POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea in Ingegneria Elettronica

PROGETTO DEL PRIMO EMULATORE PER DISPOSITIVI MEMS BASATO SU FPGA

Relatore: Prof. Giacomo LANGFELDER
Correlatore: Ing. Paolo MINOTTI
Ing. Nicola ARESI

Tesi di Laurea Magistrale di:
Leonardo GAFFURI PAGANI
Matr. 833813

Anno Accademico 2016 - 2017

alla mia Famiglia

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

In questa tesi viene presentato un sistema digitale riconfigurabile che emula un sensore inerziale MEMS, che d'ora in poi chiameremo MEMU (Mems EMUlator).

Il suo scopo principale è quello di sintetizzare le caratteristiche elettromeccaniche tipiche del dispositivo, consentendo così la caratterizzazione di un qualunque sistema elettronico (un ASIC per MEMS o strumentazione dedicata ai MEMS) da accoppiare al dispositivo MEMS emulato, (i) senza la necessità di avere il sensore fisicamente disponibile e (ii) con la possibilità di variare i parametri del sensore come desiderato, generando così una moltitudine di possibili scenari tipicamente riscontrabili in condizioni operative con un MEMS.

Il cuore del sistema è costituito da una FPGA che emula nel dominio digitale il comportamento del dispositivo. Grazie alla sua natura digitale, e perciò alla facilità di riprogrammazione, esso consente un alto grado di versatilità ed adattabilità a casi differenti di emulazione con un semplice update del firmware.

Il lavoro, in prospettiva, può risultare utile (i) sia per coloro che sviluppano e caratterizzano circuiti integrati per MEMS, (ii) sia per coloro che sviluppano schede a discreti o altra strumentazione general-purpose per la caratterizzazione di MEMS. In entrambi i casi, un MEMU consente loro una fase di debug e analisi dell'elettronica approfondita, con la possibilità di avere un MEMS (emulato) i cui parametri sono noti e variabili arbitrariamente.

Abstract

This thesis presents a fully-configurable digital system that emulates MEMS inertial sensors, from now on called MEMU (Mems EMUlator).

Its main function is to synthesize the typical electro-mechanical features of the device, in order to allow the characterization of electronic systems (ASICs for MEMS or MEMS-dedicated instrumentation) coupled to the (emulated) MEMS, (i) without the need to have a physically available sensor and (ii) with the possibility to vary the sensors parameters as desired, thus creating a wide range of scenarios that could actually be met during the operation of a MEMS device.

The core of the system is constituted by an FPGA that emulates in the digital domain the behavior of the device. Thanks to its digital, hence easy-to-configure nature, it guarantees high versatility and adaptability to different emulation scenarios with a simple firmware update.

This project could, therefore, be useful to (i) anybody designing or charaterizing integrated circuits for MEMS, (ii) or developing a discrete board or general-purpose instrumentation for MEMS characterization. In both cases, a MEMU would give the opportunity to have an additional analysis and debug phase for the electronic system in which the (emulated) MEMS has known and fixed parameters.

Capitolo 1

Introduzione

Il lavoro di Tesi presentato è il risultato della attività svolta presso il Dipartimento di Elettronica, Informazione e Bioingegneria del Politecnico di Milano riguardante l'ambito dei microsistemi elettromeccanici (MEMS).
I risultati parziali sono stati presentati in occasione della conferenza "2017 IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)" nel marzo 2017 presso Kauai (Hawaii, USA) [1].

1.1 Motivazioni per lo sviluppo di un emulatore MEMS

Un sistema MEMS completo è solitamente formato dal microsensore meccanico e dal circuito integrato (ASIC), interconnessi ed incapsulati adeguatamente (figura 1.1).

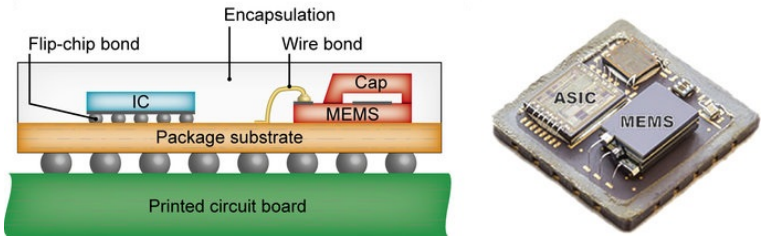


Figura 1.1: Schema e foto di un MEMS ed un ASIC accoppiati nello stesso package.

Esempi di sistemi MEMS sono sensori come accelerometri e giroscopi o attuatori come, per esempio, testine per la stampa.
Esistono diversi approcci per la caratterizzazione delle performance del sensore meccanico. In particolare, in tempi recenti, diverse compagnie ed istituti di ricerca, motivati da una crescita della richiesta del mercato, hanno realizzato tecniche per la caratterizzazione elettromeccanica [2, 3] o in-operation [2, 4, 5] del sensore, prima che questo venga accoppiato al suo ASIC (figura 1.2).

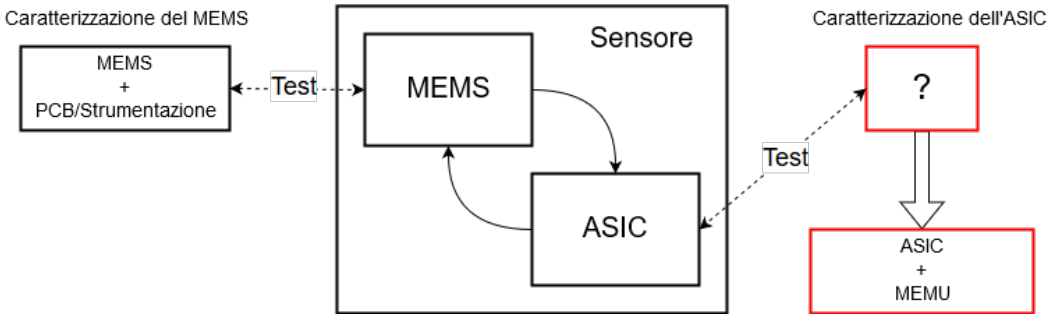


Figura 1.2: Esempio di utilizzo di MEMU per la caratterizzazione di un ASIC di un sensore MEMS.

La maggior parte di queste soluzioni sono basate su circuiti a discreti; questo permette un'alta versatilità, fornendo diversi parametri selezionabili in modo da testare il MEMS in diversi regimi di lavoro. All'atto pratico questi set-up emulano il circuito integrato e consentono di analizzare, prevedere o validare, su un particolare sensore MEMS, gli effetti di variazioni circuitali dell'elettronica a lui associata. Questo rende soluzioni di questo tipo preziose nella fase di sviluppo di un sistema.

Con un approccio duale questo lavoro propone un emulatore digitale di sensori MEMS (MEMU) a parametri variabili (figura 1.2). Questo tipo di soluzione può avere svariate applicazioni, ad esempio:

- caratterizzare e validare le prestazioni di un ASIC prima di accoppiarlo al sensore corrispondente (esempio in figura 1.3);
- accelerare la fase di debug nei casi in cui il circuito integrato sia disponibile prima del MEMS (cosa comune dovuta a ritardi produttivi causati per esempio da manutenzioni dei macchinari);
- verificare la compatibilità di un ASIC esistente con un sensore MEMS della generazione successiva (non ancora in possesso ma di cui si possono prevedere o stimare i parametri);
- predire il comportamento di un sistema nel caso in cui un parametro del MEMS drifti nel tempo o semplicemente differisca dal valore nominale (variazioni di fattore di qualità Q, etching, offset meccanico, etc. . .);
- aiutare gli sviluppatori di strumentazione dedicata ai MEMS, che solitamente non hanno accesso a tutti i tipi di dispositivi, dando la possibilità di emularne diversi in modo da poter effettuare una migliore calibrazione/debug con molti scenari differenti;
- calibrare la strumentazione in casi in cui è difficoltoso produrre un device con parametri esatti, come nel caso di Q estremamente elevati [6, 7].

In tutte queste situazioni MEMU può emulare un MEMS con parametri perfettamente noti, tramite una semplice programmazione del firmware.

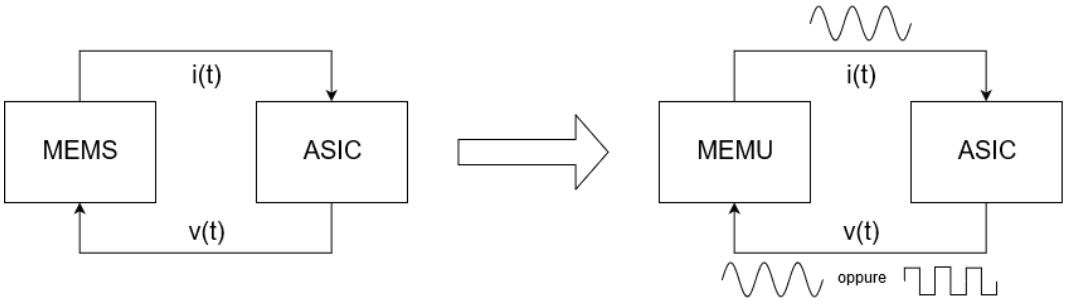


Figura 1.3: Schema della fase di test di un ASIC tramite MEMU, dove viene evidenziata la sostituzione del MEMS fisico con la sua emulazione (MEMU).

In letteratura sono presenti soluzioni simili, ma applicate ad altri campi, come ad esempio fonti [8] o rivelatori [9] di radiazioni. Queste strutture sfruttano il parallelismo dell'architettura digitale e la sua riconfigurabilità per trarne vantaggi in termini di versatilità ed efficienza dell'implementazione hardware. Non sono note all'autore applicazioni di emulatori nel campo dei MEMS.

1.2 Modellizzazione di un MEMS

Al fine di delineare le specifiche di progetto di MEMU, è comodo rappresentare il comportamento meccanico del MEMS con il suo circuito elettrico equivalente. La rappresentazione è quella di un generico MEMS a tre porte [10], che potrebbe ad esempio rappresentare un risonatore, un accelerometro con sense differenziale (in cui durante la fase di test uno dei due sense viene usato come drive) o un giroscopio.
Un MEMS di questo tipo è generalmente modellizzabile con una massa di prova m che è soggetta a:

a) la forza elastica di una molla di costante elastica k (generata dalle molle di sospensione della massa di prova);

b) la forza di attrito generata da uno smorzatore con coefficiente b (tipicamente l'attrito con il gas circostante il MEMS);

c) la forzante esterna F (che può essere presente in condizioni operative, oppure solo in fase di test).

Definita la posizione della massa rispetto a quella di riposo con x, siamo in grado di scrivere l'equazione della dinamica:

$$m\ddot{x} + b\dot{x} + kx = F \tag{1.1}$$

Portandoci nel dominio di Laplace possiamo agilmente evidenziare il trasferimento da forza $F(s)$ a spostamento $X(s)$.

$$\frac{X(s)}{F(s)} = \frac{1}{m}\frac{1}{s^2 + \frac{b}{m}s + \frac{k}{m}} \tag{1.2}$$

Considerato un classico [11] sistema di pilotaggio (drive) in tensione e lettura (sense) in corrente, possiamo riportare il trasferimento (1.2) a queste variabili ottenendo così l'ammettenza elettrica in funzione della frequenza:

$$Y(s) = \frac{i_m(s)}{V_a(s)} = \eta^2 s\frac{1/m}{s^2 + \frac{b}{m}s + \frac{k}{m}} = \eta^2 s\frac{1/m}{s^2 + s\frac{\omega_0}{Q} + \omega_0^2} \tag{1.3}$$

I parametri caratteristici della risposta in frequenza di un MEMS [12], portati in evidenza nell'equazione (1.3), sono la frequenza di risonanza ω_0 e il fattore di qualità Q:

$$\omega_0 = \sqrt{\frac{k}{m}} \qquad\qquad Q = \omega_0\frac{m}{b} = \frac{k}{\omega_0 b} \tag{1.4}$$

Da questo trasferimento possiamo notare tre zone notevoli che sono prima, dopo, ed esattamente a risonanza; in ciascuna di esse l'ammettenza è facilmente riconducibile a dei componenti passivi equivalenti:

$$\begin{aligned}
\omega \ll \omega_0 \quad &\Rightarrow \quad \frac{i_m(s)}{V_a(s)} = \frac{\eta^2}{k}s = \frac{1}{Z} = sC_{eq} \\
\omega = \omega_0 \quad &\Rightarrow \quad \frac{i_m(s)}{V_a(s)} = \frac{\eta^2}{b} = \frac{1}{Z} = \frac{1}{R_{eq}} \\
\omega \gg \omega_0 \quad &\Rightarrow \quad \frac{i_m(s)}{V_a(s)} = \frac{\eta^2}{m}\frac{1}{s} = \frac{1}{Z} = \frac{1}{sL_{eq}}
\end{aligned} \tag{1.5}$$

ottenendo così:

$$C_{eq} = \frac{\eta^2}{k}, R_{eq} = \frac{b}{\eta^2}, L_{eq} = \frac{m}{\eta^2} \tag{1.6}$$

Abbiamo perciò evidenziato come il risonatore possa essere modellizzato con un circuito $R_{eq}L_{eq}C_{eq}$ serie equivalente.[1]

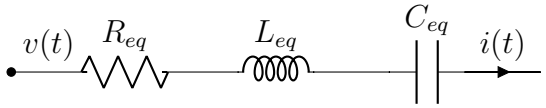


Figura 1.4: *RLC* equivalente.

Questo circuito elettrico equivalente sarà la base da cui verrà sviluppato MEMU: l'obiettivo sarà generare un sistema il cui comportamento sia descritto da queste equazioni, in modo da emularne correttamente il comportamento.

Il lettore potrebbe chiedersi quale sia la necessità di un emulatore digitale quando sarebbe quindi possibile costruire un circuito *RLC* con componenti discreti a valori variabili.
Per rispondere a questo dubbio proviamo a considerare qual è il target di questo progetto: desideriamo emulare vari tipi di dispositivi con diverse caratteristiche. In tabella 1.1 è presente un esempio dei range di valori che avremo come target da considerare per il nostro emulatore; ad esempio vediamo come i valori dei fattori di qualità Q che si vogliono emulare varino su 6 ordini di grandezza: questo perché desideriamo massimizzare il numero e tipo di dispositivi implementabili.

Variabile	**Target min**	**Target max**
Q	0.1	100k
f_0 [Hz]	500	50k
$Vdrive$ [V]	± 1	± 10

Tabella 1.1: Valori massimi e minimi di alcuni parametri caratteristici dei dispositivi MEMS da emulare, assunti come target per le specifiche di MEMU.

[1] Interessante è notare il parallelismo tra elemento dissipativo meccanico (b) ed il suo corrispettivo elettrico (R_{eq}).

Parametro	**Simbolo**	**Valore**
Costante Elastica	k	$400\,\mathrm{N/m}$
Frequenza di risonanza	f_0	$25\,\mathrm{kHz}$
Coefficiente di trasduzione	η	$0.2 \times 10^{-6}\,\mathrm{V\,F/m}$
Tensione di bias	V_{dc}	$10\,\mathrm{V}$
Fattore di qualità	Q	10 000
Parametro Elettrico	**Simbolo**	**Valore**
Resistenza Equivalente	R_{eq}	$65\,\mathrm{M\Omega}$
Induttanza Equivalente	L_{eq}	$40\,\mathrm{MH}$
Capacità Equivalente	C_{eq}	$10\,\mathrm{aF}$

Tabella 1.2: Parametri del giroscopio FM a cui faremo riferimento come esempio numerico per la progettazione di MEMU.

In particolare, nelle varie fasi di progettazione abbiamo utilizzato come parametri tipici quelli di una struttura giroscopica a modulazione di frequenza [13], i cui valori sono illustrati in tabella 1.2.

