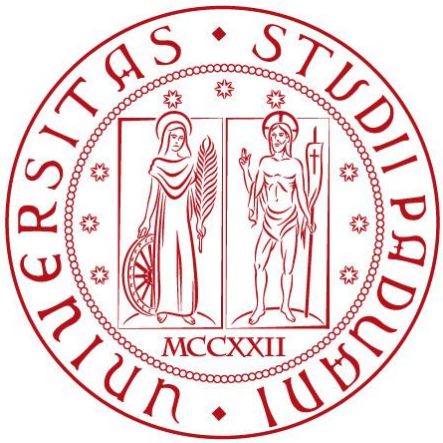

Corso di Laurea Magistrale in Ingegneria Meccatronica

Microcontrollori e DSP

Implementazione di un filtro IIR del primo ordine su scheda KL25Z

Matricola	Cognome	Nome
1197747	Braceschi	Marco
1197740	Rigon	Saverio
1205388	Zanrosso	Luca

Passo 1: inizializzazione del progetto:

Per prima cosa è necessario modificare il file startup_MKL25Z, inserendo una porzione di codice in linguaggio assembly che permette di catturare l'interrupt proveniente dall'ADC e richiamare una routine chiamata ADC0routine (che sarà implementata successivamente).

```
ADC0_IRQHandler     PROC
                    EXPORT ADC0_IRQHandler    [WEAK]
                    IMPORT ADC0routine
                    LDR R0, = ADC0routine
                    BX R0
                    EDNP
```

Nel file system_MKL25Z.h è invece necessario inserire la direttiva riportata sotto. Essa serve a fare lavorare la scheda alla massima velocità, ovvero core clock = 48MHz e bus clock =24MHz.

```
#define CLOCK_SETUP 1
```

Ora è necessario passare alla creazione del file principale del progetto, il file main. Esso dovrà contenere le seguenti routine:

- gestione dell'interrupt di reset
- gestione dell'interrupt dell'adc
- inizializzazione del convertitore analogico digitale
- inizializzazione del convertitore digitale analogico
- inizializzazione di una delle porte GPIO da usare per accendere un led

Viene sotto riportato il codice che inizializza le routine appena descritte:

```
#include "MKL25Z4.h"  //device header, contiene un richiamo dei registri di configurazione delle periferiche

void ADC_init(void){
}

void DAC_init(void){
}

void LED_init(void){
}

int main(void){
//richiama le tre routine di inizializzazione
        SystemCoreClockUpdate();//routine per aggiornare il clock
        ADC_init();
        DAC_init();
        LED_init();
```

```
        while(1){
                //loop infinito, si rimane qui in attesa di un interrupt
        }
}
```

Passo 2: inizializzazione dell'ADC

Codice per l'inizializzazione della periferica ADC

```
void ADC_init(void){  //bisogna abilitare il clock di ogni periferica, di default non viene fatto

        SIM->SCGC6|=(1UL<<SIM_SCGC6_ADC0_SHIFT); // si abilita il clock dell'adc, bit 27

        ADC0->CFG1 = (1UL<<4)|(1UL<<0)|(1UL<<3); // si abilita l'adc per eseguire una conversione a 10bit, in long time mode e con frequenza pari a busClock/2

        ADC0->SC3 = (1UL<<3); // si abilita l'adc per eseguire una conversione in continuous mode

        ADC0->SC1[0] = (1UL<<6); // si abilita il registro ADC_SC10 per acquisire in single end mode  (bit 5 a livello logico basso, come è di default) e interrupt abilitato (bit 6)

        NVIC_EnableIRQ(ADC0_IRQn); // NVIC è il vettore delle interruzione; si agisce su di esso inserendo l'indirizzo della funzione da eseguire quando la CPU riceve un interrupt.

}
```

Passo 3: inizializzazione del DAC

Codice per l'inizializzazione della periferica DAC

```
void DAC_init(void){

        SIM->SCGC6 |= (1UL<<SIM_SCGC6_DAC0_SHIFT);  // questa istruzione serve ad abilitare il clock sul DAC. Siccome questo registro è modificato anche nella configurazione dell'adc è necessario usare l'operatore di bitwise or.

        DAC0->C0 = (1UL<<5) | (1UL<<7);  //bit 7 al livello logico alto per abilitare il DAC mentre bit 5 per usare un trigger di tipo software

        DAC0->C1 = (1UL<<0); // si abilita il funzionamento in modalità buffer

        DAC0->C2 = (1UL<<0); // serve per impostare ad 1 la posizione superiore del DAC
}
```

Passo 4: inizializzazione del LED

Codice per l'inizializzazione del LED

```
void LED_init(void){

        SIM->SCGC5 |= (1UL<<12); //si abilita il clock

        PORTD->PCR[1] = (1UL<<8); // si imposta il mux in modo da selezionare come porta general purpose input-output la PTD1

        FPTD->PDDR = (1UL<<1); // si imposta la porta PTD1 in modo tale da essere un output

        FPTD->PCOR = (1UL<<1); // si azzera la porta PTD1 e accende il led

}
```

Passo 5: implementazione della routine di gestione dell'interrupt dell'adc

È importante che il nome di questa routine coincida con quello scelto nel file startup_MKL25Z
Leggo il valore acquisito dall'adc, lo uso come ingresso del dac, facendo lavorare il microcontrollore come buffer.
All'inizio del file main viene definito il registro r0; esso è usato per velocizzare l'elaborazione dei dati.

```
register unsigned int r0 __asm("r0");

int ADC0routine(void) {

        r0 = ADC0->R[0]; // si trasferisce il valore letto su ADC->R[0] la registro r0

        r0 = r0 << 2;

        if ((DAC0->C2 & DAC_C2_DACBFRP_MASK)==0) {
                DAC0->DAT[1].DATL = r0;
                r0 = r0 >> 8;
                DAC0->DAT[1].DATH = r0;
        } else {
                DAC0->DAT[0].DATL = r0;
                r0 = r0 >> 8;
                DAC0->DAT[0].DATH = r0;
        }

        DAC0->C0 |= (1UL << 4);
        return 1;
}
```

Commento al codice: queste righe di codice servono per passare il dato letto dall'adc (e salvato in r0) ai registri DATL E DATH, posizioni 0 e 1 del buffer. Siccome il DAC è in modalità buffer sono presenti due coppie di registri che mi permettono di scrivere sulla posizione 0 e 1 del buffer.
L'if serve per verificare la posizione del puntatore che deve puntare prima alla casella DAT0 del buffer e successivamente al comando di trigger (istruzione DAC ->C0 |= (1UL<<4)) al DAT1.
Il problema è che la conversione da analogico a digitale avviene su 10 bit, mentre quella da digitale a analogico su 12, per risolvere servono delle operazioni di shift.

