			mW
	Output Short-circuit Duration - note 4)	Infinite			
T_{oper}	Operating Free-air Temperature Range	-55 to +125	-40 to +105	0 to +70	°C
T_{stg}	Storage Temperature Range	-65 to +150			°C

1. All voltage values, except differential voltage, are with respect to the zero reference level (ground) of the supply voltages where the zero reference level is the midpoint between V_{CC}^+ and V_{CC}^-.
2. The magnitude of the input voltage must never exceed the magnitude of the supply voltage or 15 volts, whichever is less.
3. Differential voltages are the non-inverting input terminal with respect to the inverting input terminal.
4. The output may be shorted to ground or to either supply. Temperature and/or supply voltages must be limited to ensure that the dissipation rating is not exceeded

Figura 25: caratteristiche elettriche dell'OPAMP

7.1 IMPLEMENTAZIONE CIRCUITALE

Dei quattro operazionali presenti ne utilizzerò due. Uno verrà configurato come buffer mentre il secondo verrà utilizzato come amplificatore invertente per modificare il guadagno del filtro.

Buffer:

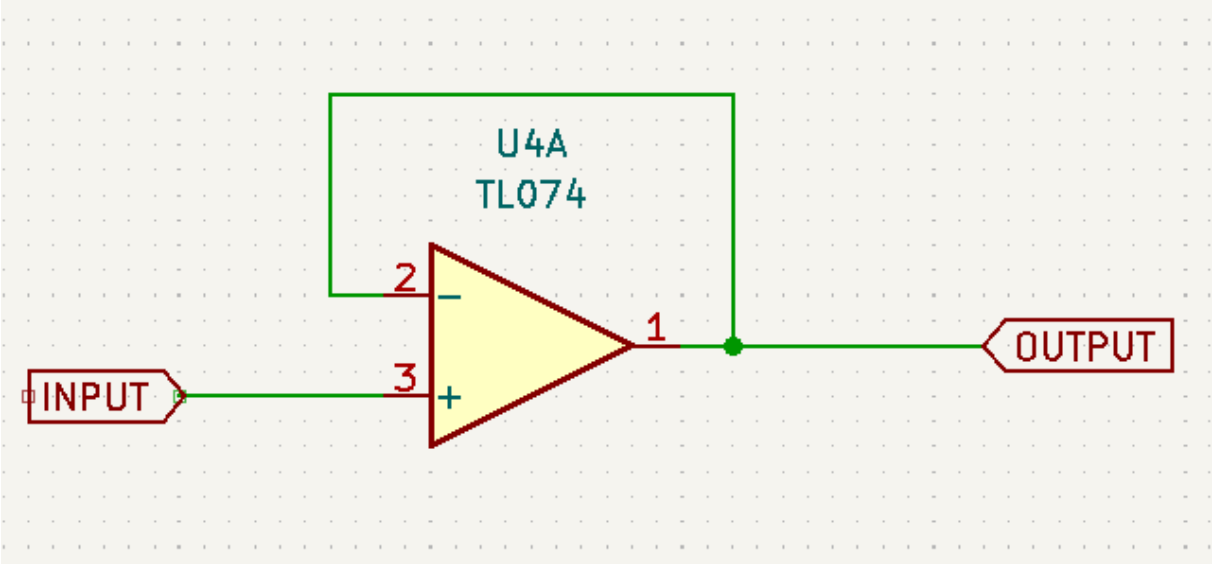


Figura 26: circuito buffer

Utilizzando un operazionale in questa modalità si isola il circuito di filtro da quello di guadagno, l'ingresso ad alta impedenza dell'amplificatore limita la corrente che scorre sulla resistenza del potenziometro digitale evitando che il circuito a valle vada a influire sul segnale in ingresso.

Il funzionamento dell'operazionale in questa configurazione è piuttosto semplice, viene utilizzata la proprietà di corto circuito virtuale. L'uscita è connessa all'ingresso invertente, in questo modo una variazione di tensione sull'ingresso non invertente porta uno squilibrio tra i pin positivo e negativo, l'uscita andrà quindi ad aumentare o diminuire in base al segno di questa differenza di potenziale. Utilizzando la retroazione negativa ad ogni perturbazione dell'ingresso l'integrato aggiusterà l'uscita per portare il sistema in equilibrio e quindi ad inseguire l'ingresso per portare a zero la differenza di potenziale tra i morsetti in ingresso.

Amplificatore invertente:

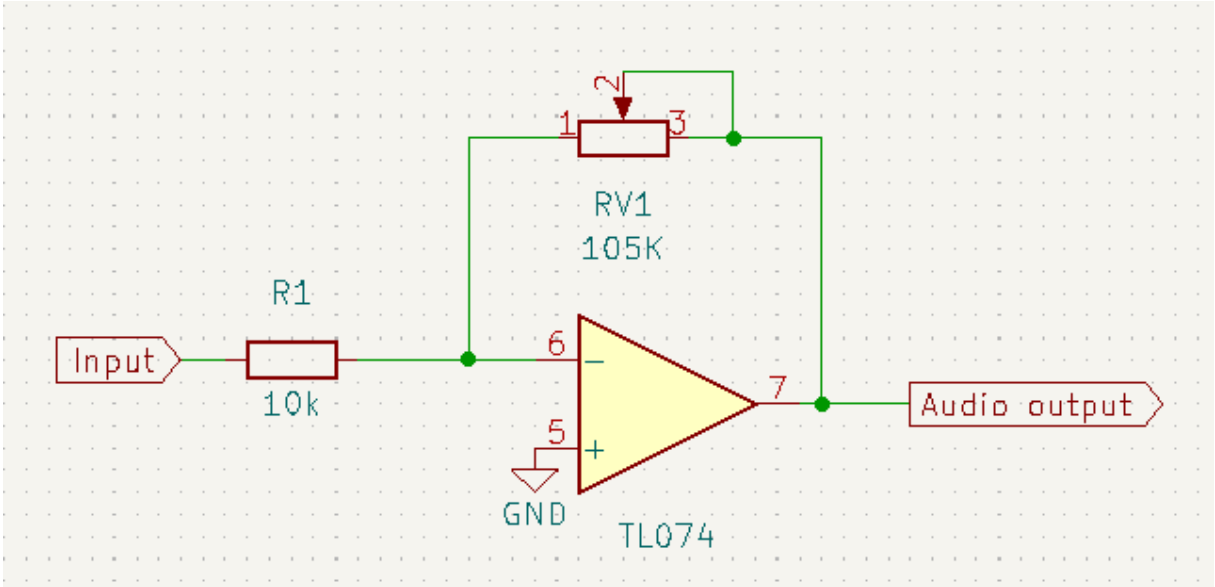


Figura 27: circuito amplificatore invertente

Per la realizzazione del circuito che si occupa di variare guadagno e quindi poter variare l'ampiezza del segnale in uscita ho utilizzato un amplificatore operazionale in modalità invertente. In questa configurazione l'uscita è collegata al morsetto invertente tramite una resistenza variabile, in questo modo avremo una configurazione ad anello chiuso con retroazione negativa con la conseguenza che il segnale di uscita avrà uno sfasamento di -180° rispetto all'ingresso.