Considerati i valori, che si derivano per i parametri elettrici equivalenti, è subito evidente come non sia realizzabile un "modello analogico" a componenti discreti: infatti si hanno valori di capacità molto piccoli e di induttanza molto elevati. L'idea è quindi quella di emulare il MEMS in digitale, andando, sostanzialmente, ad implementare un filtro a tempo discreto la cui risposta in frequenza emuli quella dell'equivalente "MEMS analogico" considerato, come mostrato in figura 1.6a.

Questa tesi si concentra esclusivamente sull'emulazione del comportamento di piccolo segnale, non prevedendo un corretto adattamento di impedenza (infatti inizialmente non considereremo le capacità C_0 viste dalle porte del MEMS verso massa), che verrà spiegato assieme alle prospettive future nel capitolo 5. In figura 1.5 possiamo vedere lo schema dell'RLC equivalente che tiene inoltre in considerazione alcune non idealità (capacità di feedthrough, che rappresenta l'accoppiamento capacitivo fra drive e sense, e rumore termomeccanico) e le due capacità C_0.

Per emulare il MEMS non possiamo limitarci allo schema ideale e perciò il nostro obiettivo comprenderà anche la sintetizzazione degli effetti dovuti a capacità di feedthrough e rumore termomeccanico.

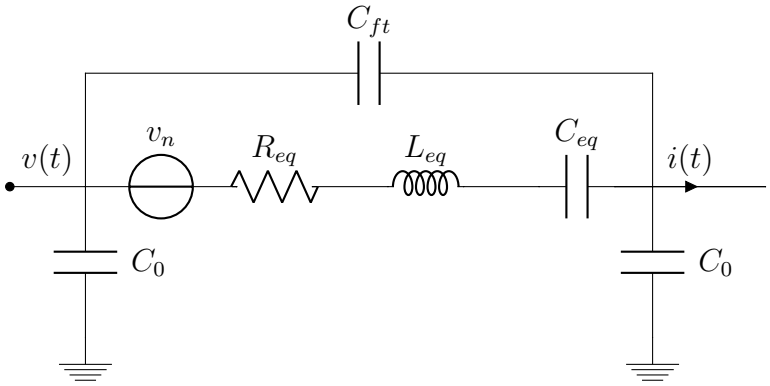
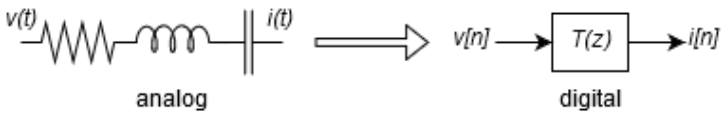
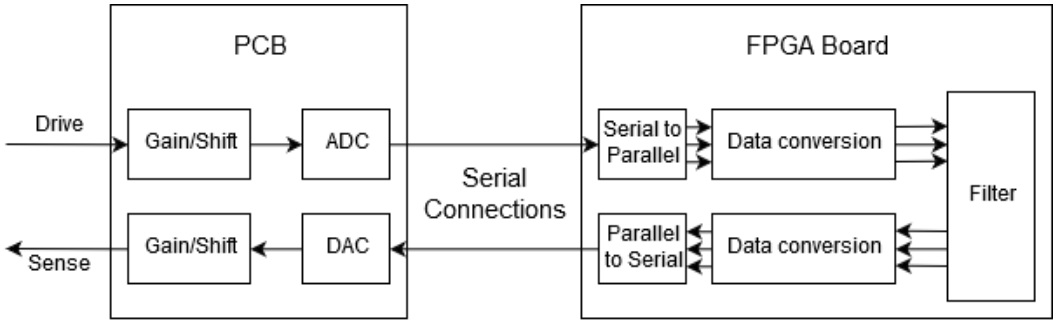


Figura 1.5: *RLC* con alcune non idealità.

1.3 Struttura della tesi



(a) Equivalente digitale



(b) Dettaglio della parte $T(z)$

Figura 1.6: Schema generale di MEMU.

Seguendo lo schema generale di MEMU (figura 1.6b) possiamo notare come questo sia costituito da una parte firmware ed una hardware:

- La scheda elettronica si occupa di creare un'interfaccia per MEMU con il mondo analogico, attraverso l'adattamento e la conversione dei segnali.

- La parte firmware, implementata in VHDL (VHSIC Hardware Description Language, dove VHSIC sta per Very High Speed Integrated Circuits) su di una FPGA, è a sua volta suddivisa in: (i) un filtro digitale che rappresenta il core principale di MEMU e (ii) una parte che si occupa della conversione dei dati tra seriale e parallelo e tra i formati necessari.

Il lavoro da me svolto in questi mesi si può sostanzialmente suddividere in quattro fasi principali, che riflettono il susseguirsi dei successivi capitoli:

- Nel **capitolo 2** si affronta l'implementazione del firmware su FPGA (Field Programmable Gate Array) del filtro digitale tramite linguaggio VHDL; discutendo anche della problematica dell'emulazione del rumore termomeccanico tramite un generatore di AWGN (Additive White Gaussian Noise).

- Il **capitolo 3** tratta la realizzazione di una scheda elettronica (PCB - Printed Circuit Board) che include ADC e DAC e la scelta dei componenti annessi. Si analizza inoltre la connessione tra FPGA e PCB ed il necessario passaggio di dominio (da analogico a digitale e viceversa).

- Nel **capitolo 4** si parlerà del set-up e delle misure effettuate, partendo dai casi più semplici (MEMS ideali) e crescendo progressivamente in complessità (feedthrough e rumore).

- Infine il **capitolo 5** illustra quali sono le conclusioni attuali e le prospettive future del progetto, con le possibili evoluzioni di MEMU.

Capitolo 2

Implementazione firmware di MEMU

2.1 Scelta della piattaforma

Quando si desidera sviluppare un sistema digitale vi sono diverse architetture da considerare:

- Microcontrollori - MCU
- Digital Signal Processors - DSP
- Field Programmable Gate Array - FPGA
- Application Specific Integrated Circuit - ASIC

Vediamo nel dettaglio vantaggi e svantaggi di ognuna di queste architetture:

Microcontrollori - MCU

I microcontrollori sono dei dispositivi general-purpose che si possono adattare ad una gran varietà di applicazioni software. Comportano una spesa contenuta per unità poiché i costi di sviluppo fissi (NRE - NonRecurring Engineering) vengono condivisi tra tutti gli utilizzatori di una determinata famiglia di MCU; inoltre godono di una bassa difficoltà di sviluppo grazie allo sfruttamento di

librerie standard e programmazione ad alto livello. Allo stesso tempo questi hanno prestazioni inferiori rispetto alle altre architetture considerate. Infatti, dato il loro essere general-purpose, tipicamente non implementano operazioni avanzate in hardware.

Digital Signal Processors - DSP

I DSP forniscono delle architetture hardware preconfigurate e dedicate allo svolgimento di determinati compiti; questo consente una maggior efficienza rispetto ad un microcontrollore nello sfruttamento dell'area ed anche maggiori prestazioni, ad esempio sono solitamente dotati di un circuito moltiplicatore (istruzione MAC). Tendono inoltre ad avere una efficienza energetica migliore: essi, infatti, sono spesso pensati per soluzioni embedded ed esecuzioni real-time. Questi vantaggi vanno però a scapito della flessibilità, in quanto ogni core hardware sarà limitato a svolgere una certa funzione per la quale è stato progettato.

Field Programmable Gate Array - FPGA

Una FPGA, tramite uno sforzo di programmazione superiore ed una minor efficienza energetica rispetto alle architetture analizzate in precedenza, consente di sintetizzare un circuito logico. Questa differenza porta dei grossi vantaggi in termini di flessibilità e riconfigurabilità ed inoltre consente un alto grado di parallelismo delle operazioni, andando però a perdere in termini di costi per unità. Inoltre, in questi dispositivi, l'area viene impiegata in modo sub-ottimo a causa di: (i) la necessità di un "tessuto" (detto *fabric*) di interconnessioni (figura 2.1) utili al collegamento dei vari blocchi logici e (ii) un'area prefissata che non è detto venga completamente sfruttata dall'applicazione.

Application Specific Integrated Circuit - ASIC

Un ASIC è la soluzione assolutamente migliore dal punto di vista di prestazioni ed efficienza, sia in termini energetici che di silicio utilizzato. Questa architettura però comporta dei tempi di sviluppo molto più lunghi, modesta flessibilità e dei costi NRE di ordini di grandezza superiori rispetto alle altre

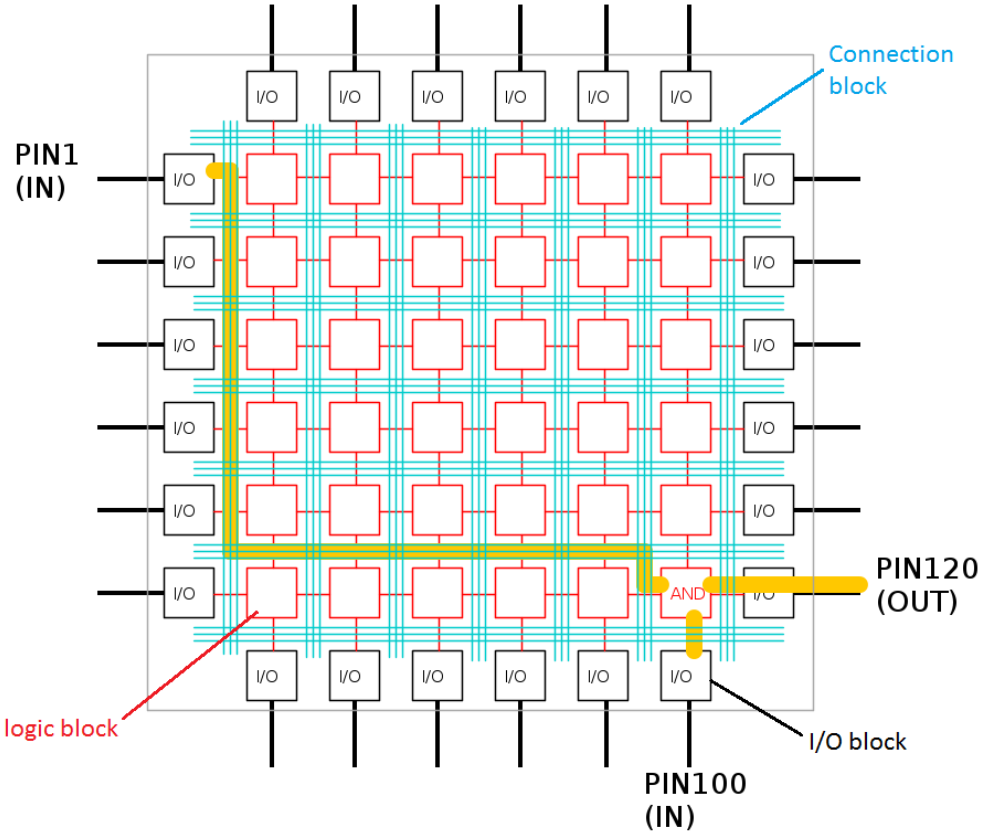


Figura 2.1: Schema della struttura interna di una FPGA con un semplice esempio di implementazione costituito da una operazione di *AND* logico.

soluzioni. Inoltre si incontra un ulteriore allungamento dei tempi dovuto alla fabbricazione dei chip.

Le caratteristiche di questo progetto richiedono che il device si possa riconfigurare per emulare diversi dispositivi. Questo ci porta subito a restringere la scelta tra un microcontrollore ed un FGPA.
Si vorrebbe inoltre poter aggiungere/rimuovere gradi di complessità al modello senza avere un eccessivo impatto sulle prestazioni; perciò il parallelismo del FPGA è la caratteristica più adatta a questo tipo di funzione, infatti a questo punto l'unico fattore limitante sarebbe l'area disponibile del dispositivo.
Prendiamo come esempio il caso in cui si voglia simulare più di un modo di risonanza di un MEMS: con un FPGA basta porre in parallelo tanti rami quanti sono i modi da emulare; la loro esecuzione avverrà in contemporanea ed avremo solo bisogno di sommare il risultato delle varie uscite prima di restituirlo. Possiamo così vedere come, a fronte di un basso aumento di latenza totale (ossia il tempo di somma), possiamo aumentare la complessità del nostro sistema emulato.
Nel caso si vogliano effettuare dei test su architetture differenti basta effettuare una riprogrammazione del firmware, potendo sfruttare le porte logiche del dispositivo per implementare di volta in volta differenti soluzioni. Inoltre la risultante implementazione, essendo un circuito logico, avrà migliori prestazioni rispetto all'esecuzione sequenziale di un MCU, sfruttando funzioni quali ad esempio il pipelining.

2.2 Generazione dei parametri del filtro digitale

Dato che il nostro obiettivo è emulare digitalmente l'ammettenza dell'RLC del MEMS, la funzione di trasferimento "analogica" $Y(s)$ (1.3) da sintetizzare deve essere trasferita in digitale ($Y(z)$); questo avviene mappando il dominio s dell'RLC in quello z, tramite una trasformazione, come mostrato in figura 2.2.

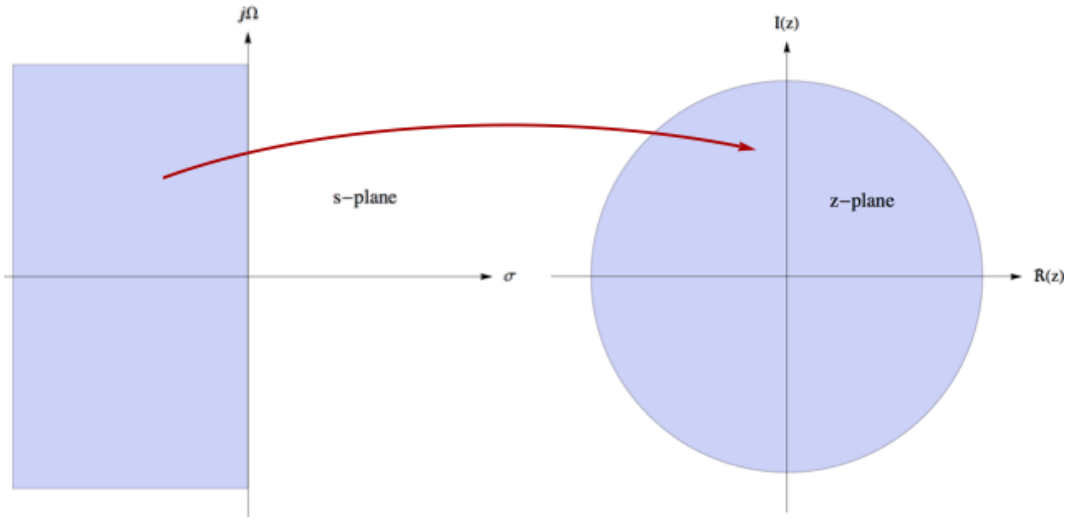


Figura 2.2: Trasformazione da dominio s a z con la bilineare. Per conservare le caratteristiche di stabilità i poli del semipiano sinistro dovranno essere mappati all'interno della circonferenza unitaria (cioè con $s = \sigma + j\Omega$, se $\sigma < 0$ allora $|z| < 1$).

La trasformazione da noi usata è quella bilineare [14] che, a differenza di altre (metodo di invarianza all'impulso o mappatura poli e zeri, per esempio), consente la trasformazione di qualsiasi tipo di filtro (passa basso, passa banda, passa alto) nel corrispondente digitale.

$$s \leftarrow \frac{2}{T}\frac{z-1}{z+1} \tag{2.1}$$

dove T è il periodo di campionamento, che nel nostro caso, come vedremo in seguito, sarà di 1 µs.