Gli 8 bit iniziali del registro r0 vengono shiftati a sinistra di due posizioni in modo da usare i bit più significativi del convertitore. Gli 8 bit meno significativi vengono copiati nella parte bassa del registro DAT.
Per accedere ai 4 bit più significativi di r0 (quelli che avevo shiftato di due a sinistra in precedenza) devo eseguire un nuovo shift a destra di 8 posizioni e poi li copio nella parte alta del registro DAT.

Passo 6: test del microcontrollore in modalità buffer

È stata applicata mediante il generatore di funzioni una sinusoide all'ingresso dell'ADC di frequenza 10kHz, ampiezza picco-picco di 2V e offset di 1.5V. Si vuole ora confrontare le 2 sinusoidi visualizzate mediante oscilloscopio, la prima presa direttamente dal generatore di funzione mentre la seconda dal DAC del microcontrollore.

Il processo di conversione A-D a 10 bit comporta una discretizzazione, assieme a un ritardo dovuto alla conversione da A a D e da D ad A, assieme al tempo di esecuzione dell'interrupt dell'adc.

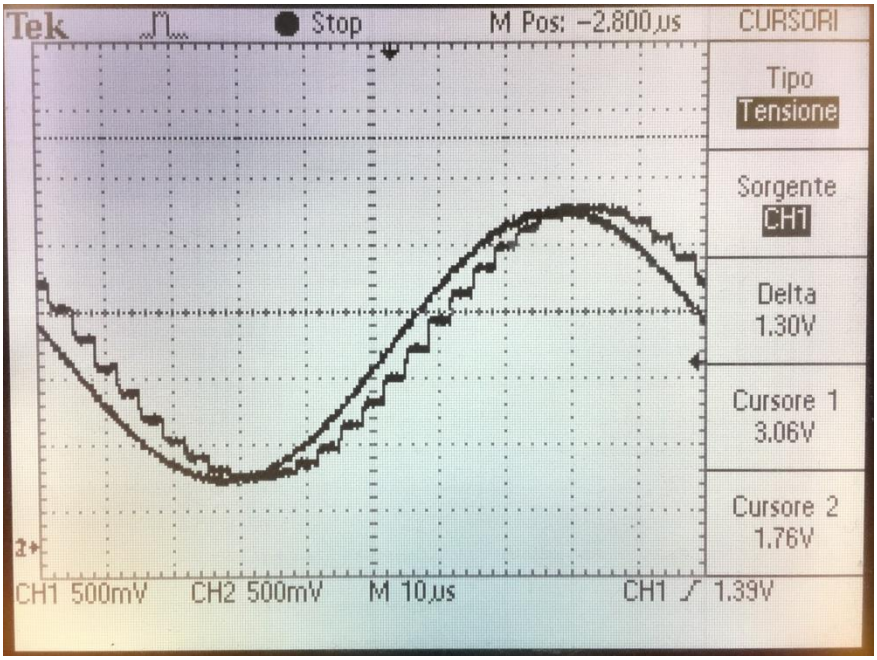

Figura 1: misura della tensione picco picco e della frequenza delle sinusoidi

In figura 1 si vede come le due sinusoidi abbiano una ampiezza picco-picco di circa 2V e una frequenza di 10kHz, come mi aspettavo.

Per quanto riguarda la stima del ritardo introdotto dal microcontrollore, l'ADC campiona il segnale con una frequenza f_c = 300kHz, quindi un periodo T_c = 3.33us. Inoltre, ipotizzando che ad ogni ciclo di clock venga svolta un'istruzione assembly, la routine richiede circa 48 istruzioni, che equivale ad un ritardo di circa 1us (ottenuto dividendo il numero di istruzioni con il valore del core clock). Si può quindi stimare approssimativamente il ritardo dovuto alla conversione digitale - analogico utilizzando il ritardo misurato sperimentalmente, ovvero 4.6us - 3.3us - 1us = 0.3us.

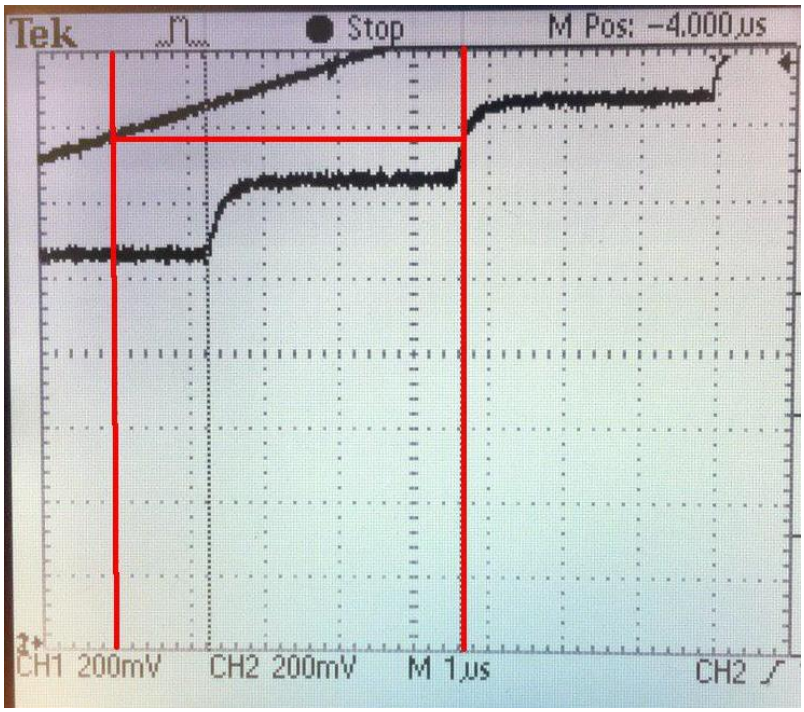

Figura 2: ritardo tra le due sinusoidi di frequenza pari a 10kHz

Passo 7: implementazione del filtro IIR

Il filtro IIR che vogliamo implementare deve avere le seguenti caratteristiche:

- filtro passa basso
- frequenza di taglio f_t = 10kHz
- attenuazione a 100kHz pari a 20dB
- guadagno unitario (ottenuto facendo si che la somma dei coefficienti a e b sia unitaria)

La frequenza di campionamento risulta essere pari a f_t =300kHz.

Il filtro IIR è nella forma sotto riportata con a = 1-b , $b = \frac{T_c}{T_c + \tau}$ e $\tau = \frac{1}{2\pi f_t}$.

y(k) = a · y(k-1) + b · x(k).

Svolgendo i conti, b=0.173 e a=0.827.