Per il calcolo del guadagno $G = \frac{V_{out}}{V_{in}}$ del circuito e la scelta della resistenza si procede analizzando le formule che relazionano la tensione con la corrente. Per il corretto funzionamento del dispositivo supponiamo che l'operazionale lavori principalmente all'interno della regione di non saturazione, la differenza di potenziale v_+-v_- è degli ordini dei µV e quindi trascurabile, possiamo quindi assumere che $v_+ = v_- = 0$V. Con questa considerazione è facile ricavare la reazione tra la corrente che scorre in R_{V1} e $V_{out} = R_{V1} i_{RV1}$. Il medesimo

ragionamento si può fare sulla tensione in ingresso V_{in} = -$R_1 i_{R1}$. Ora che ho ricavato le formule delle tensioni le posso inserire all'interno della formula del guadagno ottenendo così la relazione tra le resistenze e la corrente che scorre tra di esse $G = -\frac{R_{V1} i_{RV1}}{R_1 i_{R1}}$. Considerando l'alta impedenza in ingresso dell'amplificatore operazionale e considerando i valori letti sul datasheet del componente vedo che nei pin v_+ e v_- scorre una corrente degli ordini dei nA, molto minore rispetto quella che scorre nelle resistenze, può quindi essere trascurabile nella configurazione corrente. Fatte le seguenti considerazione posso concludere che $i_{RV1} \simeq i_{R1}$ e quindi osservare che il guadagno totale dipende solo dal valore della resistenza utilizzate $G = -\frac{R_{V1}}{R_1}$ (il segno negativo indica uno sfasamento ingresso uscita di -180°). Per rispettare le specifiche di progetto e volendo quindi impostare un guadagno massimo di 10 è facilmente ricavabile il valore della resistenza R_1= $\frac{R_{V1MAX}}{10}$=10.5KΩ. Scegliendo quindi il valore commerciale che più si avvicina a quello nominale ho una resistenza da 10KΩ. Analizzando il guadagno sull'intera estensione del potenziometro digitale ho un massimo di 10.5 quando R_{V1}=105KΩ e raggiungo il minimo di 0.003 dato che il potenziometro digitale non può scendere sotto i 30Ω.

Un fattore importante da considerare è la corrente massima che scorre nel potenziometro, per farlo prendo in esame un caso anomalo dove la tensione in ingresso è 5Vpp, ben sopra i 2Vpp da specifica, mentre il potenziometro è al suo valore minimo raggiungibile di 30 Ω. Facendo rapidamente due conti ricavo che il guadagno in queste condizioni è $\frac{30}{10000} = 0{,}003$, di conseguenza la tensione in uscita sarà quindi 5×0.003 = 0.015Vpp. La massima corrente che attraverserà il potenziometro sarà quindi $\frac{0.015V}{30} = 5 \times 10^{-4} A$, valore molto inferiore a 1mA massimo consentito da specifica. Andando ad aumentare il guadagno andrà ad aumentare proporzionalmente anche la resistenza del potenziometro lasciando la corrente invariata. Una volta che il guadagno sarà sufficientemente grande l'operazionale raggiungerà il valore di saturazione di 5V, la resistenza continuerà a crescere lasciando invariata la tensione e diminuendo la corrente che lo attraversa. In conclusione, posso confermare che il valore di resistenza trovato sia conforme alle specifiche richieste

8. RICEVITORE INFRAROSSI

Visto la semplicità di utilizzo e il basso costo ho deciso di implementare una comunicazione wireless tramite telecomando ad infrarossi. Per la ricezione e decodifica del segnale emesso da un telecomando ho utilizzato un ricevitore a infrarossi; esternamente sembra un LED con tre pin, ma internamente troviamo un sistema più complesso. Sfogliando il datasheet si nota come il funzionamento non sia così semplice: c'è un filtro passa banda, un integratore e un demodulatore con guadagno automatico che fanno sì che il segnale in uscita dal ricevitore sia privo di disturbi pronto per essere letto dal microcontrollore.

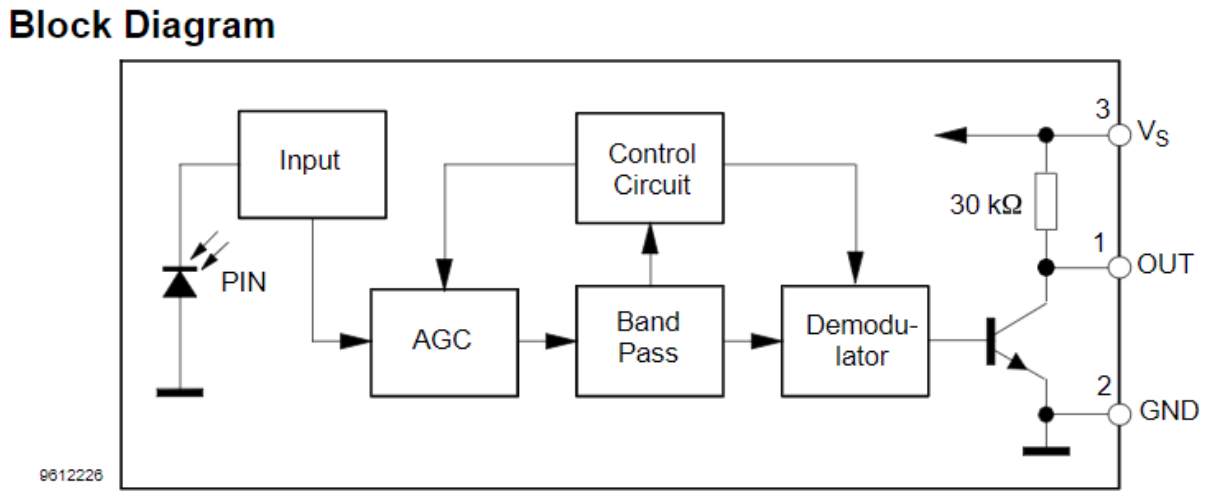


Figura 28: componenti interni ricevitore ad infrarossi

L'ingresso verrà alimentato a 5V quindi anche l'uscita sarà a 5V massimi; sarà quindi importante collegarlo ad un pin del microcontrollore capace di sopportare quella tensione senza rompersi.

In natura ci sono molteplici fonti luminose ad infrarossi che andrebbero a disturbare il segnale del telecomando, affinché il dispositivo riesca a isolare e rimuovere il rumore è importante che il trasmettitore segua dei parametri precisi. Per avere una corretta demodulazione si è deciso di trasmettere con una frequenza modulante di 38 KHz, la meno presente in natura e meglio isolabile dalla luce artificiale.

Il protocollo di comunicazione IR più utilizzato per la codifica dei dati è il NEC; per segnalare l'inizio della trasmissione viene trasmesso un impulso della durata di 560us seguito da una pausa e poi nuovamente l'impulso di prima. Se il tempo totale di trasmissione è 2.25ms allora è stato trasmesso un 1, se la durata è invece di 1.12ms si trasmette uno 0. Per la corretta trasmissione il telecomando ha una frequenza modulante di 38.222kHz.

Quello che risulta maggiormente interessante per la ricezione è come si presenta il segnale demodulato all'uscita del sensore ad infrarossi.

Prima viene trasmesso un codice che comunica l'inizio della trasmissione, il segnale va basso per 9 ms, dopo torna alto per altri 4,5 ms segnalando che inizia la trasmissione vera e propria dei bit. Per distinguere gli uni dagli zeri viene semplicemente usato un sistema di temporizzazione, se un impulso dura 1,687 ms allora si tratta di un uno, in caso duri 562.5µs allora è zero. Nel protocollo NEC vengono trasmessi 8 bit di indirizzo seguiti da 8 bit negati, 8 bit di dati e infine 8 bit negati per un totale di 32 bit.