Nel nostro scopo di emulare un MEMS è critico considerare la distorsione in frequenza, come si può vedere in figura 2.3, introdotta dalla trasformazione

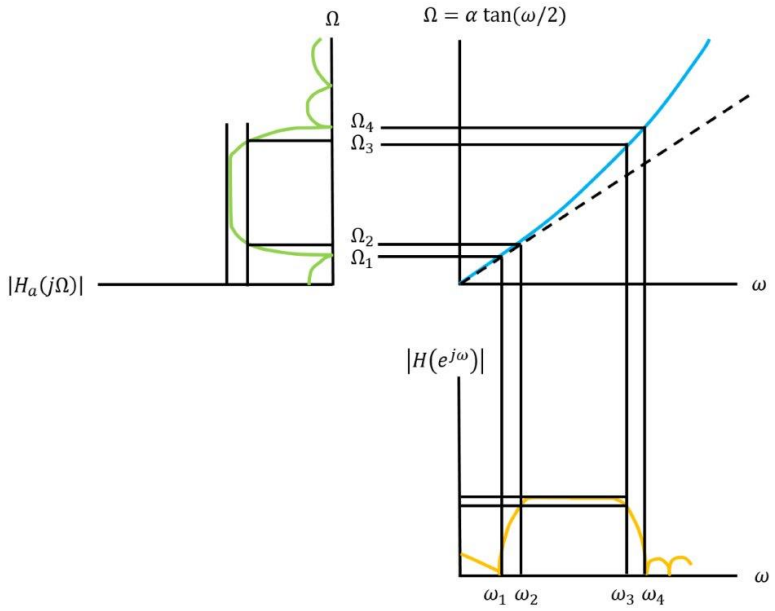


Figura 2.3: Effetto di distorsione in frequenza introdotto dalla trasformazione bilineare.

bilineare. Questa può essere compensata nell'intorno del punto di maggiore interesse tramite una pre-distorsione (equazione 2.2) [15], che fa in modo di compensare quella introdotta dalla bilineare.
Nel nostro caso questa sarà applicata alla frequenza di risonanza: visto che è questa a costituire il punto di maggior interesse per la maggior parte dei dispositivi MEMS, è qui che vogliamo che l'emulazione sia il più fedele possibile.

$$s \leftarrow \frac{\omega_0}{\tan\left(\frac{\omega_0 T}{2}\right)} \frac{z-1}{z+1} \tag{2.2}$$

che per $\omega_0 \to 0$ diventa di nuovo la trasformazione originale vista nell'equazione 2.1.

Il filtro così ottenuto è di tipo IIR. Tale trasformazione è stata applicata tramite un programma scritto nel software MATLAB, che ci consente, inoltre, di esprimere il filtro in SOS (Second Order Section), ossia sezioni del secondo ordine che saranno multiple e combinate in cascata in caso di necessità.

Allo stesso tempo è stata effettuata la scelta della rappresentazione in virgola mobile (floating-point), per assicurarsi la corretta rappresentazione di anche piccole variazioni su numeri grandi (i.e. un $Q = 1\text{M}$ con drift di 0.1), in quanto questa rappresentazione ci consente un maggiore range dinamico rispetto a quella in virgola fissa; questo ci tornerà particolarmente utile, come vedremo più avanti, per mitigare uno svantaggio della struttura usata per il filtraggio.

Per il numero di bit sono state seguite le linee guida dello standard IEEE 754-1985 [16] e la sua revisione del 2008 [17].

Esempio 2.2.1. Facciamo un breve esempio prendendo in considerazione i valori numerici riferiti al giroscopio FM visto in precedenza (tabella 1.2) e la funzione di trasferimento del risonatore (eq. 1.3). Usando la trasformazione bilineare in MATLAB (con prewarping sulla frequenza di risonanza f_0) otteniamo i coefficienti di numeratore e denominatore della funzione di trasferimento digitale:

$$Y(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.3}$$

con

```
b0 =  1.2323475573e-14;
b1 = -2.2204460492e-16;
b2 = -1.2101430968e-14;
a0 =  1;
a1 = -1.9997516944;
a2 =  0.9999984292;
```

ora con un'altra funzione MATLAB possiamo trovare i coefficienti del filtro nella forma *sos* e g, sostanzialmente andando a normalizzare anche il numeratore raccogliendo il guadagno g:

```
sos(1) =  1;            % b0
sos(2) = -0.0180180180; % b1
sos(3) = -0.9819819819; % b2
sos(4) =  1;            % a0
```

```
sos(5) = -1.9997516944; % a1
sos(6) =  0.9999984292; % a2
g      =  1.2323475573e-14;
```

con la funzione di trasferimento che ora sarà:

$$Y(z) = gY_1(z) = g\frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.4}$$

Questi sono i valori numerici che costituiranno i coefficienti del nostro filtro digitale per emulare correttamente il comportamento del giroscopio FM da cui siamo partiti.

In figura 2.4 possiamo vedere fase e modulo del trasferimento del filtro digitale, con e senza prewarping, a confronto con quello analogico. Dai riquadri di zoom si nota come la trasformazione che ha sfruttato il prewarping approssimi in modo migliore la funzione di trasferimento tempo continua iniziale, attorno alla frequenza di risonanza, rispetto alla semplice trasformazione bilineare.

Anche dal confronto delle risposte allo scalino (figura 2.5) possiamo concludere che il modello digitale emula in modo corretto quello analogico. Nei due riquadri possiamo vedere uno zoom dell'istante iniziale ed uno dopo che sono trascorsi 0.5 s: in entrambi il comportamento dei due sistemi è lo stesso.

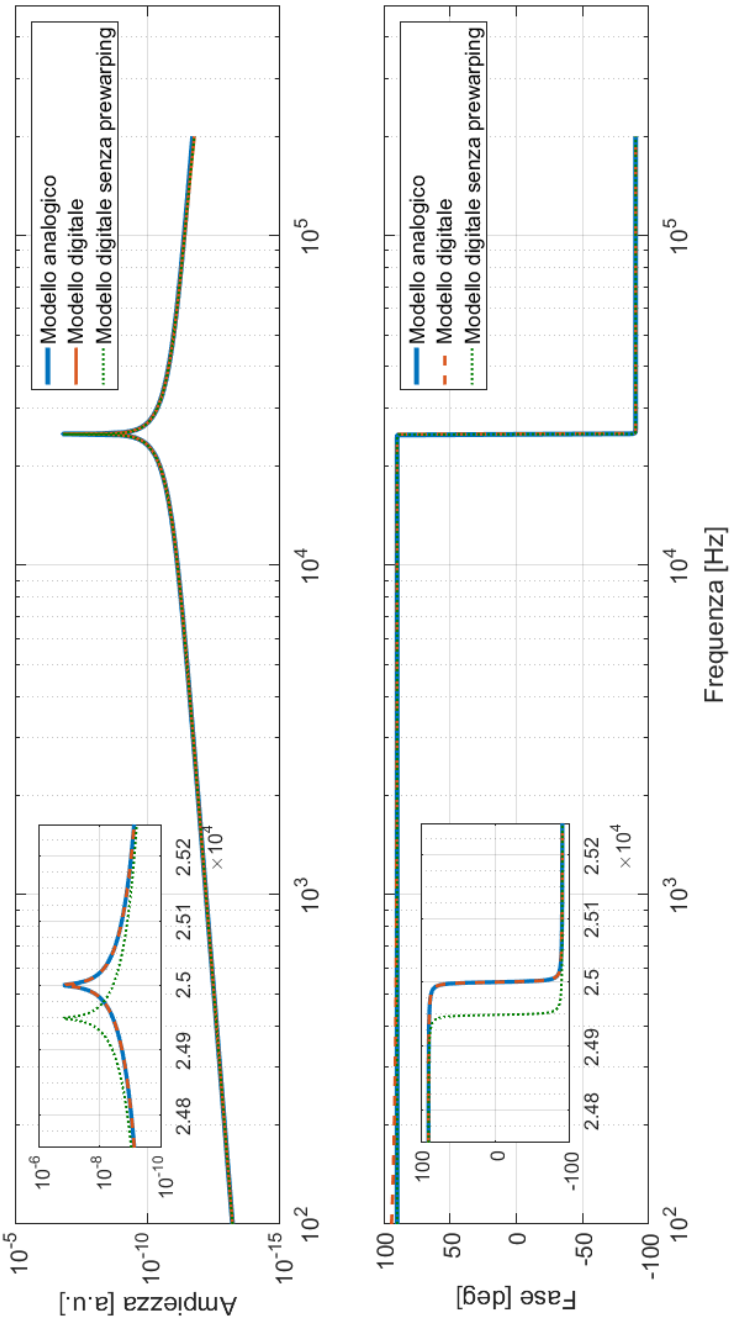


Figura 2.4: Confronto fra diagrammi di Bode di modello analogico, digitale, e digitale senza prewarping.

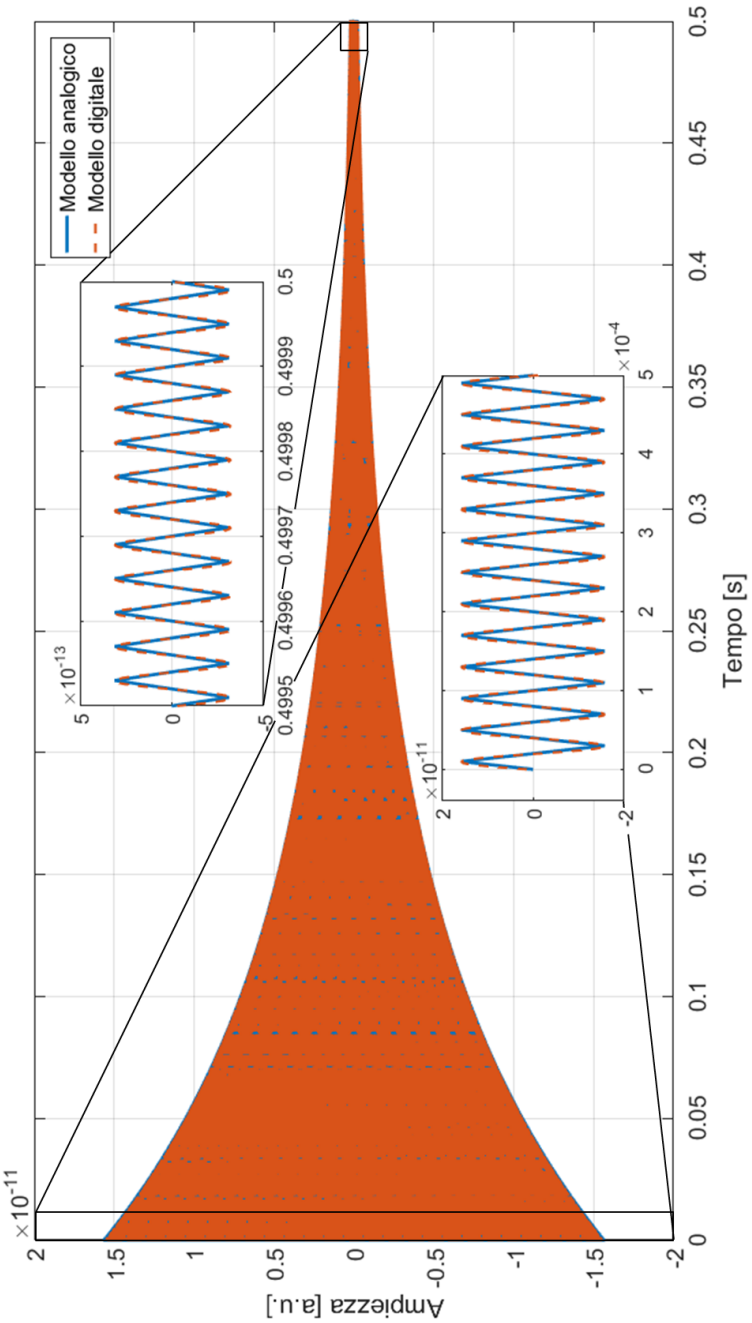


Figura 2.5: Confronto fra risposta allo scalino dei sistemi analogico e digitale.

2.3 Sintesi del filtro

Per l'implementazione del filtro interno a MEMU è stata scelta una scheda di sviluppo prodotta da XILINX, più precisamente lo SP601 Evaluation Kit, che monta una FPGA Spartan-6 XC6SLX16-2CSG324, in quanto già disponibile e con prestazioni sufficienti a questa implementazione.
Noto il filtro digitale da implementare, ottenuto tramite il procedimento descritto in 2.2, occorre scegliere la topologia con cui implementare fisicamente il filtro. La scelta è ricaduta su una cascata di filtri IIR del secondo ordine nella forma DFII (Direct-Form-II), illustrata in figura 2.6. Questo poichè la DFII permette di risparmiare area implementando una sola linea di delay (rispetto alla DFI che ne utilizza il doppio), infatti è canonica rispetto al ritardo.[1]
Inoltre, sempre in figura 2.6, sono stati assegnati dei nomi alle operazioni aritmetiche da effettuare, nel formato OP_i; questi verranno d'ora in poi usati per riferirsi ad esse.

Utilizzando codifica floating-point e sezioni del secondo ordine in cascata evitiamo due tipici difetti della struttura DFII [14]:

1. tramite l'impiego del floating-point non dobbiamo preoccuparci di utilizzare guard-bits per prevenire l'overflow dei dati numerici all'ingresso della linea di ritardo, cosa che accadrebbe in caso di fixed-point; questo avviene perchè in questa struttura i due poli precedono i due zeri creando spesso un'alta amplificazione del segnale nel ramo di feedback, che verrà poi attenuato in quello di feedforward, ossia dove intervengono gli zeri.

2. usando sezioni del secondo ordine diminuiamo la sensibilità agli errori di quantizzazione nei coefficienti del filtro (tipico problema che interessa tutti i filtri espressi nella forma poli-zeri, soprattutto di ordini alti), che è comunque già attenuata grazie alla codifica scelta.

[1] In generale, un filtro digitale di ordine N si dice canonico rispetto al ritardo quando possiede N elementi di ritardo.

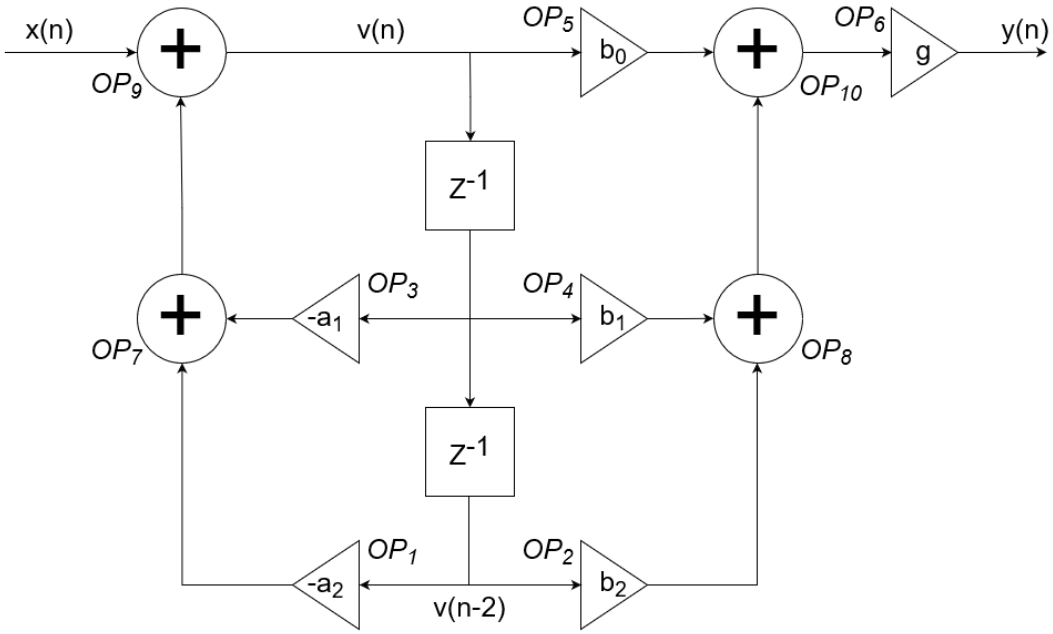


Figura 2.6: Implementazione nella forma Direct-Form-II di un filtro digitale del secondo ordine.