Vogliamo lavorare con numeri a 16 bit, in virgola fissa, quindi avendo i coefficienti a e b frazionari devo moltiplicarli per il valore 2^{16}, ovvero il fattore di scala.

codice del filtro IIR

```
#include "MKL25Z4.h"
```

```
register unsigned int r0 __asm("r0"); //registro in cui si salva il prodotto  b · x(k)
register unsigned int r1 __asm("r1"); //registro in cui si salva il prodotto a · y(k-1)

static unsigned int output = 0;  //variabile in cui si accumula il valore dell'uscita y


//valore dei coefficienti moltiplicati per 2^16
#define a 54187
#define b 11349

int ADC0routine(void) {
   r0 = ADC0->R[0]; //si mette il valore letto dall'adc nel registro r0 ( sarebbe x(k))
   r0 = b * r0;  // si esegue il prodotto b · x(k)
   r1 = a * output; // si esegue il prodotto a · y(k-1)
   r0 = r0>>16; //shift di 16 posizioni a destra, ovvero si annulla la precedente normalizzazione
   r1 = r1>>16; //shift di 16 posizioni a destra, ovvero si annulla la precedente normalizzazione
   r0 = r0 + r1; // si esegue la seguente somma a · y(k-1) + b · x(k)
   output = r0;  // si salva il risultato in r0
   // si trasferisce il contenuto di r0 nel DAC usando gli accorgimenti descritti al passo 5
   r0 = r0 << 2;

    //per passare il contenuto di r0 al dac si usano le stesse precauzioni prese al passo 5
if ((DAC0->C2 & DAC_C2_DACBFRP_MASK)==0) {
                    DAC0->DAT[1].DATL = r0;
                    r0 = r0 >> 8;
                    DAC0->DAT[1].DATH = r0;
    } else {
                    DAC0->DAT[0].DATL = r0;
                    r0 = r0 >> 8;
                    DAC0->DAT[0].DATH = r0;
    }

    DAC0->C0 |= (1UL << 4);
    return 1;
}
```

Passo 8: test del filtro IIR

Vengono sotto riportate alcune foto in cui si è testato il funzionamento del filtro IIR, a diverse frequenze.

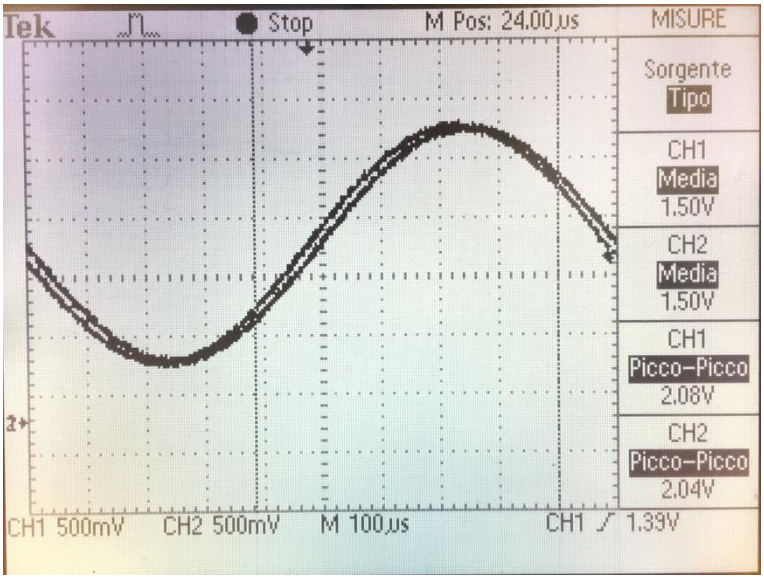

Figura 3: segnale filtrato e non filtrato a frequenza di 1Hz

A frequenza di 1Hz si nota come sia il ritardo tra i due segnali sia l'attenuazione dovuta al filtraggio sia pressoché nulla.

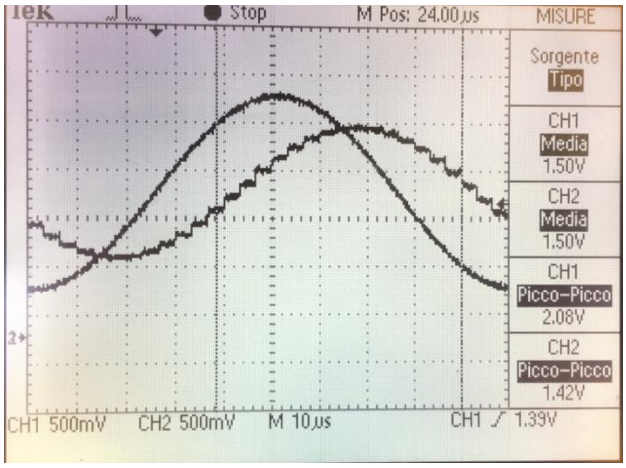

Figura 4: segnale filtrato e non filtrato alla frequenza di 10kHz

L'ampiezza del segnale proveniente dal generatore di onde è di 2 Vpp, mentre il segnale filtrato ha ampiezza di 1,42 Vpp, ovvero una attenuazione del segnale del 30% (3dB), come mi aspettavo.

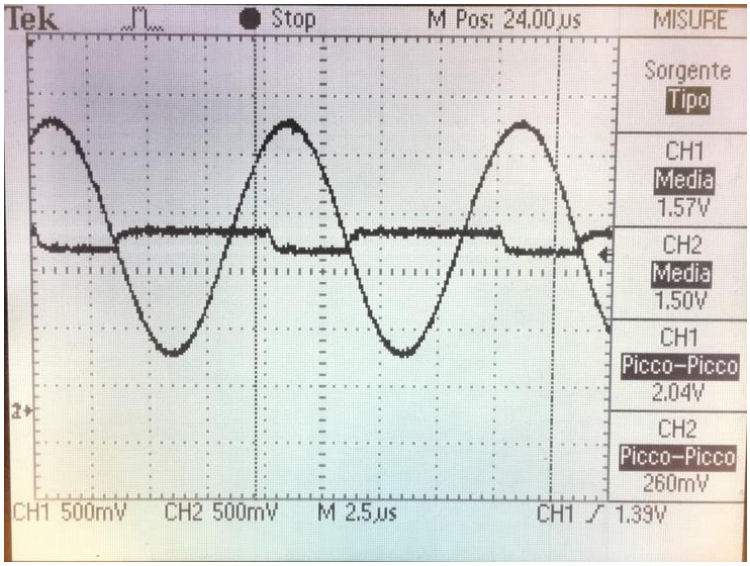

Figura 5: segnale filtrato e non filtrato alla frequenza di 100kHz

Mediante l'uso di matlab abbiamo ricavato che valore deve avere il modulo alla frequenza di 100kHz, il risultato è che deve essere pari a 0.1.
L'ampiezza del segnale proveniente dal generatore di onde è di 2Vpp, mentre il segnale filtrato ha ampiezza di 260mVpp. A 100Khz la risposta deve essere attenuata di 20dB, ovvero il valore del segnale proveniente dal generatore di funzioni deve essere moltiplicato per 0.1.
2Vpp*0.1 = 0.2Vpp, che corrisponde con un minimo di approssimazione ai 260mVpp che otteniamo mediante l'oscilloscopio.