Ho analizzato il segnale di un telecomando di un amplificatore della Onkyo che avevo già presente in casa, il protocollo è molto simile al NEC, ma con delle leggere differenze. Ad ogni modo, la parte interessante è leggere i bit in trasmissione per poi memorizzarli nel microcontrollore per poi successivamente riconoscere quale tasto viene poi premuto.

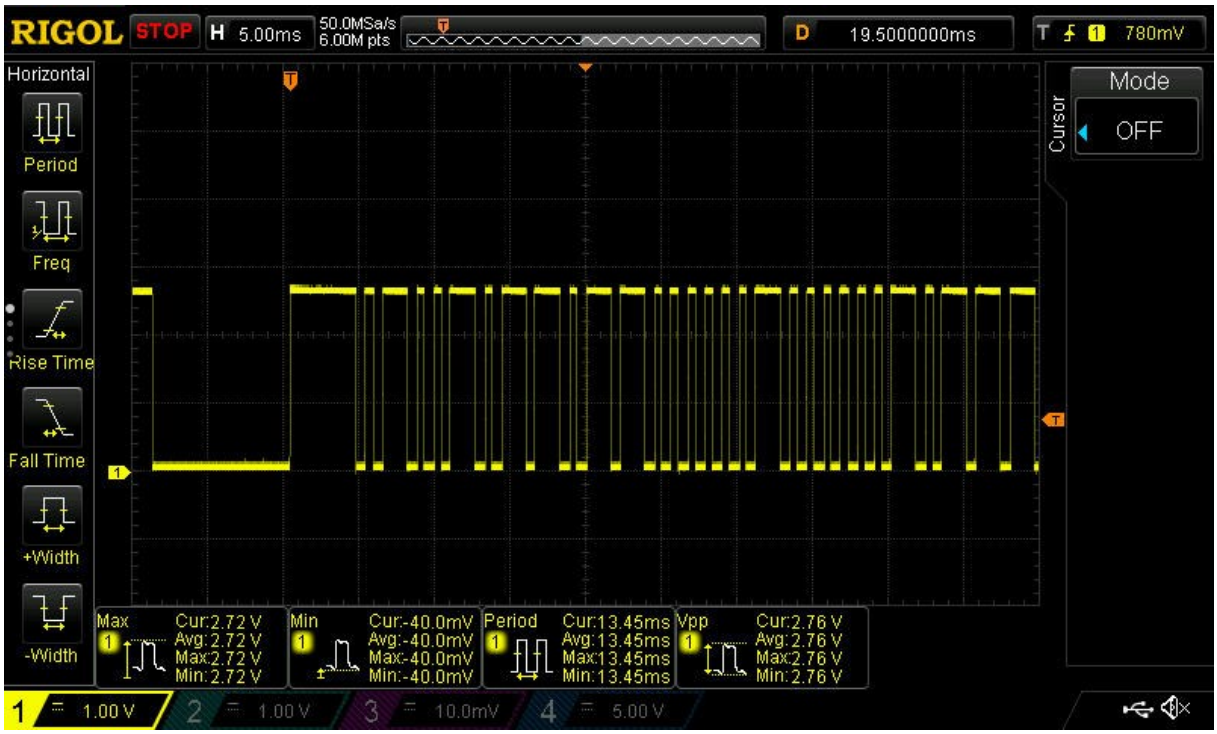


Figura 29: segnale uscita ricevitore a infrarossi

Prendiamo ad esempio il codice ricevuto dopo la pressione del tasto volume più visibile nella figura 29. Si può chiaramente notare che all'inizio abbiamo 9ms di segnale negativo seguito da 4,5ms di segnale positivo per segnalare l'inizio della trasmissione. Dopodiché, andando a decodificare impulso lungo con uno e impulso corto con zero posso trascrivere il seguente dato a 32 bit 01001011011000000100000010111111 che tradotto in esadecimale viene 0x4B6040BF. Eseguo lo stesso procedimento per gli altri tasti che verranno utilizzati.

8.1 PROGRAMMAZIONE RICEVITORE AD INFRAROSSI

Ora che ho il dato trasmesso dal telecomando e so come decifrarlo devo trascrivere una funzione che lo decodifichi e poi esegua un preciso comando un base al tasto premuto.
Collego il pin di uscita del ricevitore a infrarossi al pin PA9 precedentemente impostato come input. All'interno del main scrivo una funzione che resta in attesa dell'inizio del segnale di inizio frame. In condizioni di riposo il pin di uscita del segnale ad infrarossi rimane alto, il programma rimane quindi bloccato all'interno del while.

```
while (HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_9));   // aspetto l'inizio del frame
                data = receive_data();
                convert_code (data);
```

Appena però il ricevitore capta un segnale porta il pin di uscita basso e il programma assegna alla variabile data l'output della funzione receive_data() che si occupa della decodifica del segnale ricevuto. In un secondo momento viene chiamata la funzione convert_code() utile per eseguire il comando associato al tasto premuto.

```
uint32_t receive_data (void)
{
        uint32_t code=0;
          while (!(HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_9)));
          while ((HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_9)));

          for (int i=0; i<32; i++)
          {
                  count=0;
                  while (!(HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_9)));
                  while ((HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_9)))
                  {
                          count++;
                          delay_us(100);
                  }
                  if (count > 12)
                  {
                          code |= (1UL << (31-i));   // asseggno 1 nella posizione corretta
                  }
```

```
                else code &= ~(1UL << (31-i));  // assegno 0 alle posizione corretta
        }
                return code;
}
```

La prima parte serve solo per identificare l'inizio della trasmissione; utilizzo dei while per attendere l'arrivo della sequenza di impulsi da decodificare. Una volta entrato nel ciclo for con una serie di while conto quanto tempo il segnale rimane alto; se supera gli 1.2 ms allora assegno 1, in caso contrario assegno uno zero. Per assegnare il bit nella posizione corretta eseguo uno shift di 31-i posizioni. L'operatore |= indica un bitwise or mentre &= bitwise and. Usando ~(1UL << (31-i)) inverto i bit, in questo modo ho tutti uni tranne lo 0 nella posizione che mi interessa.

```
void convert_code (uint32_t code)
{
                switch (code)
                {
                        case (0x4B6040BF) //tasto volume su
                                DP_UP(&DP_pot);
                        break;
                        case (0x4B60C03F): //tasto volume giu
                                DP_DOWN(&DP_pot);
                        break;
                        case (0x4BE042BD): //tasto frequenza su
                                DR_UP(&DP_pot);
                        break;
                        case (0x4BE0C23D): //tasto frequenza giu
                                DR_DOWN(&DP_pot);
                        break;
                        default :
                                data=0;
                        break;
                }
                data=0;
                delay_ms(10);
}
```

Ogni volta che viene chiamata la funzione vuol dire che è stato premuto un tasto sul telecomando. La funzione controlla che sia stato premuto uno dei quattro tasti programmati e in caso positivo ne esegue il comando. Per ultimo, si esegue il reset del tasto in modo da evitare che vengano eseguite funzioni anche senza la pressione di un tasto. Ho ritenuto comodo inserire un delay nel caso l'utente volesse tenere premuto il tasto per eseguire operazioni multiple. Aggiungere una piccola pausa tra una operazione e l'altra ne migliora di molto il controllo.

9. FILTRO PASSA BASSO DEL PRIMO ORDINE

Per filtrare il segnale di ingresso e dividere le basse frequenze dagli alti ho optato per l'utilizzo di un filtro passa basso passivo del primo ordine. Il prezzo contenuto e la facilità di utilizzo la rendono una soluzione efficace. Il filtro è realizzato usando solamente un condensatore a film non polarizzato e il potenziometro digitale come resistenza variabile.