Per ogni sezione (figura 2.6) si avrà:

$$\begin{aligned} v(n) &= x(n) - a_1 v(n-1) - a_2 v(n-2) \\ y(n) &= g[b_0 v(n) + b_1 v(n-1) + b_2 v(n-2)] \end{aligned} \tag{2.5}$$

Partendo da queste equazioni è stato progettato un core in VHDL (figura 2.7) basato su:

- una macchina a stati di tipo Moore (in cui l'uscita dipende solo dallo stato corrente) che si occupa del flusso dei dati e della gestione e/o handshake degli altri componenti presenti nel blocco
- un sommatore (e sottrattore) floating-point a 32 bit generato tramite lo XILINX Core-Generator
- un moltiplicatore floating-point a 32 bit generato tramite lo XILINX Core-Generator

- 6 multiplexer usati per instradare i diversi dati a moltiplicatore e sommatore e portare i dati in uscita nei registri e porte corrette
- una RAM ampia 6 x 32 bit a cui viene assegnato il compito di linea di delay e di memorizzazione dei dati parziali
- 4 registri da 32 bit usati per sincronizzazione e conservazione dei dati significativi

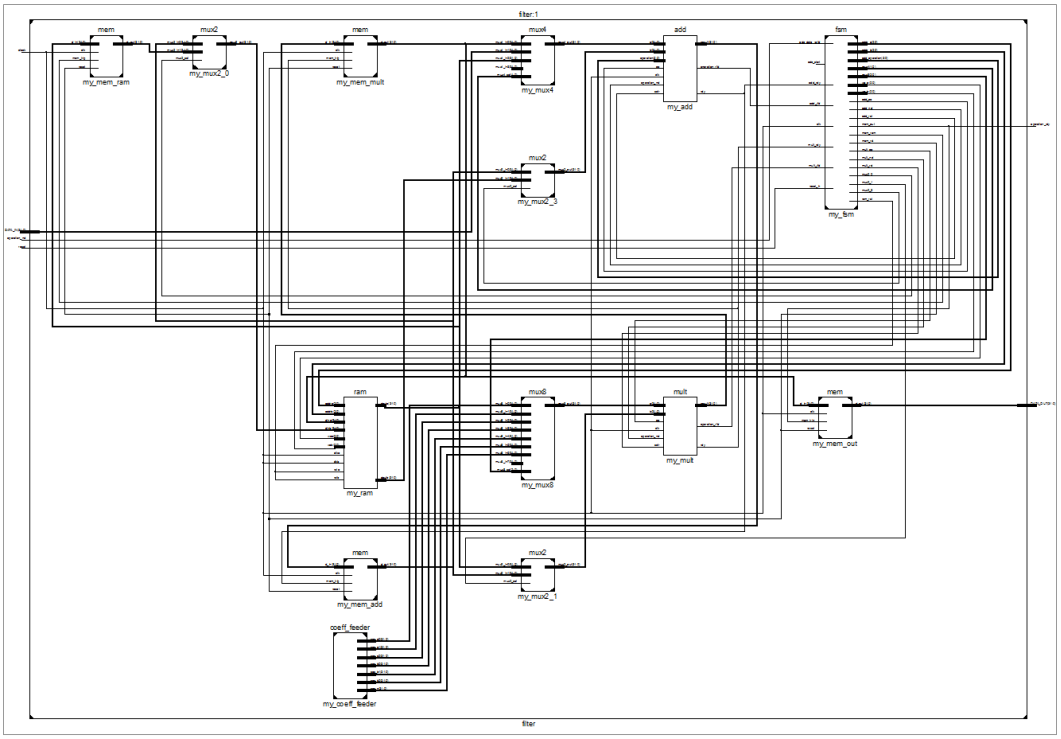


Figura 2.7: Struttura del core del filtro.

I vari componenti della struttura ricevono un clock alla frequenza di 100 MHz, proveniente da un blocco DCM (Digital Clock Manager) che ha in ingresso un clock differenziale a 200 MHz derivato da un oscillatore presente sulla scheda. Grazie a questo clock veloce avremo molti fronti nell'arco di un periodo di sample, riusciremo così a compiere tutte le operazioni necessarie su un campione prima che il successivo sia disponibile all'ingresso.

Questa soluzione è stata pensata per essere utilizzata a livello superiore come una blackbox (figura 2.8) tramite le seguenti porte:

- *ingresso* reset: per un segnale di reset attivo alto che svuota i registri e ferma l'esecuzione
- *ingresso* clock: per un segnale di clock in ingresso a 100MHz
- *ingresso* DATA_IN: per un dato in ingresso in codifica floating-point 32 bit
- *ingresso* operation_nd: un bit di valido per segnalare l'arrivo di un dato nuovo (newdata)
- *uscita* DATA_OUT: per un dato in uscita in codifica floating-point 32 bit
- *uscita* operation_rdy: un bit per segnalare un nuovo dato valido in uscita (ready)

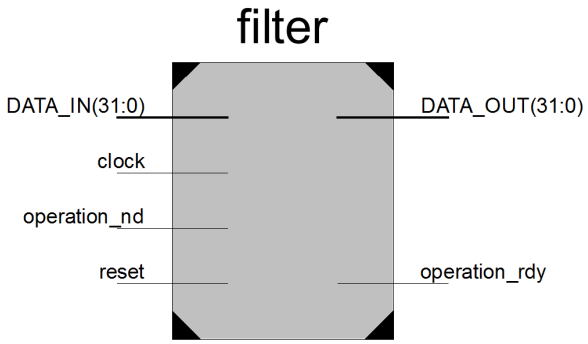


Figura 2.8: Visualizzazione schematica di ingressi e uscite del filtro.

La struttura delle due porte dedicate ai dati (DATA_IN e DATA_OUT) e delle due dedicate ai segnali di valido (operation_nd e operation_rdy) verrà mantenuta per tutti i blocchi che vedremo successivamente. Questo ci consentirà di interfacciarli tra loro ad un alto livello di astrazione, ossia considerandoli come blackbox.

2.3.1 Progettazione della macchina a stati

Per progettare la macchina a stati, che temporizza l'esecuzione delle operazioni richieste per l'implementazione del filtro digitale, ci si è basati sull'insegnamento Digital System Design Methodologies [18], in cui si è affrontato questo argomento. Inizialmente è stato scelto di utilizzare un solo sommatore ed un solo moltiplicatore, che quindi verrano multiplexati nel tempo, in modo tale da avere un risparmio di area di silicio. Dato che abbiamo diverse operazioni per il filtraggio di ogni singolo campione sarebbe utile ottimizzare la macchina stati che le gestisca in modo da parallelizzare il più possibile questi passaggi, impiegando così meno colpi di clock per avere il risultato.

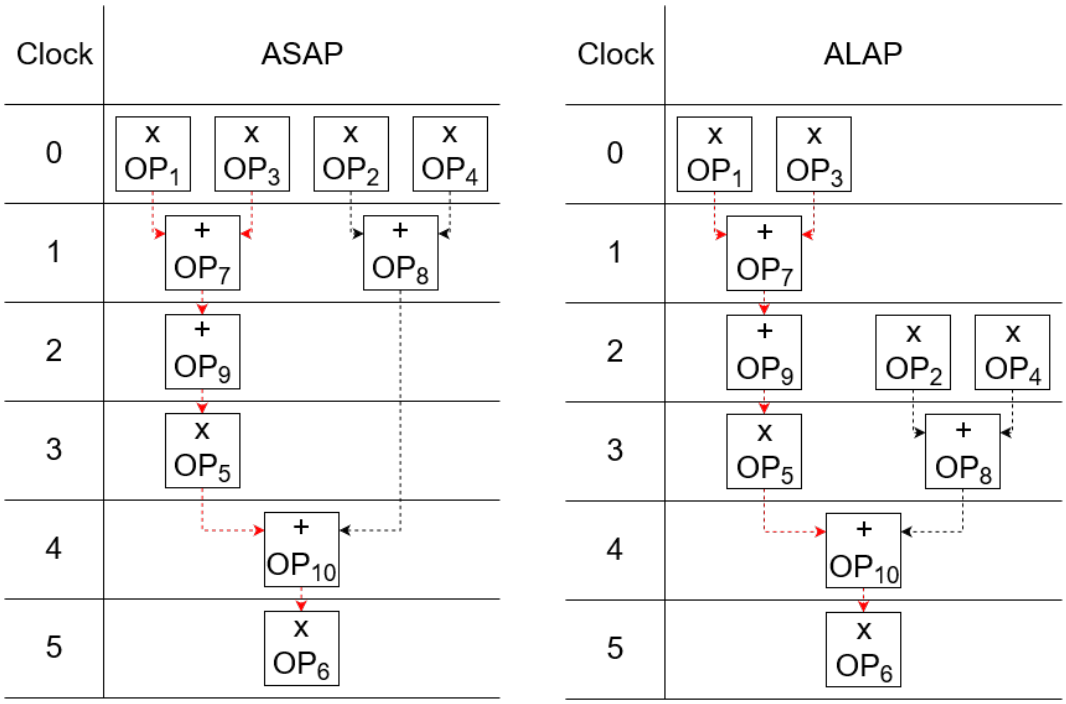
Scheduling

Partendo dalla struttura del filtro IIR DFII (figura 2.6), si è iniziato facendo lo scheduling delle operazioni senza tenere in considerazione la limitazione del numero dei blocchi aritmetici. Sono stati utilizzati:

- il metodo ASAP (As Soon As Possible), che schedula un'operazione appena i dati necessari ad essa sono disponibili;
- il metodo ALAP (As Late As Possible), che pianifica l'esecuzione in modo che il tempo totale sia definito dai cammini critici e che le altre operazioni vengano eseguite appena in tempo per avere i dati pronti al momento necessario.

Per queste fasi si è considerato il tempo di esecuzione delle due operazioni utilizzate (addizione e moltiplicazione) normalizzato a 1, siccome essi impiegano lo stesso numero di colpi di clock.

Dati i due grafici si è calcolata la mobilità S (Slack) di ogni operazione, come la differenza in numero di colpi di clock tra un grafico e l'altro nell'istante di esecuzione di quel determinato operatore. Ovviamente tutti quelli appartenenti al cammino critico avranno $S = 0$ e potranno essere avviati solo ad un determinato istante del processo. Una mobilità maggiore di 0 ci indica l'intervallo di tempo tollerato in cui un'operazione può iniziare l'esecuzione senza andare a costituire un nuovo cammino critico. Notiamo, dunque, che le uniche



(a) Scheduling ASAP (b) Scheduling ALAP

Figura 2.9: Confronto fra scheduling ASAP e ALAP per la FSM del filtro implementato. Dalle due immagini emerge che le uniche differenze sono date dallo scheduling di: OP_2, OP_4 e OP_8. In rosso sono stati evidenziati i cammini critici.

operazioni con una mobilità non nulla sono la somma OP_8 e le moltiplicazioni OP_2 e OP_4 che hanno tutte $S = 2$. Da questi dati possiamo ora ricavare il nostro scheduling delle operazioni partendo dall'albero ALAP e adattandolo spostando i nodi OP_i con $S_i > 0$ massimo di tanti colpi di clock quanto è il valore del loro Slack S_i se non vogliamo aumentare la latenza totale. Nel nostro caso notiamo come questo spostamento non sia sufficiente e quindi si andrà a creare un nuovo cammino critico più lungo del precedente. Il risultato che otteniamo dopo questa operazione è lo scheduling che useremo per la nostra macchina a stati ed è rappresentato in figura 2.10:

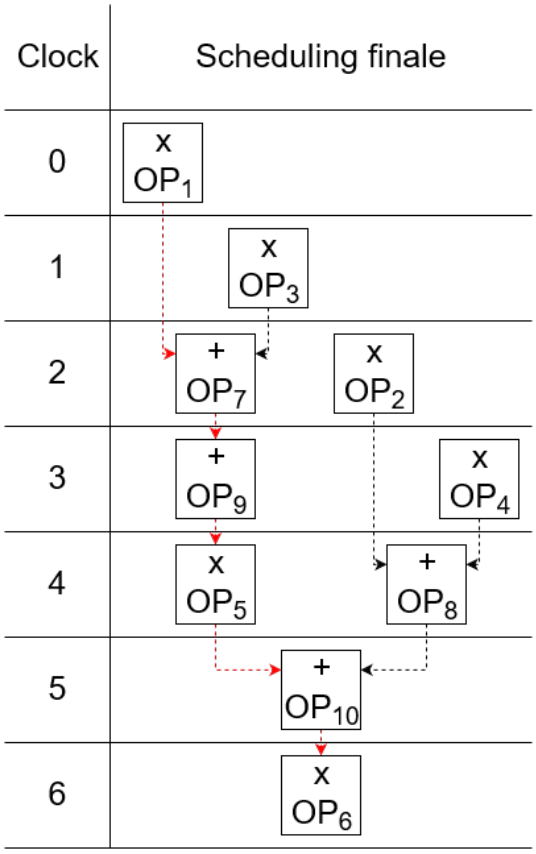


Figura 2.10: Scheduling finale scelto per la FSM di MEMU.

Register Binding

Dopo aver definito l'ordine di esecuzione delle operazioni dovremo stabilire il register binding per tutti quei risultati parziali che non vengono utilizzati appena ottenuti: dovremo perciò salvarli in modo da poterli richiamare quando saranno necessari all'esecuzione; un elemento di memoria ci sarà inoltre utile per sintetizzare le funzioni di ritardo presenti nel filtro. Si è così scelto di implementare una memoria RAM a 32 bit (dato che il filtro usa variabili float di tipo *single*). Dato l'esiguo numero di registri impiegati si è deciso di lasciarli ognuno dedicato alla propria variabile invece che condividerli, questo per fare in modo di avere un'esecuzione più semplice a fronte di una perdita di logica impiegata risibile.

Avremo così un blocco di RAM con 6 indirizzi che verranno assegnati nel seguente modo:

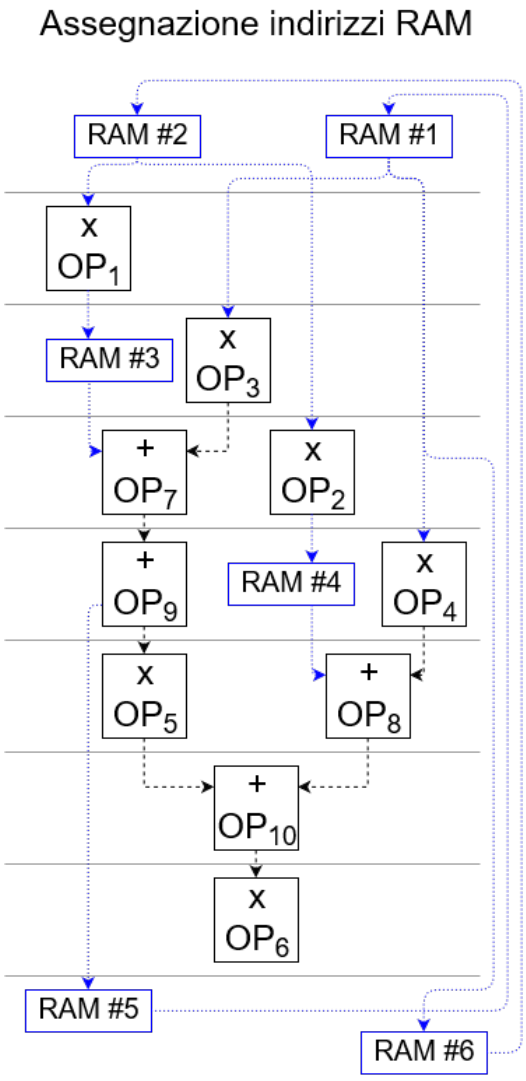


Figura 2.11: Schema dell'assegnazione degli indirizzi e dell'utilizzo della RAM usata per l'implementazione del filtro.