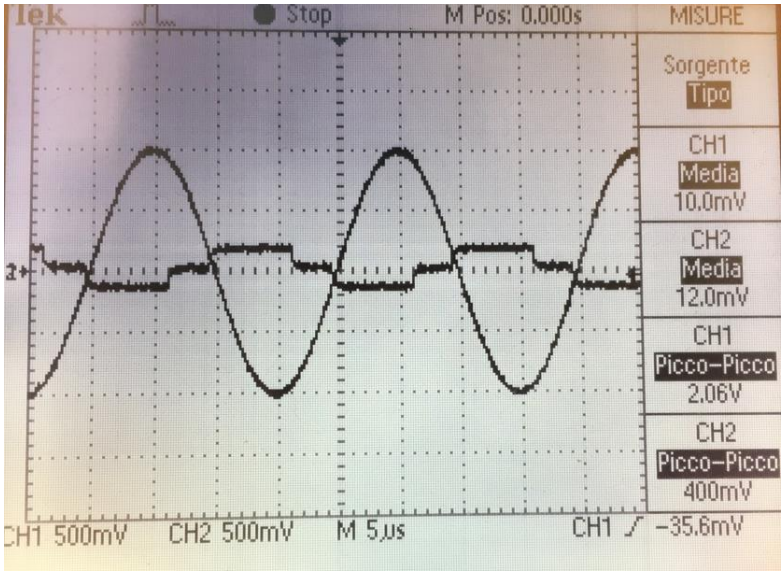

Figura 6: : segnale filtrato e non filtrato alla frequenza di 50kHz

Mediante l'uso di matlab abbiano ricavato che il valore del modulo di un filtro passa basso deve essere, a una frequenza di 50kHz pari a 0.194.
Moltiplicando il valore picco picco del segnale proveniente dal generatore di funzioni, ovvero 2V per 0.194, si ottiene 388mVpp, valore molto vicino a quello riportato nel canale 2 della figura, pari a 400mVpp.

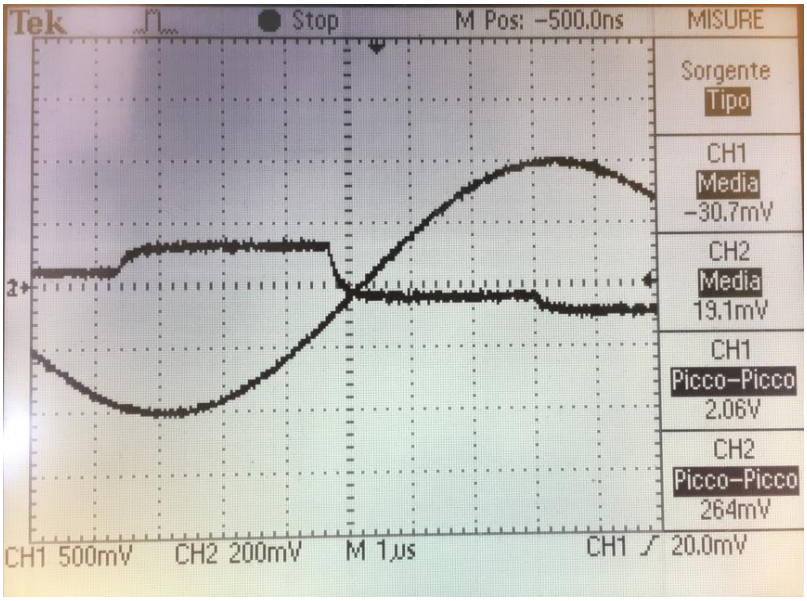

Figura 7: segnale filtrato e non filtrato alla frequenza di 80kHz

A 80kHz invece matlab suggerisce un valore di modulo pari a 0.124, 2Vpp* 0.124 = 0.248mVpp, valore simile ai 264mVpp riportati in figura.

Viene ora riportato un grafico del modulo realizzato variando la frequenza della sinusoide proveniente dal generatore di funzioni e andando a vedere il modulo e la fase del segnale filtrato mediante oscilloscopio.
Per quanto riguarda la fase il diagramma è stato tracciato misurando con l'oscilloscopio il ritardo tra segnale non filtrato e filtrato, togliendo a questo valore i 4,6 us di ritardo introdotto dalla scheda e trasformando il risultato in una misura di fase.
I due grafici riportano in blu i valori del segnale reale, mentre in arancio ci sono i valori reali ottenuto con matlab.