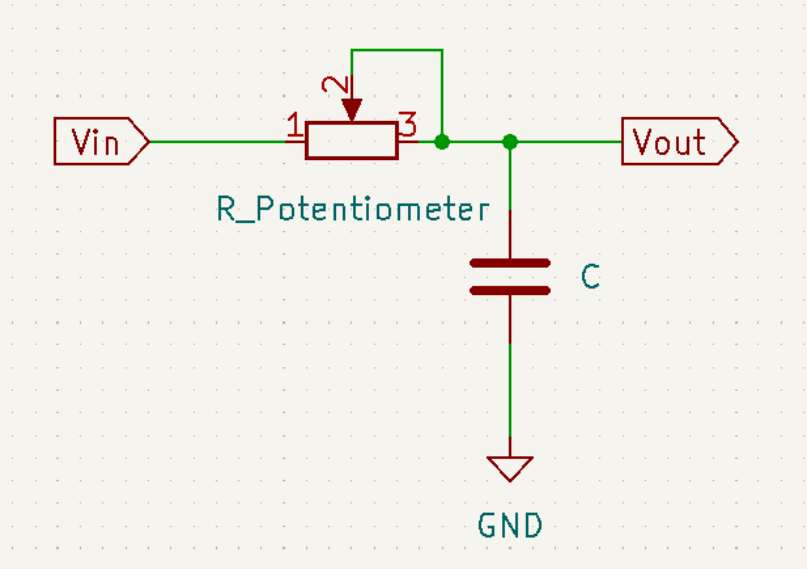


Figura 30: filtro passa basso con potenziometro

Una parte fondamentale nella progettazione del circuito sta nella corretta scelta del valore del condensatore avendo il già scelto il valore della resistenza. Come prima cosa scriviamo la funzione di trasferimento del sistema

$$V_{out}(\omega) = V_{in}\frac{\frac{1}{j\omega C}}{R+\frac{1}{j\omega C}} = V_{in}(\omega)\frac{1}{1+J\omega RC}$$

Semplificando ponendo $\tau = RC$ ottengo la finzione di trasferimento:

$$G(\omega) = \frac{1}{1+J\omega\tau}$$

Lo posso anche scrivere come:

modulo: $|G(\omega)| = \frac{1}{\sqrt{1+\omega^2\tau^2}}$

e fase: $\varphi(\omega) = -\arctan(\omega\tau)$

Si può notale che sono entrambe funzioni monotone decrescenti, andando quindi a fare il limite per ω che tende a infinito il modulo tenderà a zero, da qui il nome passa basso. Dalla funzione della fase si può invece notare come lo sfasamento massimo sia di $-\frac{\pi}{2}$

Per lo scopo di questo progetto interessa trovare la frequenza di taglio (per comodità in Hertz) oltre la quale il modulo viene ridotto di 3dB, per farlo pongo $|G(2\pi f_t)| = \frac{1}{\sqrt{2}}$ quindi $f_t = \frac{1}{2\pi\tau} = \frac{1}{2\pi RC}$. La resistenza del potenziometro digitale può variare tra un valore massimo di 105KΩ e un minimo di 30Ω. Si vuole raggiungere una frequenza di taglio compresa tra i 70 e i 200Hz, per la scelta del condensatore basta invertire la formula e inserire i valori di resistenza e frequenza voluti. Facendo i conti con una resistenza di 105KΩ e in condensatore da 30nF si raggiunge una frequenza di taglio di 50Hz. Se però si va a guardare il valore minimo raggiungibile, ovvero con la resistenza di 30Ω la frequenza salirebbe a ben 176 KHz ben fuori scala. Per ovviare a questo problema imposterò un limite inferiore via software di 20KΩ in modo da raggiungere una frequenza di 265Hz senza andare a sacrificare troppo la risoluzione.
Il valore di condensatore più vicino a quello nominale che sono riuscito a trovare è di 33nF con tolleranza del 10%

10. CIRCUITO FINALE

Il circuito finale si può vedere in figura 31. Rappresenta il minimo indispensabile per il funzionamento indicato del dispositivo.

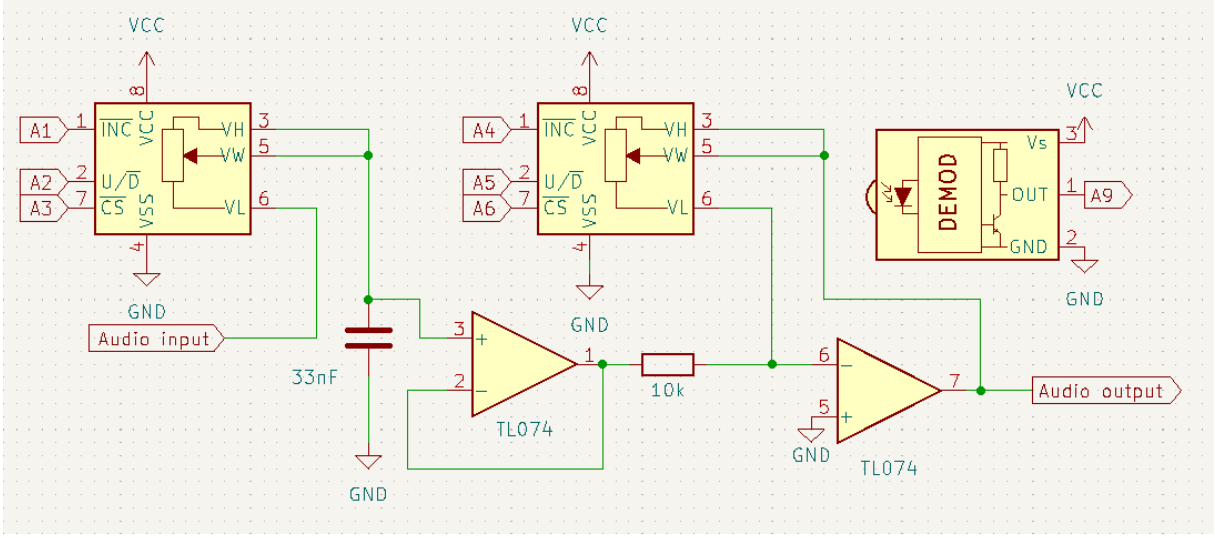


Figura 31: circuito completo del progetto

Sicuramente è possibile migliorare il circuito aggiungendo condensatori vicino alle alimentazioni per ridurre al minimo la possibilità di ripple sulle alimentazioni. Può essere inoltre una buona idea aggiungere un ulteriore filtro passa alto con frequenza di taglio molto bassa per rimuovere le eventuali tensioni continue, dovute ad esempio ad una saturazione dell'operazionale che potrebbero danneggiare un diffusore. Questa funzione, tuttavia, spesso è integrata nell'ingresso di un eventuale amplificatore da collegare tra il subwoofer e l'uscita del filtro variabile. Nonostante sia importante assicurarsi che non ci siano tensioni continue sul diffusore aggiungere un filtro aggiuntivo può essere un costo inutile se già presente nello stadio successivo.

Al fine di eseguire i test di funzionamento, il circuito è stato assemblato su una scheda perforata e la tensione è stata fornita da un alimentatore esterno. In un prodotto finale un opportuno circuito di alimentazione è obbligatorio. L'alimentazione negativa è stata fornita da un alimentatore a +5V collegando il morsetto positivo alla massa e il morsetto negativo al pin -5V. Per avere un segnale pulito come uscita dall'amplificatore operazionale è importante avere un alimentatore di qualità; mi è capitato di usarne uno molto economico e l'uscita presentava parecchio rumore.