- indirizzo 1 dedicato al primo delay, che fornirà dati in ingresso a OP_3 e OP_4.
- indirizzo 2 dedicato al secondo delay, che fornirà dati in ingresso a OP_1 e OP_2.

- indirizzo 3 per memorizzare il risultato di OP_1 che altrimenti sarebbe stato corrotto da OP_3 prima di arrivare a OP_7
- indirizzo 4 per memorizzare il risultato di OP_2 che altrimenti sarebbe stato corrotto da OP_4 prima di arrivare a OP_8
- indirizzo 5 dedicato a salvare il risultato di OP_9 che verrà poi trasferito all'indirizzo 1 prima del ciclo successivo
- indirizzo 6 dedicato a salvare il valore presente all'indirizzo 1 che verrà poi trasferito all'indirizzo 2 prima del ciclo successivo

2.4 Simulazione e verifica

Dopo aver sviluppato il core del filtro si è passati ad una fase di verifica preliminare. Qui, infatti, testeremo solo il filtro digitale, paragonando poi la sua uscita con quella prevista dal modello MATLAB. In modo più specifico andremo a testare solo la corretta funzionalità tramite una simulazione di tipo *behavioral*.

Una simulazione behavioral utilizza un alto livello di astrazione per la modellizzazione del design. Potrebbe, per esempio, contenere operazioni di alto livello come una somma a 4-bit (non un sommatore, come in un design strutturale) senza specificare come essa verrà poi effettivamente implementata (infatti potrebbe anche non essere fisicamente implementabile). Il tool di sintesi utilizza questo modello per inferire la struttura, composta da blocchi logici e le rispettive connessioni, che verrà poi utilizzata per realizzare la simulazione.

La simulazione viene effettuata con una descrizione Hardware Description Language (HDL) pre-sintesi. Tra i diversi tipi di simulazione possibili quella behavioral è la più rapida, ma anche quella che fornisce meno informazioni sul design (ritardi dei gate, delle connessioni...). Questo ci consente di verificare sintassi e funzionalità senza, per ora, occuparci di eventuali problemi di timing. Durante lo sviluppo di un design, buona parte della verifica è effettuata con questa modalità in modo da evidenziare presto e velocemente eventuali errori, che potranno così essere risolti con minor sforzo proprio perché individuati durante le fasi iniziali.

Per le simulazioni sono stati realizzati diversi testbench per le diverse prove che si sono volute effettuare. Principalmente sono stati usati quattro tipi di input:

impulso: per simulare un ingresso impulsivo si è usato un contatore, il cui bit di overflow funge da select a un multiplexer; questo ha sull'ingresso 0 il valore 0 in codifica floating-point e a quello 1 il valore 1. Dopo l'overflow il contatore viene disattivato in modo che la condizione non si ripeta. Si noti che modificando la dimensione del contatore e il suo valore iniziale si può scegliere a che istante $\bar{t}$ avere l'impulso.

scalino: rispetto al caso precedente l'unica differenza è che l'overflow viene utilizzato come input di un blocco logico la cui uscita pilota il multiplexer. La sua funzione è quella di tenere, dopo il primo rising-edge sull'ingresso, l'uscita alta anche quando il suo input tornerà al valore logico basso.

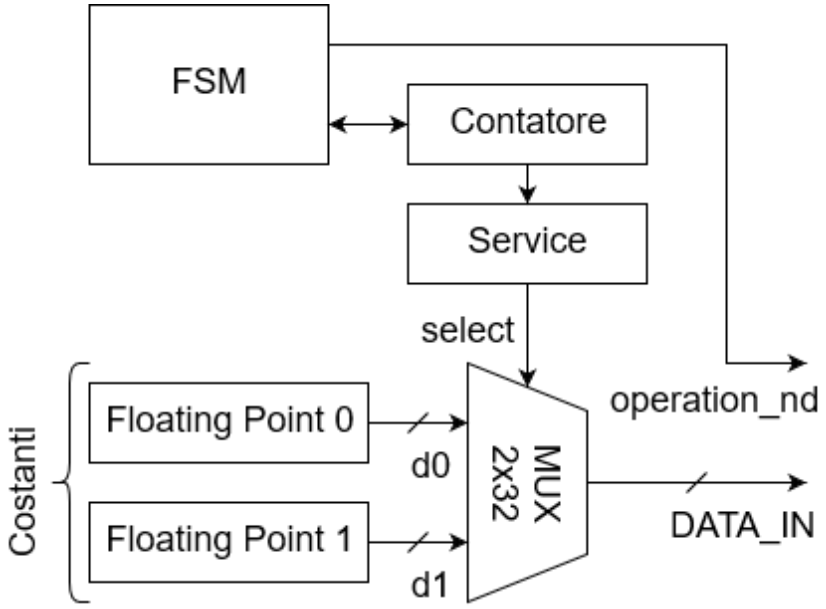
onda quadra: in questo caso si fa in modo che il tempo tra due overflow del contatore sia la metà del periodo che si vuole sintetizzare; il blocco logico in cascata ad esso effettua un toggle del valore in uscita per ogni rising-edge sul suo ingresso, andando così a creare un'onda quadra.

sinusoide: data la maggiore complessità del segnale in ingresso è stato deciso di utilizzare la lettura da file, il quale è stato precedentemente popolato con uno script MATLAB.

In ognuno dei casi sopracitati l'uscita del blocco viene salvata in un file di testo ogni volta che si ha un rising-edge sulla porta di *operation_ rdy*, ossia sul bit di valido in uscita dal filtro. Così è stato verificato che il comportamento del filtro digitale realizzato in MATLAB, tramite la trasformazione bilineare, e quello del modello behavioral scritto in VHDL siano identici.

I testbench per impulso, scalino e onda quadra sono stati scritti in modo da essere sintetizzabili (struttura in figura 2.12), ossia in modo che si possano effettivamente implementare sull'FPGA; questo ci risulta utile nella successiva modalità di verifica, nella quale implementiamo fisicamente il filtro su FPGA e gli forniamo in ingresso i vari tipi di stimoli visti. Tramite CHIPSCOPE INTEGRATED LOGIC ANALYZER (ILA), un tool fornito con la suite XILINX, è possibile aggiungere al nostro progetto un core dedicato il cui compito è salvare il valore di un segnale, quando una certa condizione (trigger) è soddisfatta, in un buffer instanziato nell'FPGA; questo dati saranno poi trasferiti al computer tramite interfaccia USB con il software CHIPSCOPE PRO ANALYZER per una successiva analisi. Nel nostro caso, il segnale da memorizzare nel buffer sarà, ovviamente, $DATA\_OUT$ e il trigger sarà impostato sul rising-edge del segnale *operation_ rdy*, che segnala un nuovo dato disponibile (valido) all'uscita del filtro.

In figura 2.13 possiamo vedere il confronto tra le risposte all'impulso ottenute: (i) in questo modo, (ii) con simulazione behavioral e (iii) la corrispettiva



(a) Schema a blocchi logici del circuito che genera gli input in fase di test.

Input desiderato	**Funzione implementata dal blocco *Service***
Impulso	Corto circuito
Scalino	Singola commutazione da 0 a 1 al primo overflow
Onda quadra	Output toggle ad ogni overflow

(b) Implementazione del blocco *Service* in funzione dell'input che si vuole generare.

Figura 2.12: Schema della struttura utilizzata per sintetizzare gli input dei test effettuati sull'FPGA tramite ChipScope.

simulazione Matlab. Il filtro di riferimento è il solito $Y(z)$ dell'equazione (2.3), che sintetizza il giroscopio FM più volte citato.

Dalla figura si può notare uno sfasamento delle due forme d'onda digitali rispetto al modello analogico: questo è coerente con il risultato atteso, avendo il filtro un ritardo intrinseco che è la latenza del circuito digitale; si può individuare che il ritardo è di 1 µs, in linea con quanto previsto, avendo progettato il filtro con latenza di un campione, ed essendo la frequenza di campionamento 1 MHz.

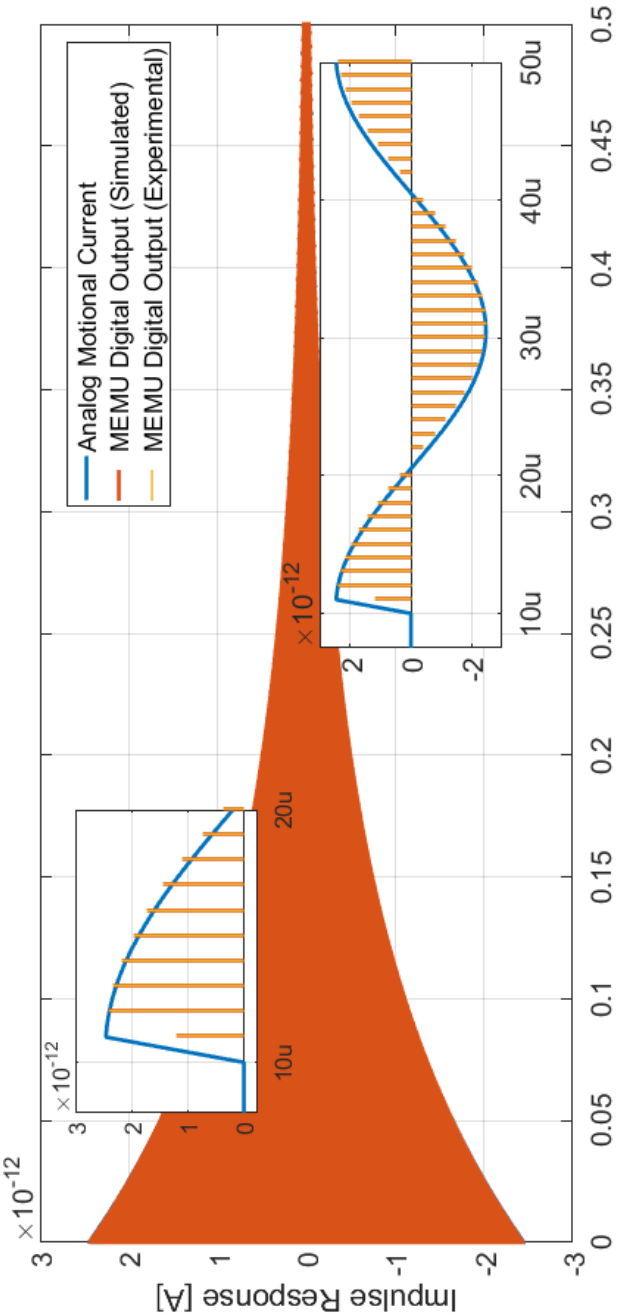


Figura 2.13: Confronto tra la risposta all'impulso del modello analogico del giroscopio FM e delle simulazioni behavioral e su FPGA tramite ChipScope.

2.5 Generazione di rumore bianco (AWGN)

Come accennato nell'introduzione, una feature importante di MEMU consiste nell'emulazione, non solo della funzione di trasferimento del MEMS, ma anche del rumore termomeccanico intrinsecamente generato dalla struttura. Vediamo ora come abbiamo implementato il generatore di rumore equivalente.
Dato il modello *RLC* equivalente visto in 1.2, sappiamo [10] che associato al risonatore ci sarà un generatore di rumore bianco equivalente in tensione la cui densità spettrale è:

$$\sqrt{S_{v_n}} = \sqrt{4K_B T R_{eq}} \tag{2.6}$$

Data la frequenza di campionamento di 1 MHz di MEMU, la corrispondente σ della realizzazione di rumore è:

$$\sigma = \sqrt{S_{v_n} \frac{1\,\text{MHz}}{2}} \tag{2.7}$$

Per l'emulazione dobbiamo quindi sintetizzare un generatore di numeri casuali a distribuzione gaussiana, la cui σ sia quella in equazione (2.7).
Per risolvere questo problema ci siamo basati su un application note di XILINX [19]. In questo documento si parla dell'utilizzo di Linear Feedback Shift Register per la generazione di numeri pseudocasuali; il risultato si ottiene partendo dalla struttura di uno Shift Register inizializzata ad un certo seed e utilizzando come nuovo dato in ingresso il risultato di una operazione XNOR tra alcuni suoi taps. Questa struttura ciclerà tra tutti i possibili stati del registro, tranne quello in cui tutti i bit hanno valore 1.
Importante è fare attenzione a inizializzare con un seed diverso da quello costituito da solo valori 1 perché questa è esattamente la condizione di lock-up del circuito, che altrimenti non si verificherebbe mai.[2]
Per scegliere i taps corretti ci si rifà ad una tabella presente nell'application note che indica quali usare per avere un tempo di ripetizione massimo in funzione della lunghezza scelta.

[2]Se fosse necessario includere questo caso, si potrebbe utilizzare un LFSR con porte XOR invece delle XNOR; in questa implementazione il caso di lock-up sarebbe composto da una stringa di soli 0. Sarà anche necessario cambiare in modo adeguato i taps utilizzati come dati in ingresso dell'operatore.

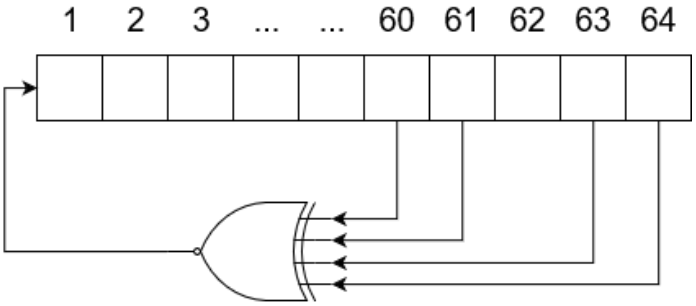


Figura 2.14: Implementazione del Linear Feedback Shift Register usato per la generazione di AWGN.

Nella nostra implementazione abbiamo scelto un LFSR a 64 bit, che utilizza i taps 64, 63, 61, 60 per il feedback (figura 2.14). Il feedback è stato fatto con porte XNOR in modo da tenere valido il caso di tutti i bit a '0'; poiché per loro natura i flip-flop di un FPGA tendono a essere resettati.
Il suo tempo di ripetizione è talmente alto che può non essere considerato un problema per qualsiasi scopo pratico. Infatti esso è definito come:

$$T_{ripetizione} = \left(2^{N_{taps}} - 1\right) \frac{1}{f_{clock}} \tag{2.8}$$

Che con $N_{taps} = 64$ e $f_{clock} = 100\,\text{MHz}$ diventa:

$$T_{ripetizione} \approx 1.8447 \times 10^{11}\,\text{s} \approx 5850\,\text{anni} \tag{2.9}$$

Da questa struttura otteniamo una sequenza di numeri che possiamo considerare casuale, ma essa avrà distribuzione uniforme in quanto tutti i numeri vengono ripetuti sistematicamente una sola volta. Come è possibile trasformare questa in una distribuzione normale?
Si può sfruttare il teorema del limite centrale [20]:

> Sia X_j una di n variabili aleatorie indipendenti e identicamente distribuite, e siano $E[X_j] = \mu$ e $Var[X_j] = \sigma^2\ \forall j$, con $0 < \sigma^2 < +\infty$ (ovvero $X_j \in \mathcal{L}^2$). Posto $Y_n = \dfrac{\sum_{j=1}^{n}(X_j - n\mu)}{\sigma\sqrt{n}}$ allora Y_n presenterà una distribuzione normale standard, $Y_n \to Y \sim N(0,1)$.