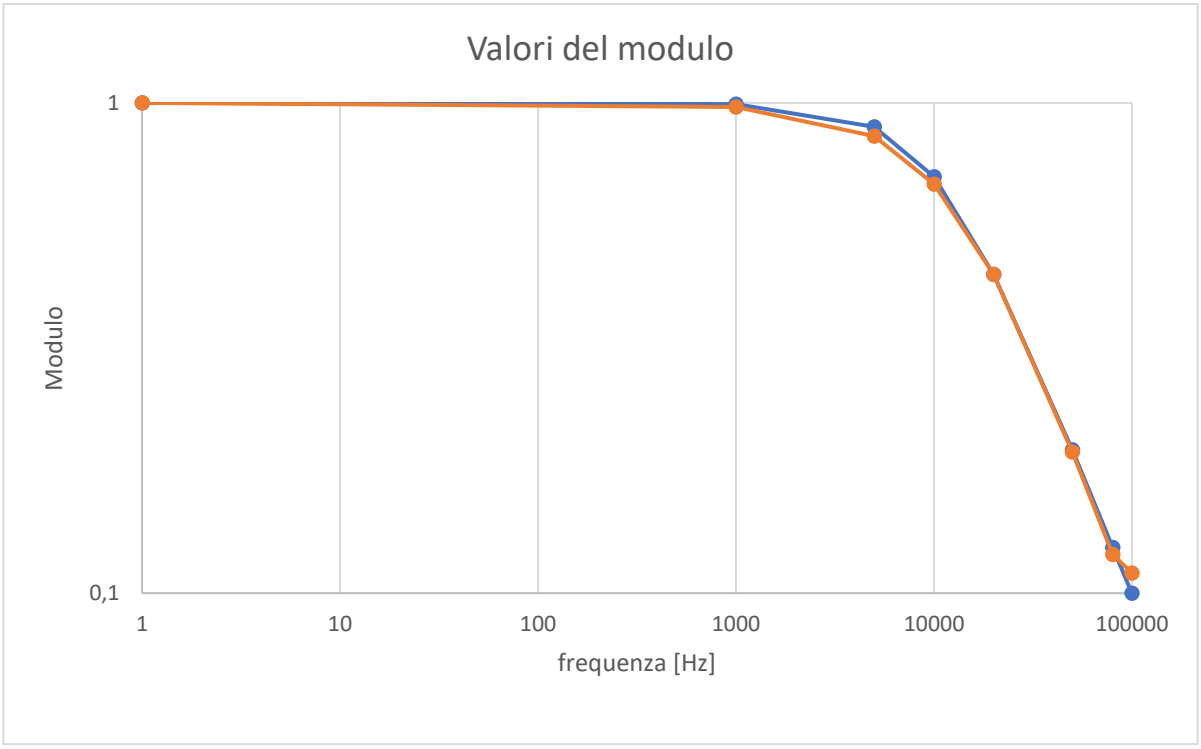

Figura 8: diagramma di bode sperimentale del modulo

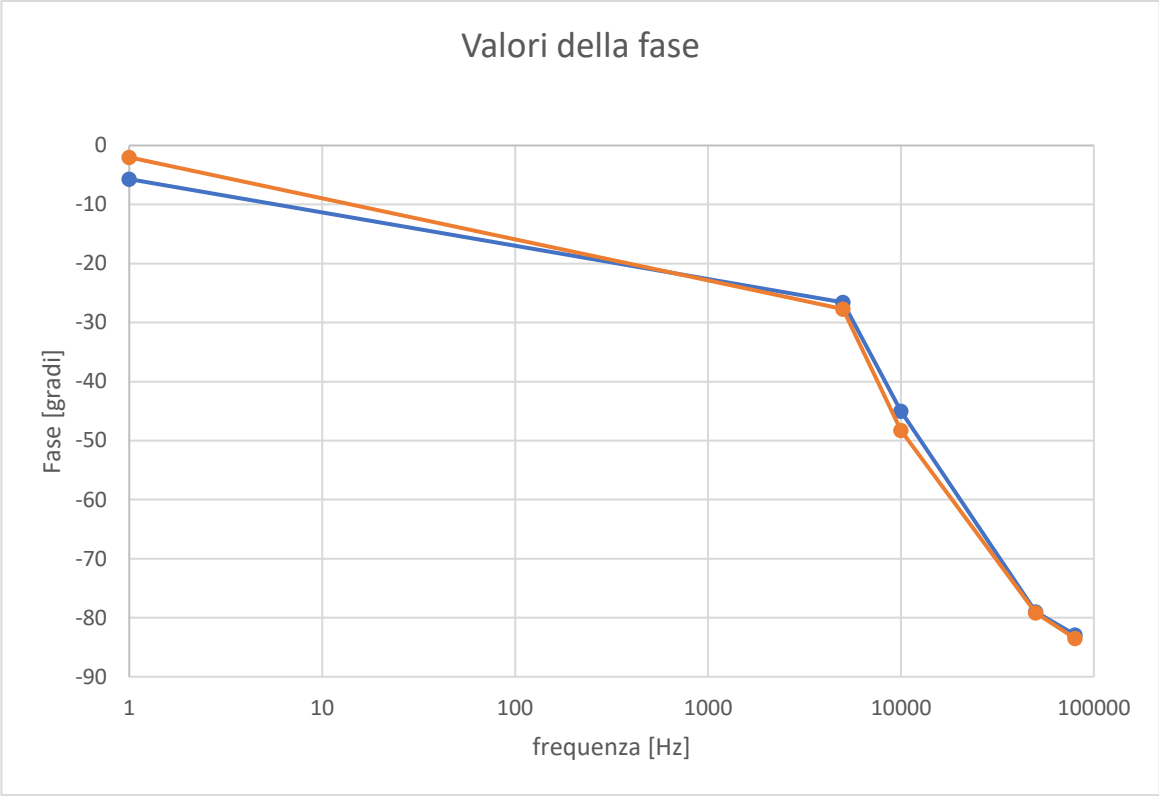

Figura 9: diagramma di bode sperimentale della fase

Viene di seguito riportata una tabella con contenuti i dati usati per tracciare i grafici

	Modulo		Fase			
Frequenza	matlab	sperimentale	matlab	sperimentale	ritardo matlab (in us)	ritardo (in us)
1000	0,995	0,981	-5,7	non misurabile	15,9	non misurabile
5000	0,894	0,856	-26,6	-27,72	14,6	20
10000	0,707	0,683	-45	-48,24	12,5	18
50000	0,196	0,194	-79	-79,2	4,3	9
80000	0,124	0,12	-82,9	-83,52	2,9	7,5
100000	0,1	0,11	-84,2		2,3	

Si può in particolare studiare il ritardo per una sinusoide a 10kHz.

Il ritardo intrinseco del filtro è pari a 12.5us, calcolato con la seguente formula $^{\pi/4}/_{f_t}$.

Il ritardo introdotto dal microcontrollore, filtro escluso, è di circa 4.6us, come visto nel precedente passo. Il ritardo, complessivo quindi è 17.1us.

Si confronta il ritardo appena calcolato con quello misurato sperimentalmente, mediante oscilloscopio riportato in figura, che è di 17 us. Il risultato atteso è molto vicino al valore misurato.