11. ANALISI DELLE PRESTAZIONI

Una volta realizzato il prototipo del circuito ho realizzato alcuni test. Tramite l'utilizzo di un computer ho usato una funzione all'interno del programma REW che permette di generare attraverso la scheda audio del pc un segnale sinusoidale a frequenza impostabile dall'applicazione. Prima ho impostato il potenziometro della frequenza al valore massimo in modo da avere una frequenza di taglio nominale intorno ai 50Hz e quello del volume a 10KΩ per ottenere un guadagno unitario. Misurando con una sonda dell'oscilloscopio del sistema ottengo la seguente forma d'onda visibile in figura 32. Tramite il computer genero una sinusoide con frequenza di 10Hz e ampiezza 1 V, poi collego l'uscita della scheda audio direttamente al filtro variabile tramite un jack audio da 3.5 mm. Collegando la sonda dell'oscilloscopio all'uscita del sistema visualizzo la seguente onda:

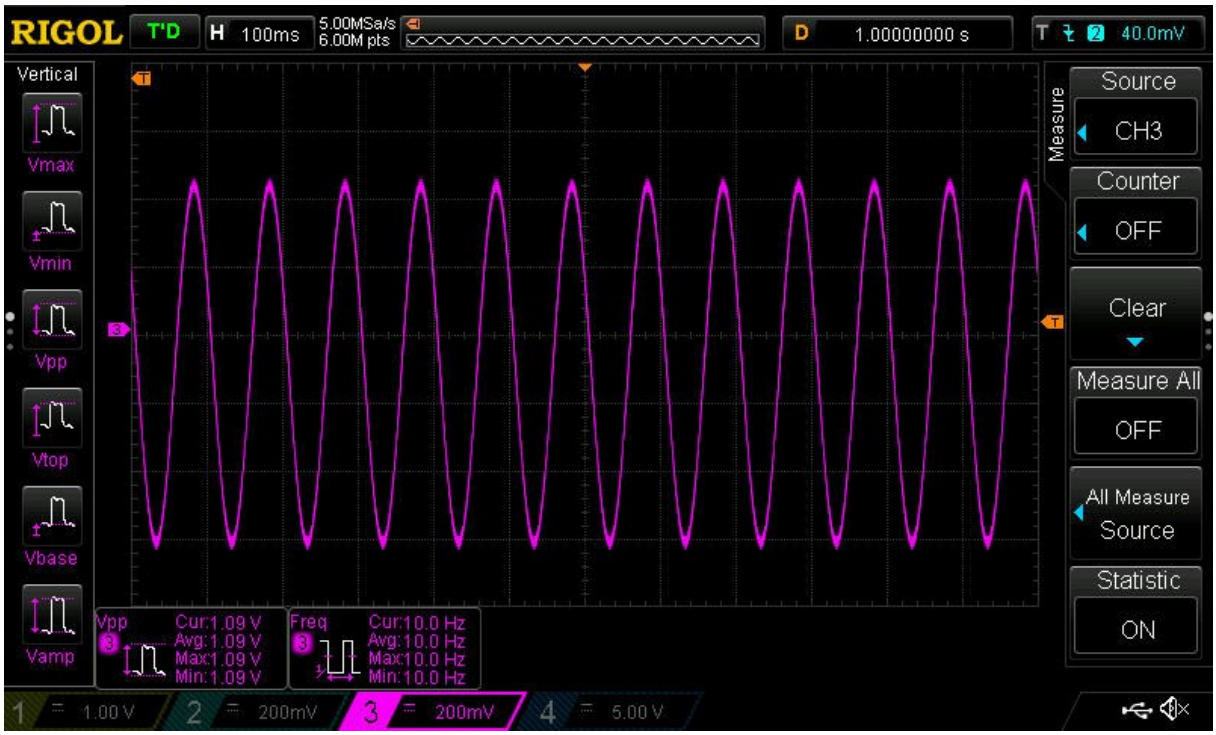


Figura 32:uscita del sistema con segnale in ingresso in centro banda

Attivando la misurazione automatica dei parametri posso osservare come la frequenza sia molto precisa mentre l'ampiezza è leggermente più alta, ma questo può essere dovuto alle tolleranze della resistenza.

Per misurare la frequenza di taglio incremento la frequenza del seno in ingresso finché non vedo una riduzione dell'ampiezza in uscita del 30% quindi 732mV.

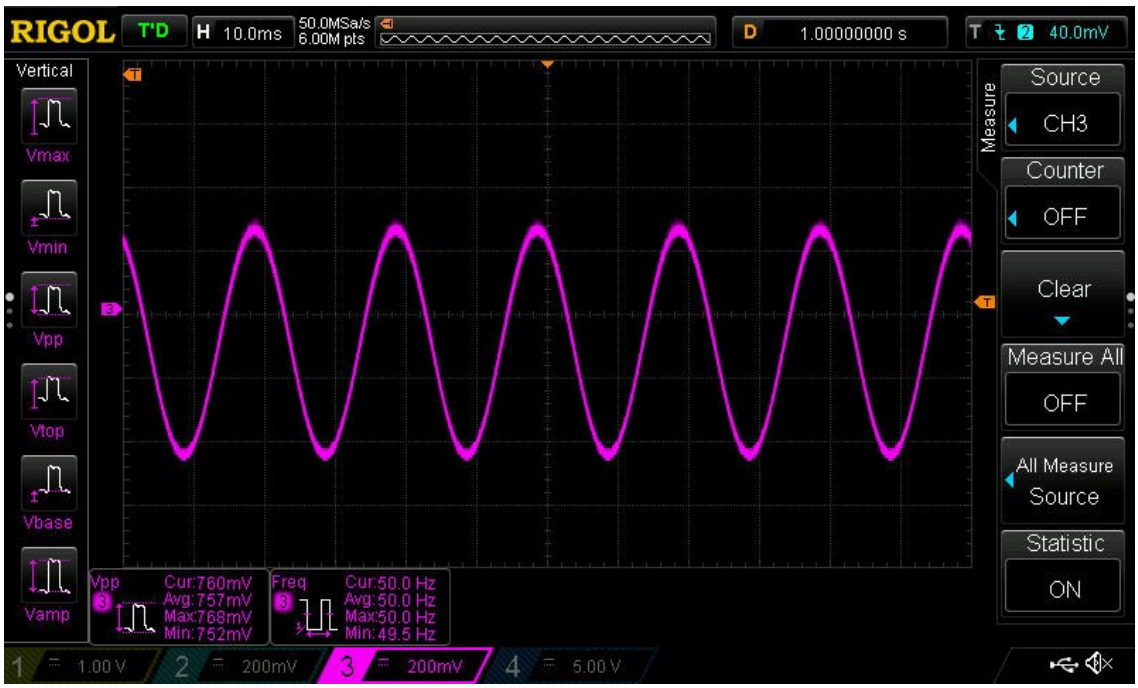


Figura 33: segnale uscita alla frequenza di taglio inferiore

Come si può notare dalle misurazioni a display l'ampiezza passa a 760mV quando raggiungo una frequenza di 50Hz; posso ritenermi soddisfatto della precisione della frequenza di taglio inferiore.