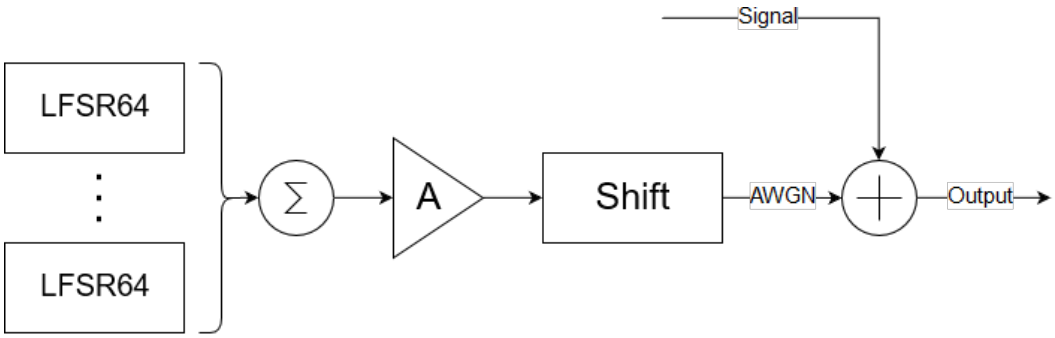


Figura 2.15: Schema semplificato del generatore di AWGN.

Nella nostra implementazione, sintetizzeremo quindi diversi LFSR inizializzati con diversi seed, in modo da rendere le variabili indipendenti.
Il risultato così ottenuto verrà inizialmente mappato su tutta la dinamica e poi scalato di un fattore di attenuazione A e allineato con il centro della dinamica del segnale (figura 2.15).

Capitolo 3

Firmware ausiliario e scheda elettronica

Poiché un MEMS, quale quello modellato nell'equivalente elettrico in figura 1.5, riceve normalmente in ingresso una tensione di attuazione analogica e restituisce in uscita una corrente altresì analogica, dobbiamo ora, come visto in precedenza in figura 1.6, interfacciare il filtro progettato nel capitolo 2 con il circuito tramite un ADC, un DAC e un'opportuna interfaccia sia hardware che firmware di condizionamento. Sarà infatti necessaria una parte di condizionamento firmware per portare i dati nel formato adatto e una PCB esterna per la parte di adattamento analogico.
Per quanto riguarda il firmware (i) non solo servirà un adattamento di formato da binario, proprio dei convertitori ADC e DAC, a floating point, proprio del filtro implementato, (ii) ma bisognerà inoltre parallelizzare i dati in ingresso dall'ADC e serializzare quelli in uscita verso il DAC.

La prima fase è stata costituita dallo scegliere i convertitori in base alle prestazioni richieste, iniziando dall'ADC:

- per questo primo prototipo, è stato considerato sufficiente che la risoluzione in ingresso sia minore di 100 µV su un range di 5 V, perciò ci serviranno almeno 16 bit (ottenendo così un $LSB \approx 70\,\mu V$). I limiti

prestazionali si incontreranno su segnali di ampiezza ridotta, quale ad esempio il rumore in ingresso;

- dato il range di frequenze considerato in tabella 1.1 e tenendo del margine di sicurezza, la frequenza di campionamento è stata fissata a 1 MHz;
- desideriamo inoltre avere una bassa latenza dei campioni, in modo da non introdurre un eccessivo sfasamento dei segnali (fattore critico ad esempio in un oscillatore), quindi la scelta è stata ristretta alla famiglia di ADC ad approssimazioni successive (SAR - Successive Approximation Register).

Detto questo la scelta è stata quella di utilizzare lo LTC2378-16 [21] prodotto da Linear Technology, un ADC a 16 bit di tipo SAR, differenziale e compatibile con una frequenza di campionamento di 1 MHz. Questo ha un'interfaccia seriale ed utilizza codifica in complemento a due. La scelta di un componente differenziale deriva dal desiderio di poter utilizzare dei segnali di drive sia single che double-ended; perciò l'elettronica verrà progettata di conseguenza.

Per quanto riguarda invece il DAC:

- vogliamo sempre una bassa latenza per il motivo visto in precedenza;
- dato che in uscita i MEMS hanno correnti che arrivano nell'ordine dei nA desideriamo un'architettura a basso rumore;
- è necessario che sia adatto alla generazione di forme d'onda accurate e quindi abbia alte prestazioni di linearità.
- il numero di bit imporrà il limite inferiore di ampiezza dei segnali sintetizzabili: sostanzialmente introdurrà una soglia che il rumore in uscita deve superare per poter essere visualizzabile ($LSB = V_{dd}/2^{N_{bit}}$).

La famiglia di DAC scelta è stata quella a moltiplicazione (MDAC - Multiplying DAC), caratterizzata da un settling time molto breve (fino a 0.05 µs) ed ottime prestazioni di rumore e linearità.

Il componente scelto è il DAC8811 [22] prodotto da Texas Instruments, un DAC a moltiplicazione a 16 bit, con input seriale e uscita in corrente a basso rumore, seguito da uno stadio a transimpedenza per portare l'uscita in tensione; questo è stato fatto per motivi di semplicità nel debug e nelle misure della prima implementazione di MEMU. Infatti con un'uscita in tensione è immediato interfacciarsi, ad esempio, con un oscilloscopio o un analizzatore di spettro, mentre la situazione sarebbe decisamente più complessa nel caso di un'uscita in corrente.

Il dato in ingresso al DAC dovrà essere adattato in modo da essere in formato binario a 16 bit.

3.1 Firmware ausiliario

3.1.1 Da ADC a filtro

Dal datasheet del componente LTC2378-16 [21] otteniamo il suo diagramma dei timing, che è rappresentato in figura 3.1.

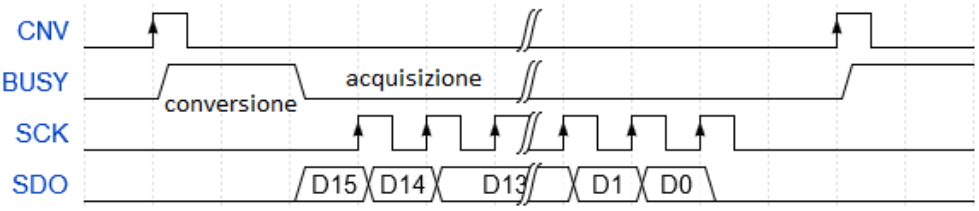


Figura 3.1: Diagramma del timing dell'interfaccia seriale dell'ADC LTC2378-16.

Per rispettare il diagramma dei tempi fornito è stata realizzata una macchina a stati (FSM) che gestisce la comunicazione seriale a 1 MHz.
Più precisamente il processo sarà il seguente:

- la macchina a stati richiede la conversione alzando per il tempo necessario il segnale CNV;
- l'ADC alza la linea di $BUSY$ ed effettua la conversione;
- a conversione terminata l'ADC è pronto per effettuare la trasmissione dei dati e abbassa il segnale di $BUSY$;
- la macchina a stati, dopo aver atteso la discesa della flag di $BUSY$, trasmette sulla linea SCK 16 colpi di clock a 50 MHz;
- l'ADC ad ogni fronte di salita di SCK mette un bit del dato da trasmettere sulla linea SDO, partendo dal bit più significativo (MSB - Most Significative Bit).

Dal lato FPGA i dati vengono inseriti in uno shift register a 16 bit con uscita parallela (anch'esso gestito dalla macchina a stati) ed una volta che tutti sono stati trasmessi (e quindi tutti si trovano sull'uscita), la macchina a stati alza una flag di dato valido (operation_rdy) per il blocco successivo.

A questo punto abbiamo un dato a 16 bit in parallelo in codifica complemento a due (Int16 - 16 bits Integer). Grazie ad un blocco implementato tramite lo XILINX Core Generator si effettua la conversione in floating-point a 32 bit, in modo che risulti compatibile con il core del nostro filtro digitale. In figura 3.2 possiamo avere uno sguardo generale sul percorso che compiono i dati durante il condizionamento precedente al filtraggio.

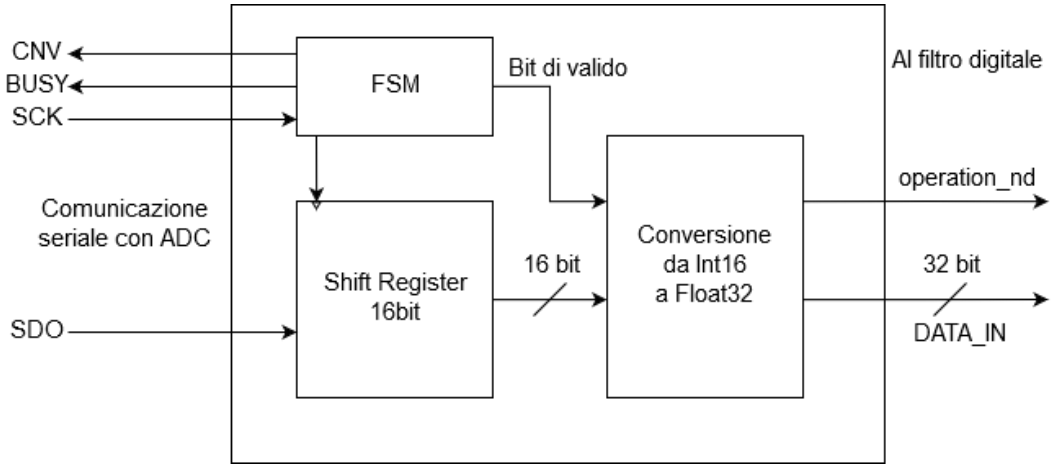


Figura 3.2: Schema completo (semplificato) del firmware implementato per connettere l'ADC al filtro.

3.1.2 Da filtro a DAC

In modo speculare a quanto fatto per l'ingresso siamo partiti dal considerare lo schema dei timing fornito dal datasheet del DAC8811 [22], che è stato riportato in figura 3.3.

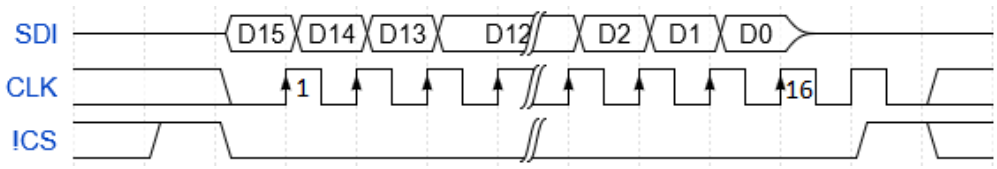


Figura 3.3: Diagramma del timing dell'interfaccia seriale del DAC8811.

Prima di tutto i dati in uscita dal filtro andranno convertiti nel formato opportuno, ossia da floating-point a 32 bit a binario a 16 bit (Uint16 - 16

bits Unsigned Integer); per fare questo ci serviremo di due blocchi: (i) uno derivato sempre dallo XILINX Core Generator che si occupa di trasformare il floating-point32 in binario a 16 bit in codifica complemento a due (Int16), (ii) mentre l'altro blocco, implementato in VHDL, compie la transizione finale a binario a 16 bit (Uint16).

Per dialogare con il DAC è stata dunque realizzata anche in questo caso una macchina a stati; la comunicazione avviene in questo modo:

- la presenza di un nuovo dato da convertire è segnalata dal blocco precedente tramite il segnale operation_rdy;
- la macchina a stati si occupa di caricare i bit del dato in parallelo in uno shift register con uscita seriale;
- la comunicazione seriale viene avviata abbassando la linea $!CS$;
- da questo momento, ad ogni colpo di clock sulla linea CLK (pilotata dalla FSM), il DAC legge un bit su SDI;
- arrivato al 16^o fronte di salita del clock, l'ultimo bit di dato viene letto ed inizia la fase di conversione;
- a questo punto la linea $!CS$ può essere lasciata al valore logico basso o alzata, l'importante è che torni alta almeno 20 ns prima della successiva conversione, in modo che un fronte di discesa di questo segnale la possa avviare nuovamente.

Ora che abbiamo considerato tutti i blocchi necessari, possiamo visualizzare (in figura 3.4) uno schema generale della parte di firmware che si occupa della corretta comunicazione tra il nostro filtro digitale e il convertitore da digitale ad analogico.

Unendo gli schemi in figura 3.2 e 3.4, includendo anche il blocco del filtro presentato nel capitolo precedente (figura 2.8), possiamo arrivare quindi alla struttura completa della logica implementata sulla nostra FPGA (figura 3.5).

Dopo aver scritto il codice VHDL per tutti i blocchi illustrati (che non derivino dal Core Generator), sono state effettuate delle simulazioni per verificarne la corretta implementazione. Perciò abbiamo simulato in primo luogo

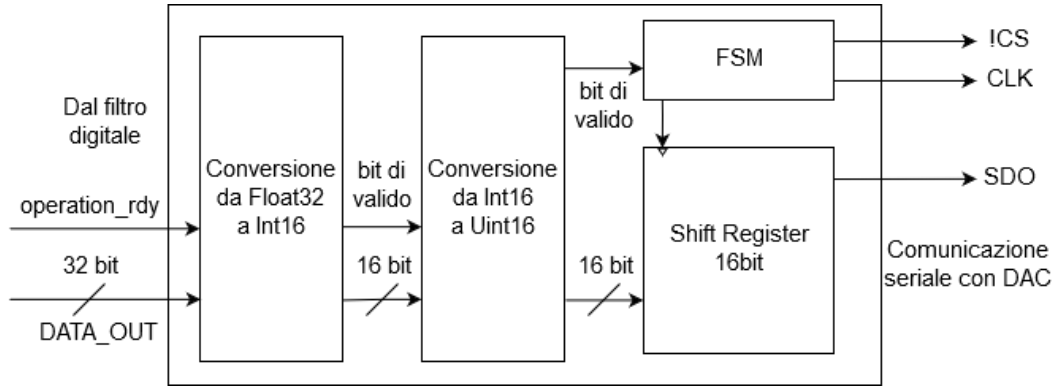


Figura 3.4: Schema completo (semplificato) del firmware implementato per connettere il filtro al DAC.

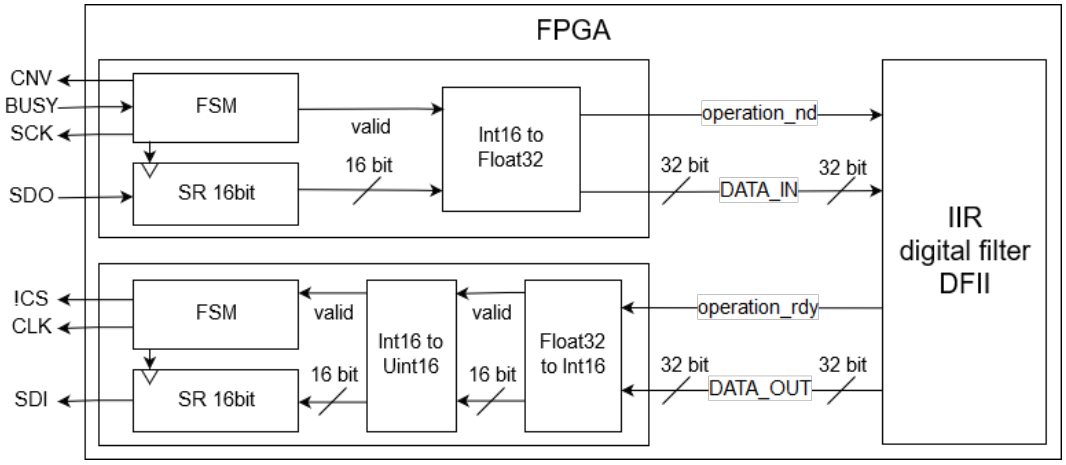


Figura 3.5: Schema completo del firmware implementato.

le interfacce separate (con segnali esterni forniti tramite testbench), per poi giungere alla verifica del sistema completo, ossia da una porta seriale all'altra. In questo modo abbiamo avuto la possibilità di controllare sia l'integrità e coerenza dei dati trasmessi, sia il rispetto dei timing alle interfacce di comunicazione.