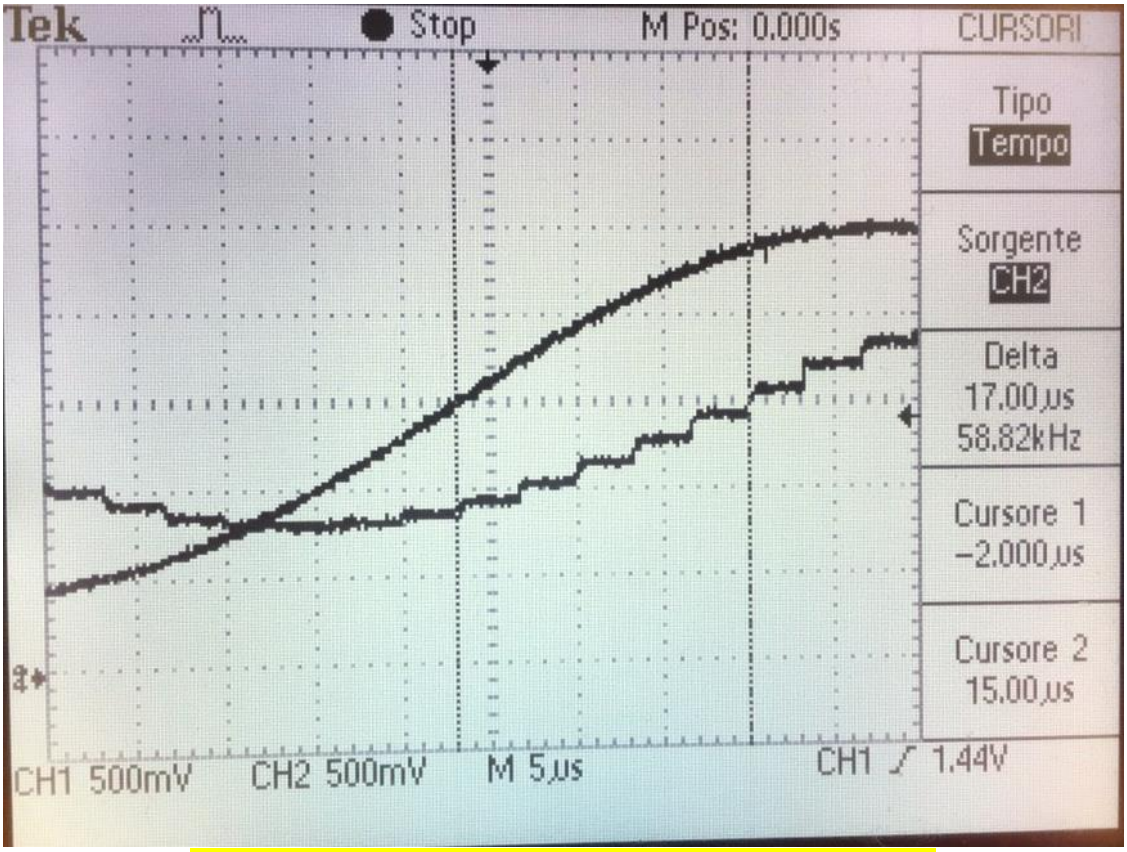

Figura 10: misura del ritardo introdotto dal filtraggio alla frequenza di 10kHz

Attività facoltative

Implementazione di un filtro FIR a media mobile

Specifiche di progetto

- frequenza di taglio f_t = 10kHz
- attenuazione a 100kHz di 20dB

Il filtro a media mobile di tipo fir è stato implementato mediante la seguente formula:

$$y(k) = \frac{1}{N}\sum_{i=0}^{N-1} x(k-i)$$

Mediante la funzione freqz di matlab si sono trovati i valori dei coefficienti b ($b = \frac{1}{N}$) che garantiscono le specifiche richieste ovvero 0.0769, infatti come si vede in figura si ha che alla pulsazione di $2\pi f_t$ il fir ha una attenuazione di 3dB.

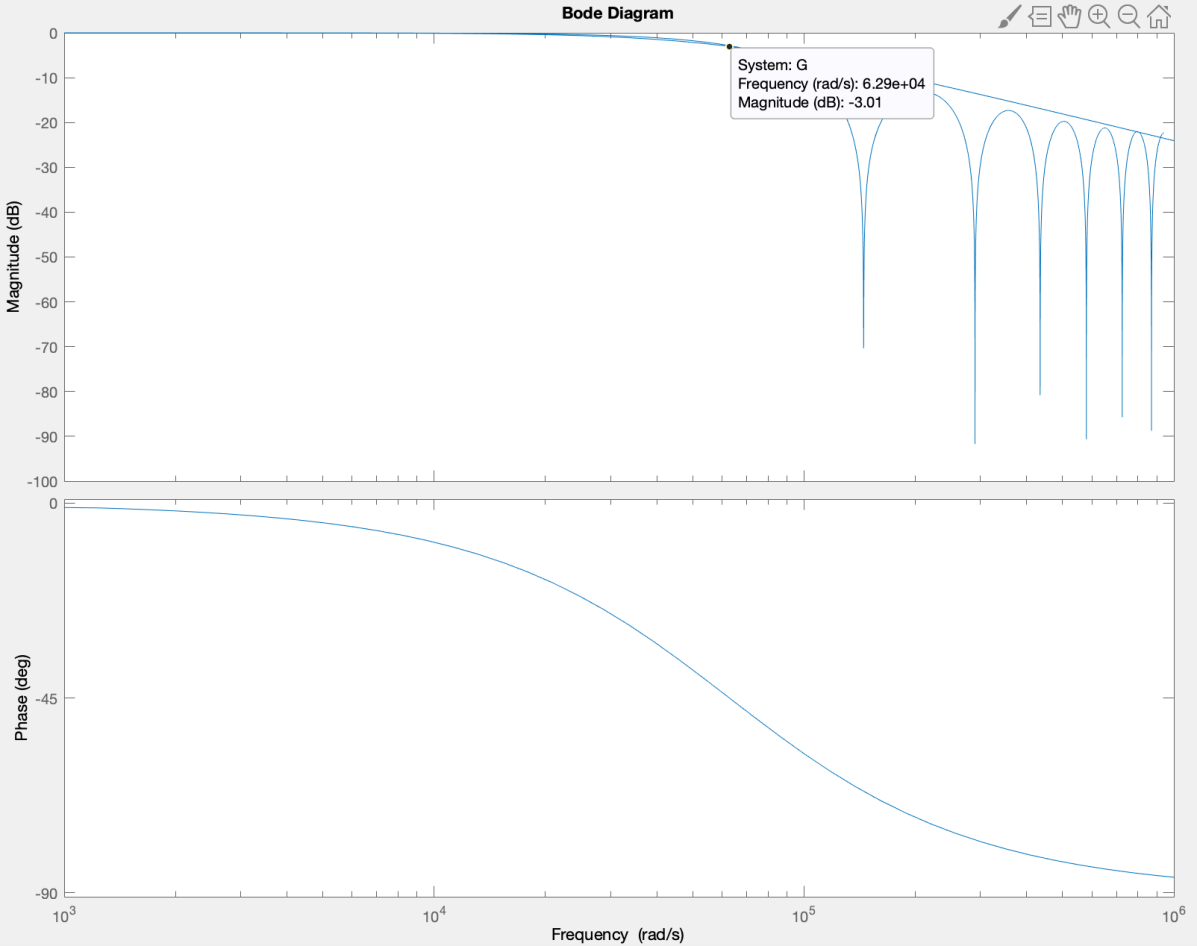

Figura 11:diagramma di bode di un filtro bassa passo e di un filtro passa basso di tipo fir

Codice per l'implementazione del fir, mediante la formula sopra riportata

```
        #define c 5041  // 0.0769 * 2^16
        Int values[13];
        int i = 0;

int ADC0routine(void) {
        r0 =  ADC0->R[0];
        r0 = c* r0;
        r0 = r0>>16;
        output -= values[i];
        output += r0;
        values[i] = r0;
        i = (i + 1) % 13;
        r0 = output << 2;
        if ((DAC0->C2 & DAC_C2_DACBFRP_MASK)==0) {
                DAC0->DAT[1].DATL = r0;
                r0 = r0 >> 8;
                DAC0->DAT[1].DATH = r0;
        } else {
```

```
            DAC0->DAT[0].DATL = r0;
            r0 = r0 >> 8;
            DAC0->DAT[0].DATH = r0;
        }

        DAC0->C0 |= (1UL << 4);
        return 1;
}
```