Ora sposto il potenziometro digitale al minimo consentito di 20 KΩ ed eseguo la medesima misura

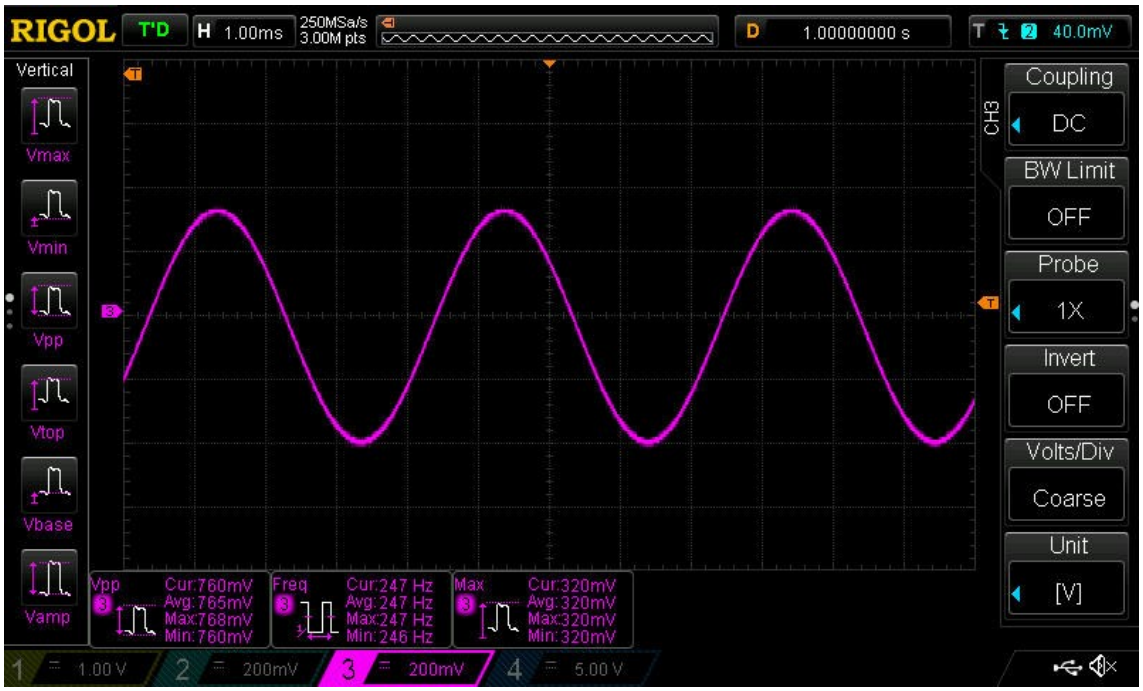


Figura 34: segnale uscita alla frequenza di taglio inferiore

Posso visualizzare come la frequenza di taglio superiore sia di 247Hz, molto vicina ai 260Hz nominali, un valore più che accettabile.

Passiamo ora all'analisi del segnale in uscita. Nella figura 35 possiamo notare in blu il segnale in ingresso e in giallo il segnale in uscita dall'amplificatore invertente. Salta subito all'occhio uno sfasamento di -180°. Il seno in ingresso ha l'ampiezza massima attesa di 2Vpp e settando il potenziometro digitale a 25 KΩ si ha un'amplificazione di 2.4.

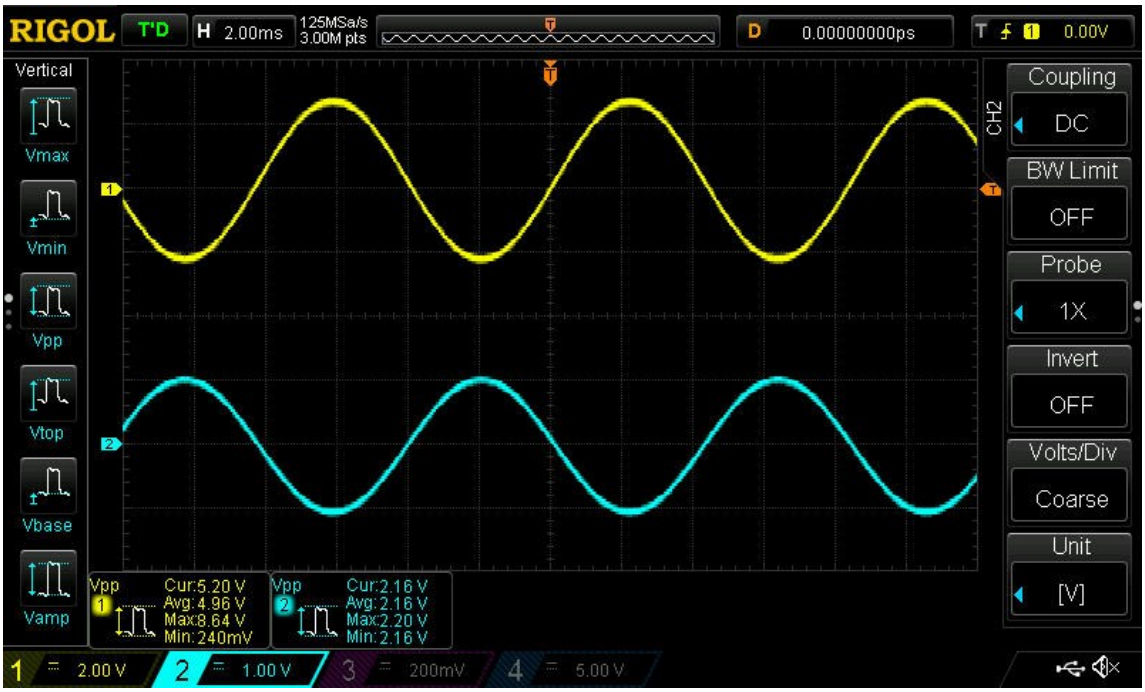


Figura 35: segnale in uscita con potenziometro a 25KΩ

Se invece vado a ridurre al minimo il potenziometro digitale e quindi abbasso il volume posso notare che il segnale in uscita si riduce drasticamente fino a scomparire quando arrivo a 30 Ω.