3.2 Scheda elettronica

3.2.1 Condizionamento analogico prima dell'ADC

Focalizziamoci ora sul front-end analogico che precede l'ADC e che adatta il segnale alle sue specifiche mostrato in figura 3.6.

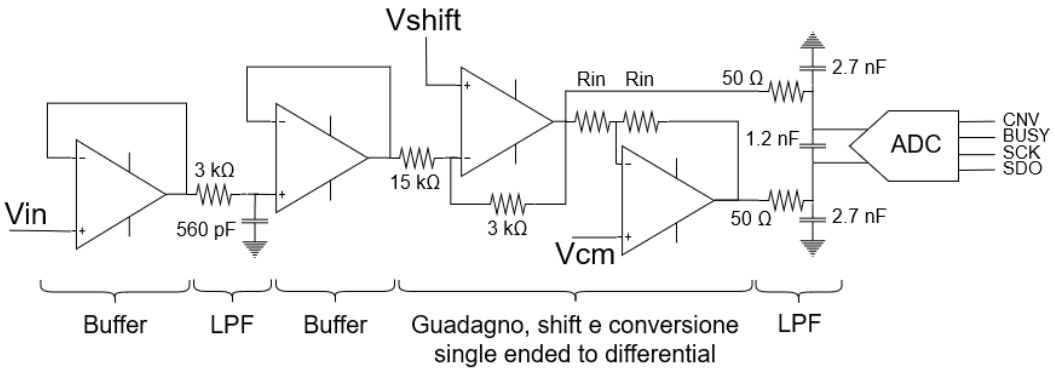


Figura 3.6: Stadio d'ingresso analogico dell'ADC, nel caso di ingresso single-ended e guadagno selezionato 0.2.

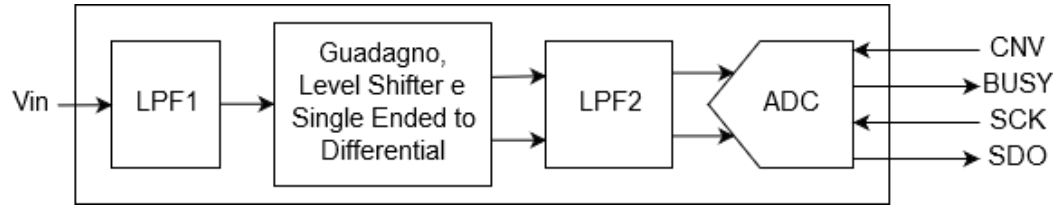
Sostanzialmente consiste in un filtraggio passa-basso anti-aliasing e in un level-shifter per centrare in dinamica il segnale da campionare. Ci siamo basati sulle architetture consigliate dal datasheet del convertitore [21] in quanto esse, con piccole modifiche, sono adatte ai nostri scopi e alle prestazioni necessarie a questa applicazione. Il primo stadio consiste in un buffer di ingresso, seguito da un filtro passivo passa-basso del primo ordine per eliminare eventuali disturbi ad alta frequenza. Utilizzando un altro buffer come disaccoppiamento si passa poi ad uno stadio di guadagno e level shift. Il guadagno è regolabile tramite dei jumper, consentendo così diversi range di tensioni in ingresso (vedere tabella 3.1); questo deve portare la tensione nel range 0.5-4.5 V dell'ingresso dell'ADC. La tensione verrà poi mappata, internamente all'ADC, su una dinamica di 0-5 V tramite Digital Gain Compression (DGC); questa funzione dell'ADC consente di limitare le non linearità che si creerebbero con segnali prossimi alle alimentazioni.
Tramite degli switch si può selezionare se in ingresso si ha un segnale single-ended o differenziale, visto che l'ADC ha ingresso differenziale dovremo in ogni

caso riportarci in questa condizione. Se venisse selezionata la modalità single-ended il blocco di guadagno si occuperebbe anche di creare una copia invertita del segnale, in modo che in ogni caso da questo blocco in poi si avrà un segnale differenziale indipendentemente dalla tipologia dell'ingresso.

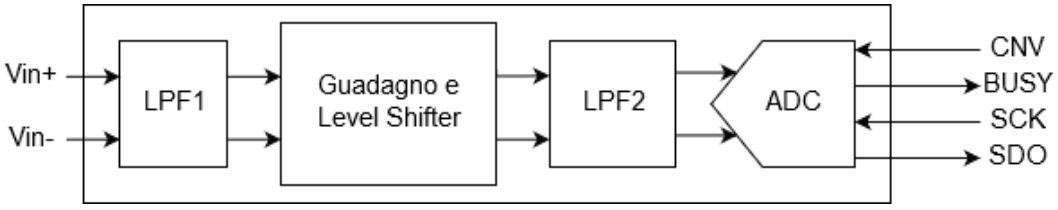
Selezione	**Guadagno**	**Dinamica di Vin**
1	0.2	$\pm 10\,\mathrm{V}$
2	0.4	$\pm 5\,\mathrm{V}$
3	0.8	$\pm 2.5\,\mathrm{V}$
4	1.6	$\pm 1.25\,\mathrm{V}$

Tabella 3.1: Valori di guadagno selezionabili per lo stadio d'ingresso e relativa dinamica di tensione in ingresso.

In seguito a questa fase si applica un secondo filtraggio passa-basso, utilizzando sempre un filtro passivo ma del secondo ordine, necessario ad evitare che segnali spuri a frequenze superiori causino dell'aliasing nel segnale campionato. Dopo questo ultimo filtraggio il segnale è pronto per essere mandato all'ingresso dell'ADC. Questa parte della scheda è schematizzata in figura 3.7 e 3.6.



(a) Per segnali single ended.



(b) Per segnali differenziali.

Figura 3.7: Schema funzionale dello stadio d'ingresso dell'ADC.

3.2.2 Da DAC a tensione di uscita

Anche per quanto riguarda il DAC ci siamo basati sull'architettura consigliata nel datasheet [22]. Dal datasheet vediamo che il componente può essere alimentato a due valori di tensione: 2.5 V oppure 5 V. Nel nostro caso è stato alimentato a 2.5 V in modo da avere maggior robustezza al rumore sull'interfaccia digitale rispetto a quando esso è alimentato al livello di tensione superiore; infatti la comunicazione con l'FPGA avviene secondo lo standard LVCMOS25, e quindi in caso si scegliesse l'alimentazione superiore si avrebbe una tolleranza sul livello alto di soli 100 mV (infatti in quel caso la soglia VIH dell'ADC è fissata a 2.4 V).

Come si può vedere in figura 3.8, il DAC è seguito da uno stadio a transimpedenza per trasformare la corrente in tensione. All'uscita di questo stadio il range disponibile è −5-0 V.
Per adattarlo alla nostra dinamica in uscita desiderata, che è ±5 V, utilizziamo in cascata uno stadio che si occupa di guadagnare un fattore 2 ed inserire un offset di 5 V.

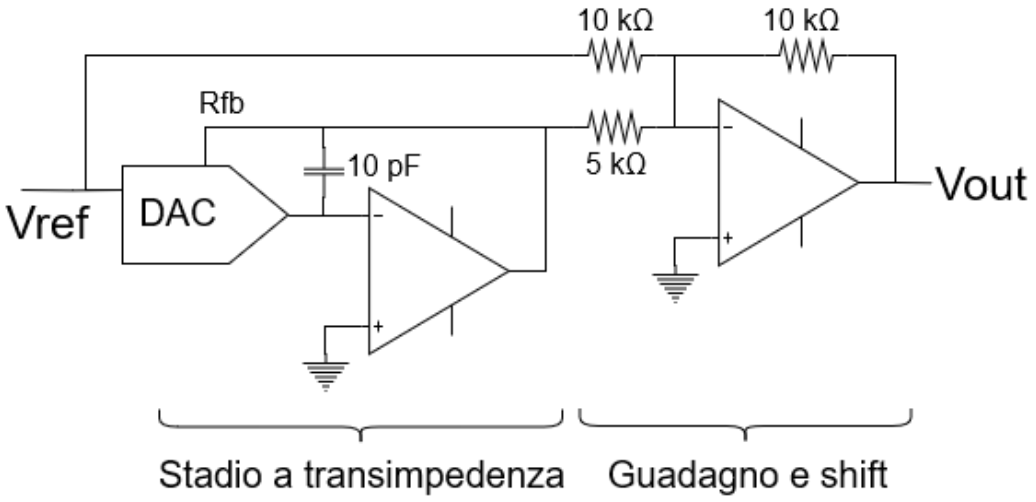


Figura 3.8: Stadio d'uscita.

La parte di circuito che è stata appena discussa è schematizzata in figura 3.9. Ora possiamo quindi unire i vari blocchi, che sono stati illustrati nella

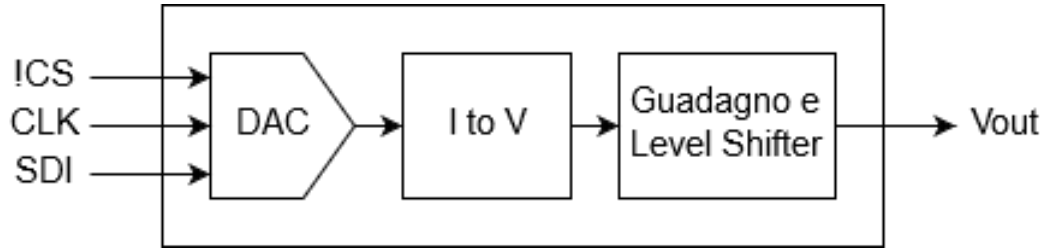


Figura 3.9: Schema a blocchi dello stadio di uscita analogico di MEMU.

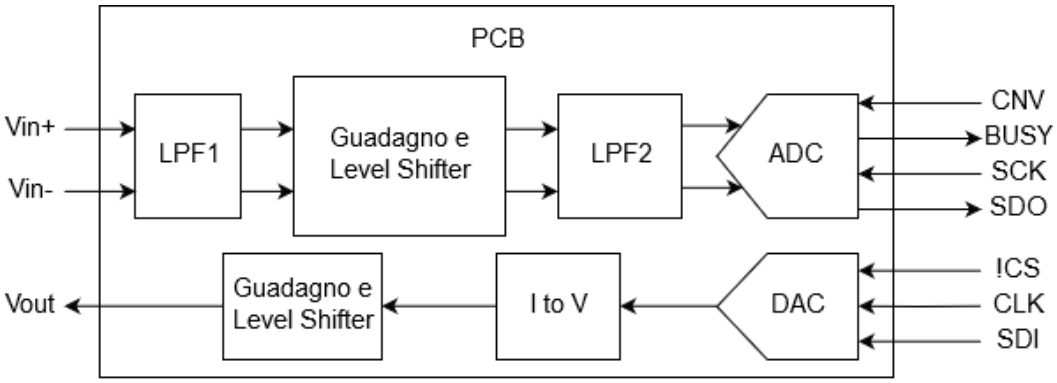


Figura 3.10: Schema completo della scheda elettronica.

sezione corrente, per giungere allo schema completo della scheda elettronica realizzata (figura 3.10).

Possiamo ora collegare quanto ottenuto con lo schema precedente della parte firmware interna all'FPGA (in figura 3.5), attraverso i segnali dell'interfaccia seriale, per visualizzare lo schema completo di MEMU ed avere più chiaro il funzionamento del sistema complessivo, mostrato in figura 3.12.

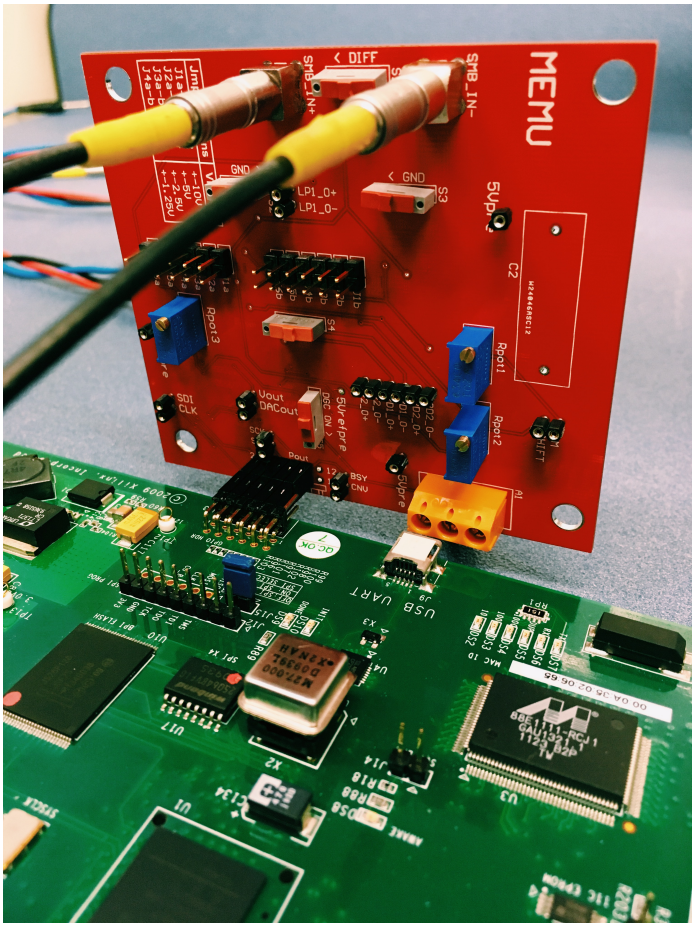


Figura 3.11: Foto della scheda elettronica connessa all'FPGA.