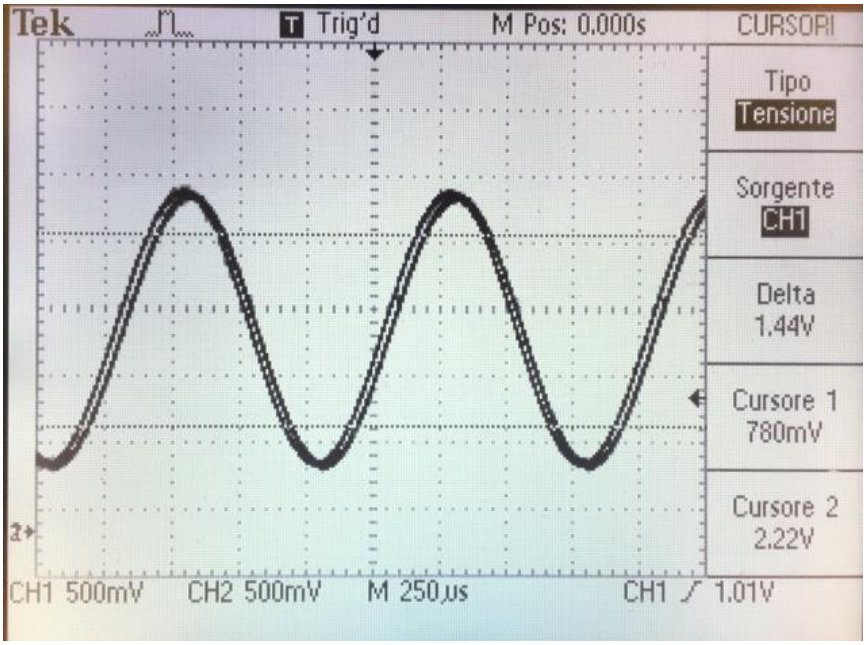

Figura 12: segnale filtrato (filtro fir) e non filtrato alla frequenza di 1Hz

Misura a 1kHz, dove si vede che il segnale filtrato non è attenuato ma solamente ritardato.

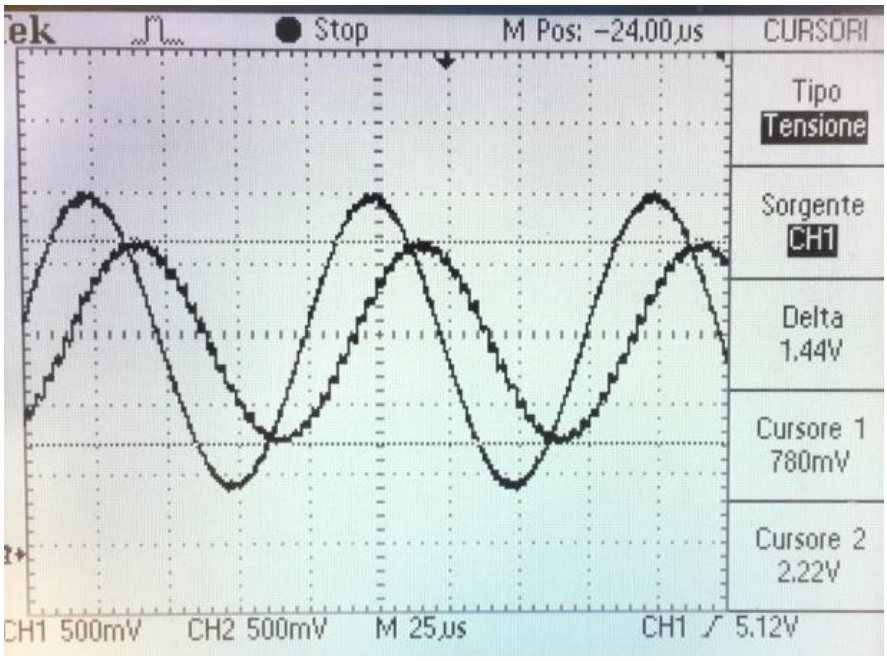

Figura 13: segnale filtrato (filtro fir) e non filtrato alla frequenza di 10kHz

L’ampiezza del segnale, proveniente dal generatore di onde è di 2 Vpp, mentre il segnale filtrato ha ampiezza di 1,44 Vpp, ovvero una attenuazione del segnale del 30% (3dB), come ci si poteva aspettare.

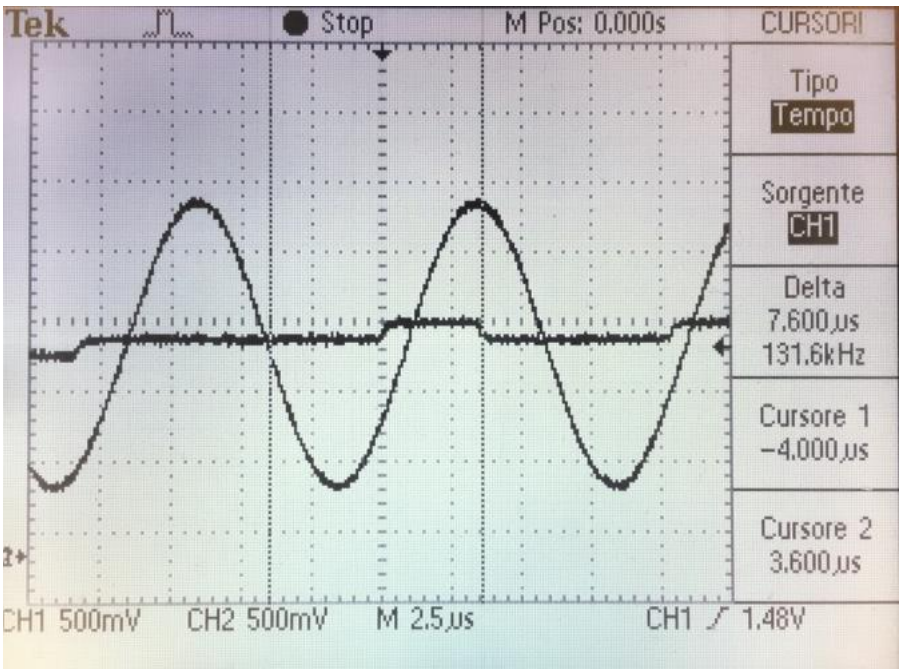

Figura 14: segnale filtrato (filtro fir) e non filtrato alla frequenza di 100kHz

A 100Khz la risposta deve essere attenuata di 20dB, cioè di 0.1, cosa che risulta essere in buona approssimazione verificata dai valori dell’oscilloscopio.

Misura del tempo di ritardo tra i due segnali.