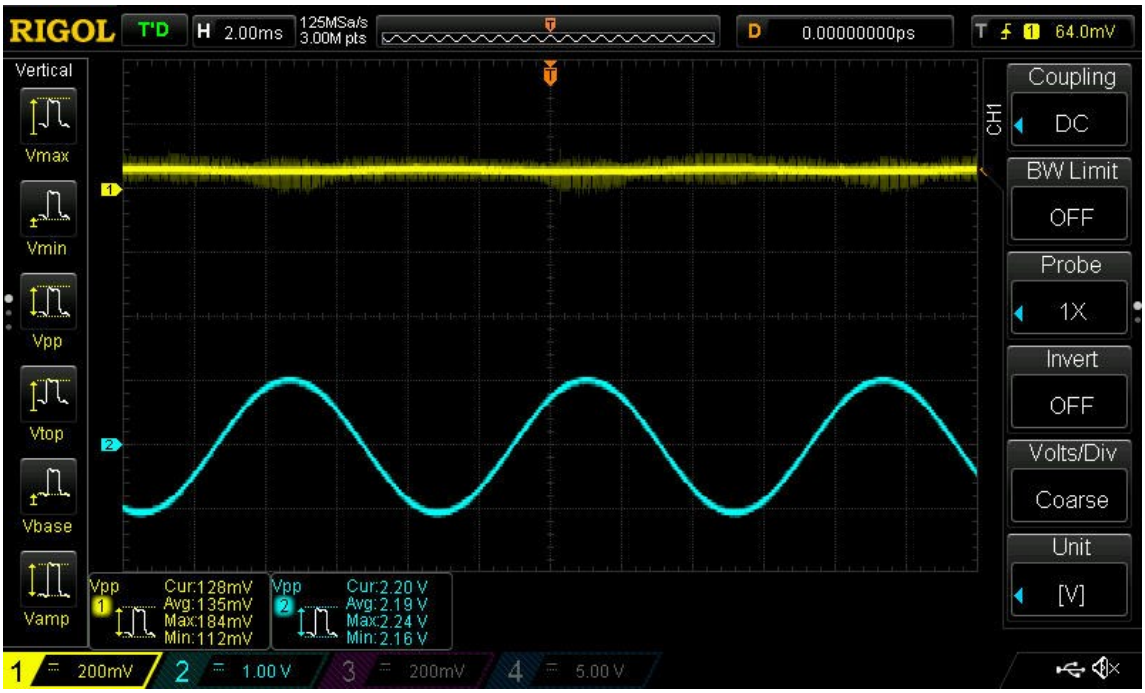


Figura 36: segnale in uscita con potenziometro al minimo

Posso quindi concludere che anche il circuito del guadagno funziona correttamente e permette all'utente di mutare il dispositivo anche con il volume massimo in ingresso. Inoltre, il segnale è privo di particolari disturbi lasciando l'onda in uscita pulita come quella in ingresso.

12. PROPOSTA ALTERNATIVA

Se si desidera avere un controllo più preciso ed affidabile con pochi sforzi è possibile modificare il programma per impostare la resistenza del potenziometro digitale usando un computer e la comunicazione seriale tramite porta micro-usb già presente all'interno del dispositivo. In questo modo è possibile comandare il dispositivo in un ambiente più rumoroso e posizionarlo in ambienti dove un segnale ad infrarossi non sarebbe in grado di raggiungerlo.

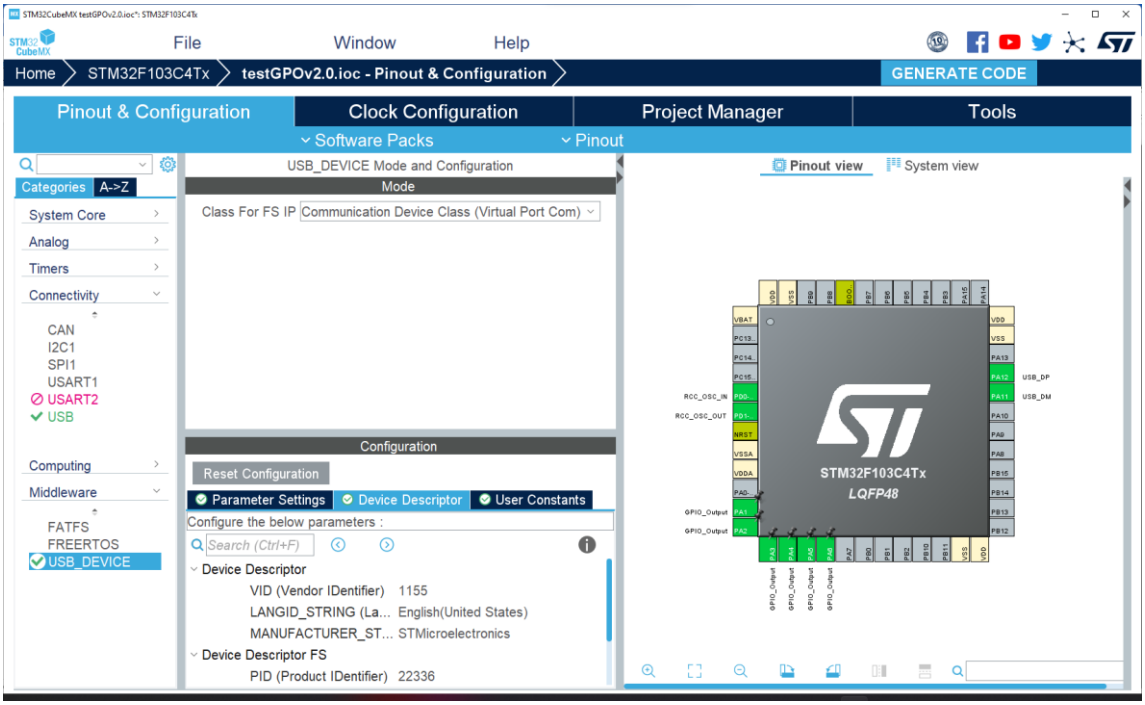


Figura 37: pagina iniziale STMCubeMX

Per la realizzazione, come prima, inizio con la configurazione delle periferiche tramite CubeMX. Prima ho abilitato la USB e poi nella sezione Middlewere si è attivata USB_DEVICE. Ora ne posso modificare le impostazioni così ho selezionato Class For FS IP e scelto Communication Device Class nel menù a tendina. Le altre impostazioni si possono lasciare invariate, servono per cambiare il nome del dispositivo una volta riconosciuto dal computer. Come ho fatto precedentemente abilito i pin di output per la comunicazione con i potenziometri digitali.

Subito dopo aver generato il codice inizializzo all'interno del main.c un array di interi senza segno a 8 bit che per comodità chiamo send. Verrà utilizzata per scambiare dati tra la periferica e il computer

```
uint8_t send[7]="p50000";
```

Per ricevere informazioni dal computer viene usata la funzione CDC_Receive_FS presente nel file usbd_cdc_if.c generato da CubeMX. Inizialmente questa funzione sarà vuota, sta al programmatore implementarla per la corretta comunicazione col dispositivo

```
static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)
{
  /* USER CODE BEGIN 6 */
  USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
  USBD_CDC_ReceivePacket(&hUsbDeviceFS);

  memset (send, '\0', 65);  // pulisce la memoria quando si riceve un dato
  uint8_t len = (uint8_t) *Len;
  memcpy(send, Buf, len);  // copia il dato nel buffer
  memset(Buf, '\0', len);   // pulisce il buffer
  return (USBD_OK);
  /* USER CODE END 6 */
}
```

Questa funzione verrà chiamata in automatico quando il microcontrollore riceve un dato dal computer, copia il valore ricevuto nella variabile send in modo da poterne successivamente leggere il valore.
Quello che mi resta da fare è adattare il dato ricevuto che sarà un array di numeri interi senza segno a 8 bit e convertirlo in intero per renderlo compatibile con la funzione DP_setvalue scritta precedentemente.
Per farlo ho scritto la funzione ToInt.

```
void ToInt(uint8_t send[],int *d,int *p){
        int i;
        int a,b=0;
        int n=0;
                for(a=6;a >=1 ;a--){
                        char c = send[a];
                        i = potenza(b);
                switch (c){
                        case '1':
                                n =( n + (i)*1);
                                b++;
```

```
break;
case '0':
        n = n * 10 ;
        b++;
break;
case'2':
        n = n + (i)*2;
        b++;
break;
case '3':
        n = n + (i)*3;
        b++;
break;
case'4':
        n = n + (i)*4;
        b++;
break;
case '5':
        n = n + (i)*5;
        b++;
break;
case '6':
        n = n + (i)*6;
        b++;
break;
case'7':
        n = n + (i)*7;
        b++;
break;
case '8':
        n = n + (i)*8;
        b++;
break;
case '9':
        n = n + (i)*9;
        b++;
break;
case 0x00:
```

```
                    break;
            }
            c = send[0];
                    if(c =='p') *p = n;
                    if(c== 'd') *d =n;
            }
}

int potenza(int a){
            int i;
            int pot=1;
            if (a == 0){ return pot=1;}
            else{
            for(i=0;i< a; i++){
            pot= pot*10;
            }
            return pot;
            }
}
```

Semplicemente legge ogni elemento di array e lo somma all'elemento precedente moltiplicandolo prima per 10^i, dove i è la posizione del numero all'interno dell'array. La posizione iniziale è riservata alla lettera p o d necessaria per comunicare al microcontrollore se modificare il valore del potenziometro del volume o del filtro.