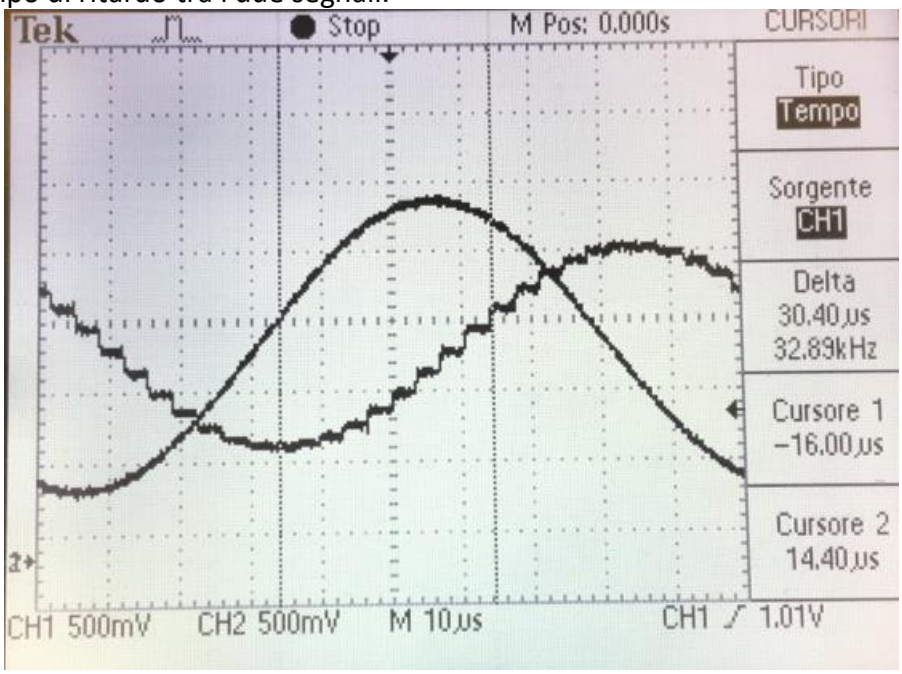

Figura 15: misura del ritardo tra segnale di partenza e segnale filtrato

Come si vede dai cursori della figura il ritardo tra i due segnali risulta essere pari a 30.40 µs.

Il ritardo introdotto dal FIR è maggiore rispetto al filtro IIR. Infatti, sono necessari più campioni per ottenere un risultato simile al filtro IIR.

Utilizzo del DAC senza buffer

Si modifica la routine dell'ADC usando solo il bit 0 del registro DAT, lasciando cioè fisso il puntatore su di esso.

```
int ADC0routine(void) {


        r0 = c * ADC0->R[0];
        r0 = r0>>16;
        output -= values[i];
        output += r0;
        values[i] = r0;
        i = (i + 1) % 13;
        r0 = output << 2;

//uso solo DAT[0], quindi elimino l'if e l'incremento del puntatore
        DAC0->DAT[0].DATL = r0;
        r0 = r0 >> 8;
        DAC0->DAT[0].DATH = r0;
        DAC0->C0 |= (1UL << 4);
        return 1;
}
```

Il risultato della simulazione con questa modifica del codice è il seguente:

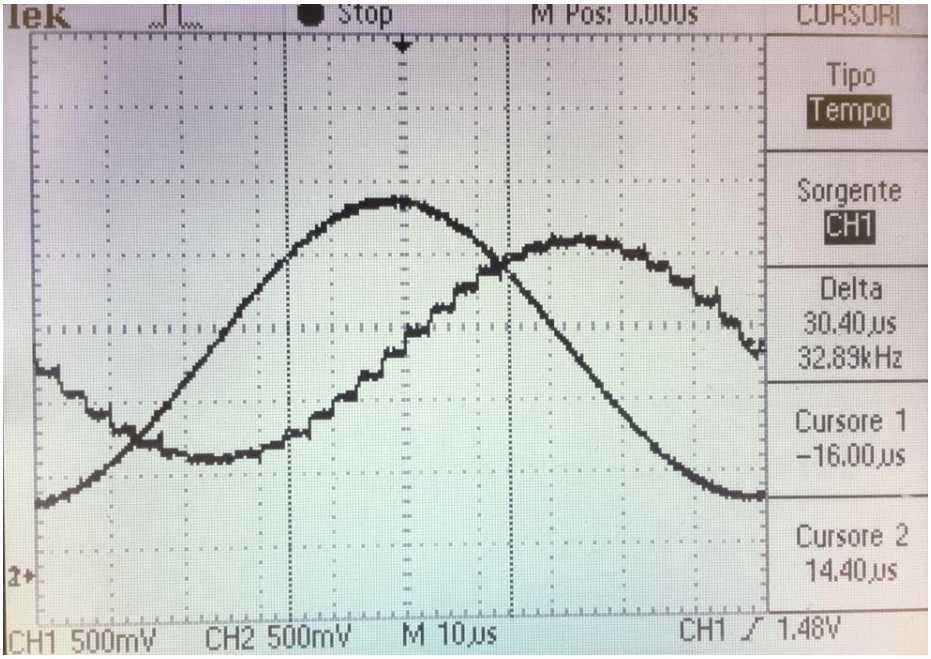

Figura 16: filtro FIR a frequenza di 10Khz, senza l'uso del buffer.

Si possono notare delle lievi sbavature in prossimità dei gradini: esse sono chiamate spikes e succedono perché la parte bassa cambia mentre quella alta rimane ancora relativa al dato precedente per un brevissimo istante (vedi codice sopra, DATL e DATH). Quindi, anche se per un tempo molto breve, si ottiene un dato sbagliato.

Viene riportato per completezza il codice completo di tutta l'esercitazione:

```
#include "MKL25Z4.h"
register unsigned int r0 __asm("r0");
register unsigned int r1 __asm("r1");

static unsigned int output = 0;
int values[13];
int i = 0;

#define a 54187
#define b 11349
#define c 5041

void ADC_init(void){
	SIM->SCGC6|=(1UL<<SIM_SCGC6_ADC0_SHIFT
	ADC0->CFG1 = (1UL<<4)|(1UL<<0)|(1UL<<3);
	ADC0->SC3 = (1UL<<3);
	ADC0->SC1[0] = (1UL<<6);
	NVIC_EnableIRQ(ADC0_IRQn);
}

void DAC_init(void){

	SIM->SCGC6 |= (1UL<<SIM_SCGC6_DAC0_SHIFT);
	DAC0->C0 = (1UL<<5) | (1UL<<7);
	DAC0->C1 = (1UL<<0);
	DAC0->C2 = 0;

}

void LED_init(void){

	SIM->SCGC5 |= (1UL<<12);
	PORTD->PCR[1] = (1UL<<8);
	FPTD->PDDR = (1UL<<1);
	FPTD->PCOR = (1UL<<1);

}

int main(void){
/
```

```
	SystemCoreClockUpdate();
	ADC_init();
	DAC_init();
	LED_init();

	while(1){
		//loop infinito
	}

}

int ADC0routine(void) {

	//IIR
	r0 = ADC0->R[0];
	r0 = b * r0;
	r1 = a * output;
	r0 = r0>>16;
	r1 = r1>>16;
	r0 = r0 + r1;
	output = r0;
	r0 = r0 << 2;

	// FIR
	r0 = c * ADC0->R[0];
	r0 = r0>>16;
	output -= values[i];
	output += r0;
	values[i] = r0;
	i = (i + 1) % 13;
	r0 = output << 2;


	if ((DAC0->C2 & DAC_C2_DACBFRP_MASK)==0) {
		DAC0->DAT[1].DATL = r0;
		r0 = r0 >> 8;
		DAC0->DAT[1].DATH = r0;
	} else {
		DAC0->DAT[0].DATL = r0;
		r0 = r0 >> 8;
		DAC0->DAT[0].DATH = r0;
	}

	DAC0->C0 |= (1UL << 4);
	return 1;
}
```