All'interno del main ho scritto solo poche righe per richiamare le funzioni appena scritte nell'ordine corretto

```
ToInt(send,&recivedD,&recivedP);
DP_setvalue(recivedP, &DP_pot);
DR_setvalue(recivedR, &DR_pot);
```

Per la ricezione del dato non bisogna richiamare nessuna funzione, verrà eseguito tutto in background. Il programma continua a convertire il dato ricevuto e lo imposta ai potenziometri, nel caso il valore sia indifferente da quello già impostato non viene fatto nulla.
Per la comunicazione seriale tra il computer e il microcontrollore ho utilizzato un programma open source di nome Hercules.

Questo programma lo trovo immediato, ha tutte le impostazioni necessarie sottomano e visibili nella schermata principale.

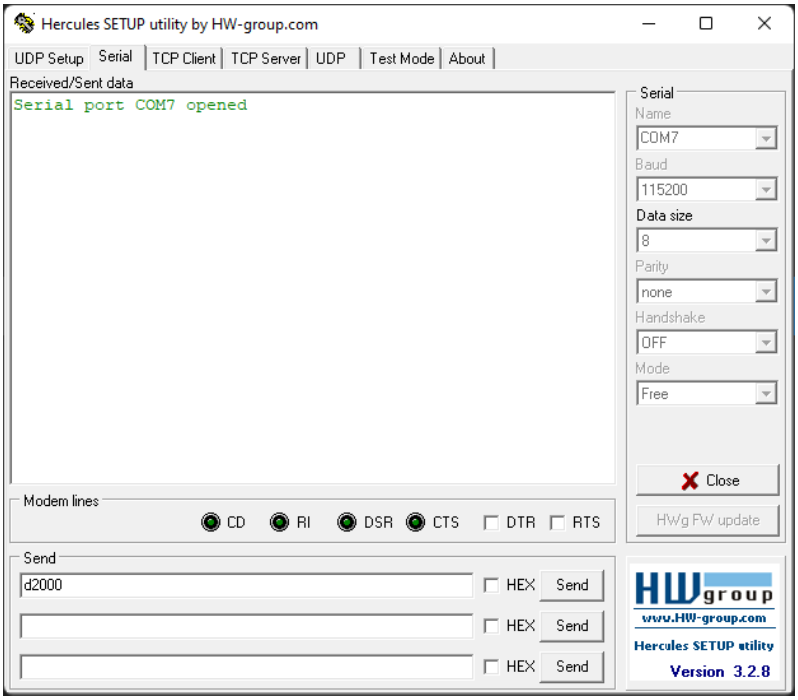


Figura 38: esempio di comunicazione tramite Hercules

Seleziono la porta seriale e imposto sulla destra tutti i parametri come su li ho trovati su CubeMX. La porta corretta (nel mio caso la COM7) può essere trovata tramite la gestione dispositivi di windows e varia in basa alla porta usb alla quale è collegato il dispositivo. Posso poi cliccare su connect e si aprirà la connessione col dispositivo; ne ho conferma grazie alla scritta verde che compare a schermo. Da questo momento posso mandare il comando dalla casella send; ce ne sono diverse ma fanno tutte la stessa funzione. Il comando inviato è composto da un primo carattere (p o d) che seleziona rispettivamente il potenziometro del volume o del filtro, seguito dal valore numerico della resistenza che poi verrà impostata. Basta premere send e nel giro di qualche istante il potenziometro si modifica per raggiungere il valore inviato. Nel caso sia inviato un valore non disponibile non succede nulla.

13. CONCLUSIONI

Vista la semplicità hardware con la quale è stato realizzato e la presenza di un microcontrollore facilmente programmabile il dispositivo può essere modificato per ogni svariato utilizzo. Può essere utilizzato in ambiente domestico per la variazione della frequenza di taglio per un home theater oppure come filtro in un mixer digitale in un evento live. Il secondo potenziometro digitale può anche essere utilizzato per comandare un filtro passa alto, avendo così a disposizione un filtro passa banda.

Certamente presenta limitazioni; parliamo comunque di un dispositivo dal costo molto contenuto e facilmente aggiornabile.

Durante la realizzazione del progetto ho avuto diversi problemi con il software µVision, presenta infatti qualche problema e rallentamento durante il debug con il simulatore, molto probabilmente dovuto alla pesantezza del programma scritto.

Eseguire il debug direttamente sulla scheda non è stato possibile con gli strumenti a mia disposizione, vista la fase iniziale di prototipazione sarebbe stato più appropriato usare una scheda nucleo con programmatore e debugger integrato come la NUCLEO-F103RB. Tuttavia, non sono riuscito a trovare la scheda in nessun negozio fisico e online hanno un prezzo molto più elevato della soluzione da me trovata.

Mi ritengo soddisfatto del risultato finale, ho trovato molto stimolante risolvere i problemi che si sono presentati durante lo svolgimento del progetto. Avere un budget limitato mi ha incoraggiato ad abbandonare la soluzione più facile per la ricerca di quella più ottimizzata.

14. BIBILIOGRAFIA

[1] Gran parte delle informazioni generiche riguardo il microcontrollore e le immagini relative ai componenti interni sono stato prese dai datasheet del produttore "DataSheetSTM32F103" scaricabile al seguente link: https://www.st.com/resource/en/datasheet/stm32f103ze.pdf

[2] Informazioni aggiuntive e dettagliate sulle periferiche sono state acquisite tramite il "ReferenceManualSTM32F10X" scaricabile al seguente link: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[3] Per avere un riassunto sulle funzioni del dispositivo in esame ho consultato il datasheet "stm32f103c8" scaricabile al seguente link: https://www.alldatasheet.com/datasheet-pdf/pdf/201590/STMICROELECTRONICS/STM32F103C8.html

[4] Specifiche tecniche e figure riguardo il potenziometro digitale sono state prese da "X9C104 Xicor datasheet" scaricabile al seguente link: https://html.alldatasheet.com/html-pdf/34248/XICOR/X9C104/125/1/X9C104.html

[5] Specifiche tecniche e figure riguardo l'amplificatore operazionale sono state prese da "TL074CN Motorola" scaricabile al seguente link: https://html.alldatasheet.com/html-pdf/5777/MOTOROLA/TL074CN/258/1/TL074CN.html

[6] Specifiche tecniche e figure riguardo il ricevitore a infrarossi sono state prese dal datasheet del produttore al seguente link: https://pdf1.alldatasheet.com/datasheet-pdf/view/26587/VISHAY/TSOP1730.html

[7] Le informazioni riguardo il protocollo ad infrarossi le ho trovate sul sito di DigiKey consultabili al seguente link: https://www.digikey.com/en/maker/blogs/2021/understanding-the-basics-of-infrared-communications

[8] Le immagini dei circuiti elettronici sono state realizzate da me utilizzando il software gratuito KiCad, le restanti illustrazioni sono screenshot del computer o dello schermo dell'oscilloscopio.