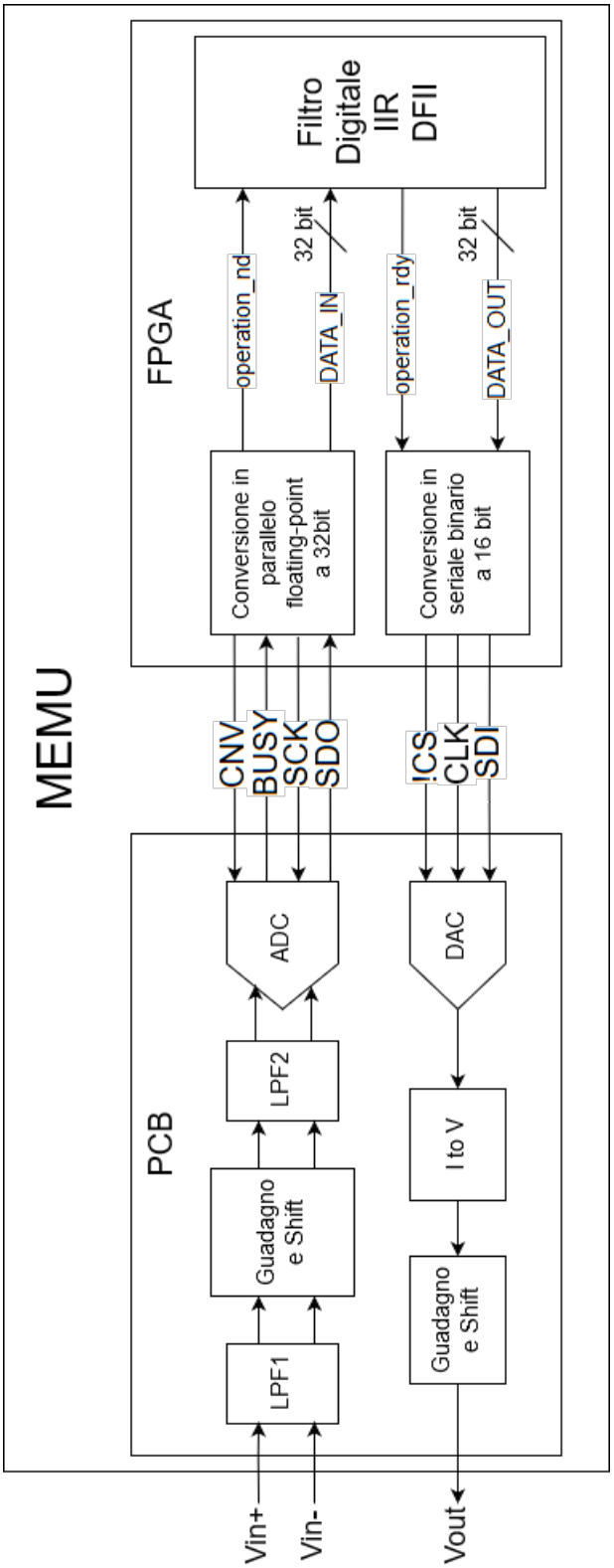


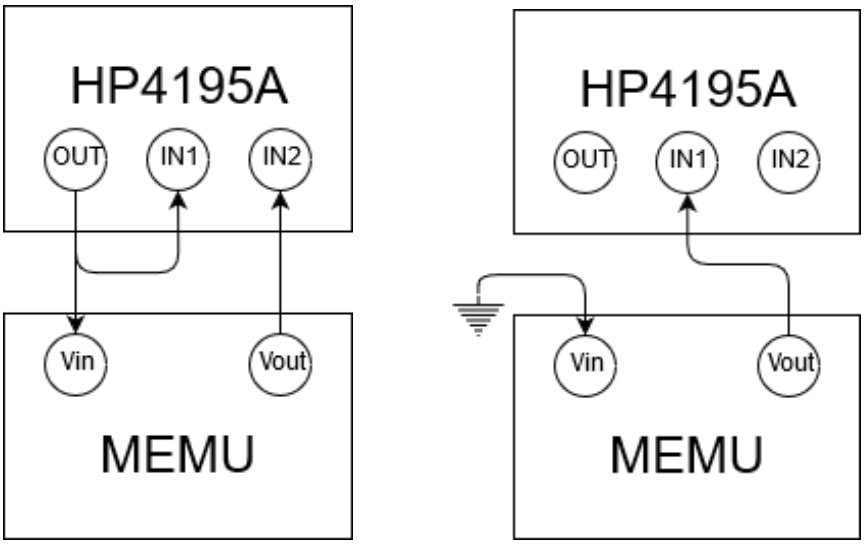
Figura 3.12: Struttura completa semplificata di MEMU.

Capitolo 4

Misure preliminari di validazione

4.1 Setup di misura

Dopo aver analizzato nei capitoli precedenti l'implementazione ed il funzionamento del sistema, possiamo passare ad analizzare i risultati delle misure effettuate sul sistema.
Le misure qui illustrate sono state effettuate, per semplicità e soprattutto per compatibilità con la strumentazione a disposizione, con il front-end di ingresso all'ADC in configurazione single-ended.
Lo strumento utilizzato per le misure è l'analizzatore di rete/spettro HP4195A; in figura 4.1 si possono vedere le due configurazioni utilizzate: la prima è stata utilizzata per l'analisi delle funzioni di trasferimento, mentre la seconda per valutare il trasferimento del rumore. I dati così raccolti sono stati poi trasferiti ad un computer tramite LabVIEW e analizzati tramite Matlab.
Si è iniziato dall'emulazione di dispositivi MEMS ideali (quindi il solo risonatore), per poi aggiungere successivamente anche altri effetti, quali ad esempio presenza di capacità parassite feedthrough e presenza di rumore termico-meccanico. In figura (4.2) sono state tracciate, in un unico grafico, diverse delle funzioni di trasferimento dei MEMS emulati. È così possibile notare l'alta flessibilità di MEMU; infatti vediamo fattori di qualità che vanno da meno di un'unità a 60 000, diverse frequenze di risonanza (da 3 kHz a 25 kHz) e costanti elastiche.



(a) Analizzatore di rete. (b) Analizzatore di spettro.

Figura 4.1: Schematizzazione dell'utilizzo dell'analizzatore di rete/spettro HP4195A usato per la caratterizzazione sperimentale di MEMU.

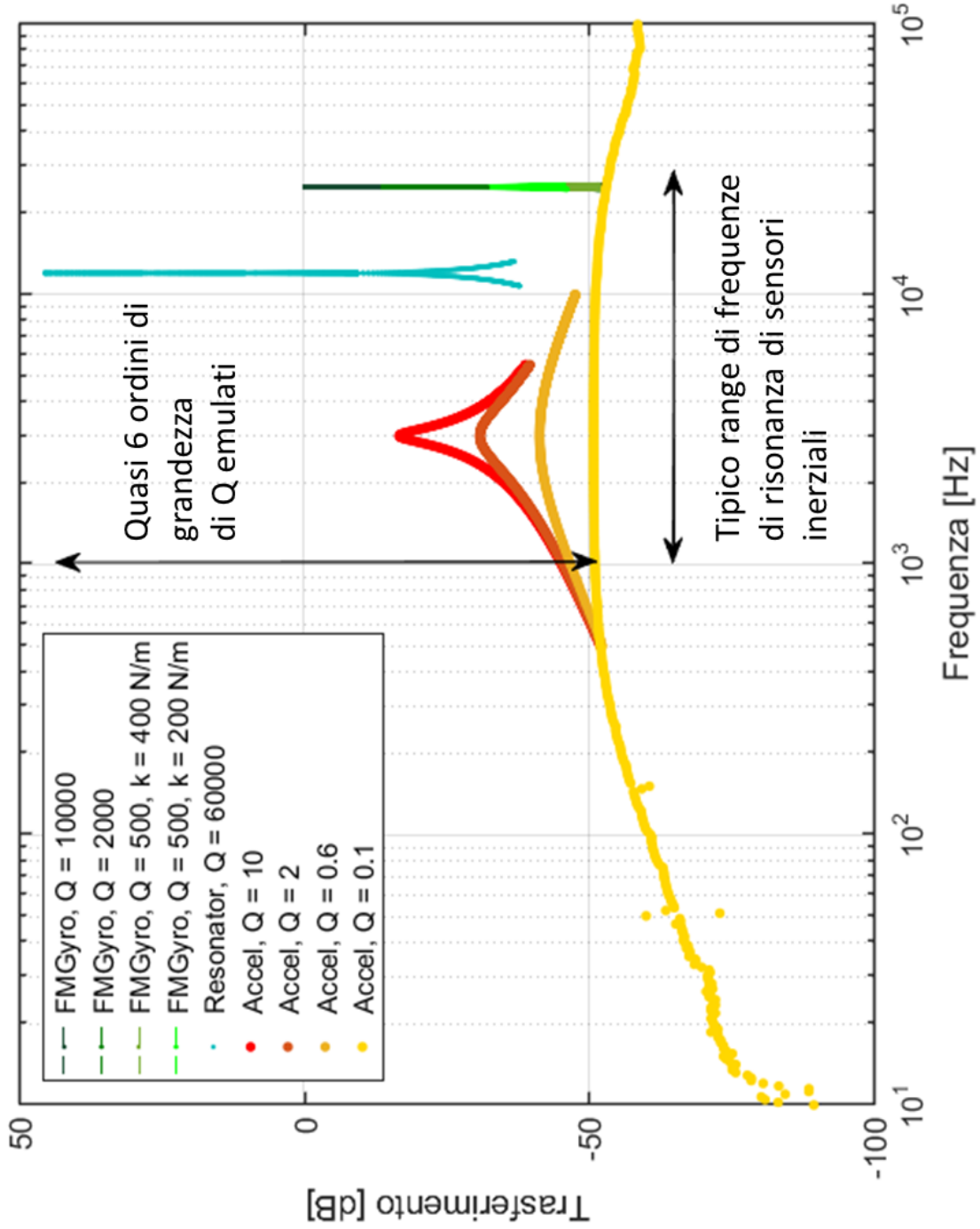


Figura 4.2: Trasferimento di vari dispositivi emulati con MEMU.

4.2 Emulazione di MEMS ideali

Iniziando dal caso più semplice, sono stati emulati dei dispositivi MEMS ideali. Lo scopo di questa serie di misure è di identificare eventuali limiti al range di frequenze e, soprattutto, di Q implementabili. Difatti, se le tipiche frequenze di risonanza di sensori inerziali variano in un range di circa 2 ordini di grandezza (da 1 kHz a 100 kHz), è sul fattore di qualità che si manifestano le variazioni più grandi su diversi tipi di sensori, con valori che possono spaziare da meno di un'unità a 10^6.
Le diverse misure sono state effettuate con un analizzatore di rete, riprogrammando di volta in volta MEMU. Si riporta sull'asse y il modulo della funzione di trasferimento, misurata in modalità "analizzatore di rete", in funzione delle frequenze di stimolo iniettate nell'ADC di MEMU.
Il primo dispositivo ad essere stato emulato è il giroscopio FM di riferimento. I risultati ottenuti (mostrati in figura 4.3) evidenziano come sia possibile osservare sul picco le variazioni del fattore di qualità Q (valori utilizzati: 10 000, 2000 e 500), e lontano da risonanza si può apprezzare la variazione della costante elastica k del sistema meccanico. I risultati sono perfettamente in accordo con il modello atteso.
Si è poi passati ad emulare degli accelerometri a diversi valori di Q (10, 2, 0.6 e 0.1) con frequenza di risonanza di 3 kHz ed una struttura risonante a 12 kHz con fattore di qualità elevato ($Q = 60\,000$). I risultati sono illustrati rispettivamente nelle figure 4.4 e 4.5.
Anche in questi due casi i risultati sono in accordo con quanto previsto.

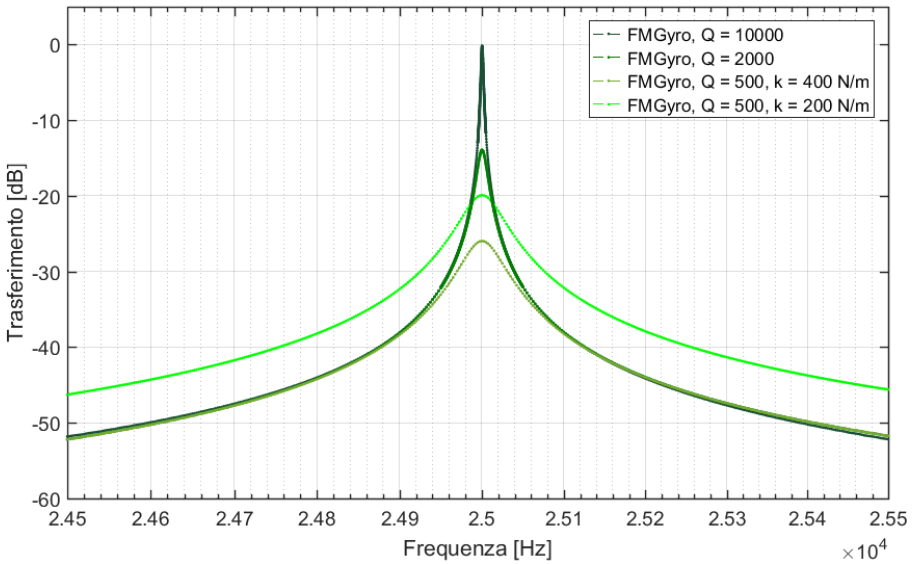


Figura 4.3: Funzioni di trasferimento dei giroscopi FM emulati. Si possono notare le variazioni dovute a diversi fattori di qualità Q e costanti elastiche k.

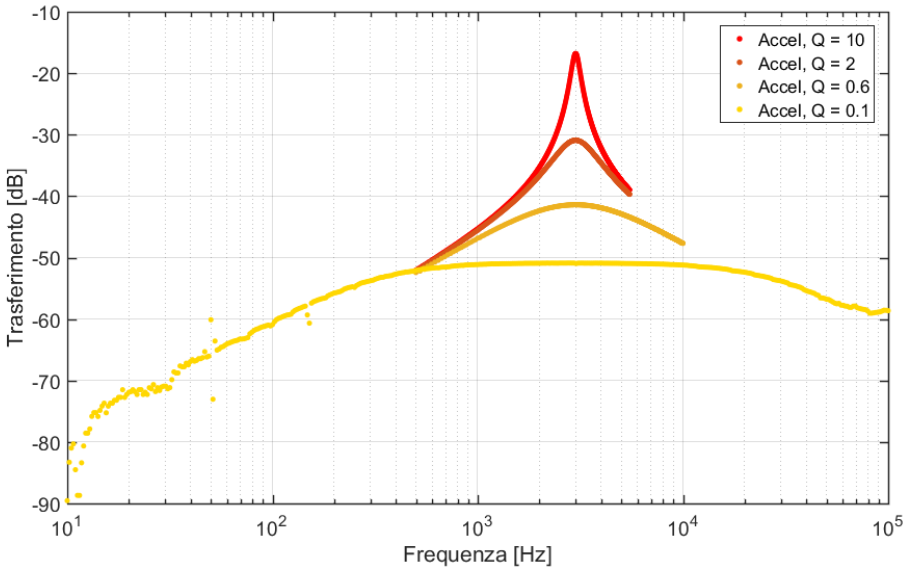


Figura 4.4: Variazioni della funzione di trasferimento di un accelerometro al variare del fattore di qualità Q.

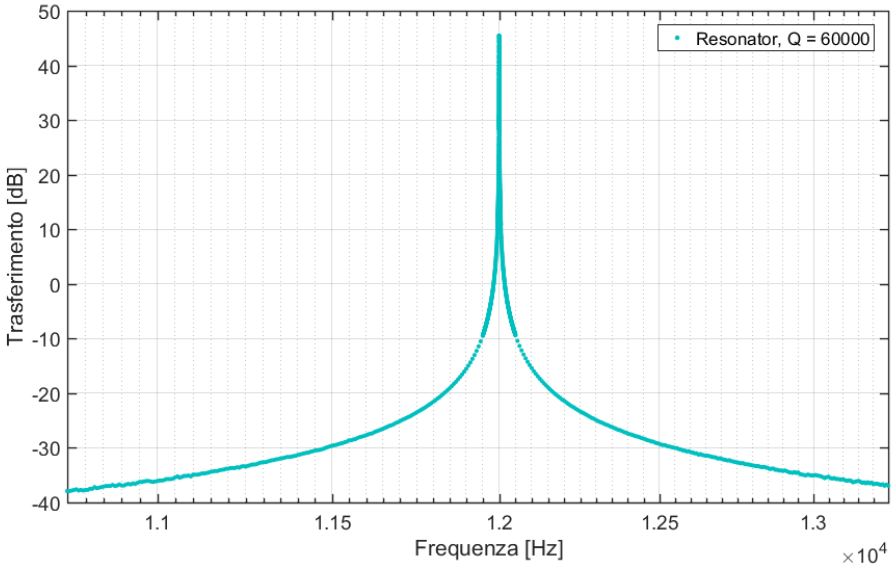


Figura 4.5: Funzione di trasferimento di un risonatore con un elevato fattore di qualità Q.

4.3 Emulazione di MEMS con capacità di feedthrough

Come abbiamo già visto, un parametro importante di un MEMS è la capacità di feedthrough (C_{ft}), ovvero l'accoppiamento parassita tra drive e sense che va a modificare il trasferimento del dispositivo. Dacché il dimensionamento di un circuito oscillatore (integrato o non) accoppiato a un risonatore può dipendere criticamente dal valore del feedthrough, può essere utile avere la possibilità di emulare un medesimo MEMS al variare della C_{ft}.
Per implementare questa capacità all'interno di MEMU possiamo applicare il principio della sovrapposizione degli effetti. Infatti, ricordando la figura 1.5, la corrente totale in uscita dal dispositivo è data dalla somma della corrente che scorre nell'RLC serie e nella capacità di feedthrough, la cui ammettenza è:

$$Y_{ft}(\omega) = j\omega C_{ft} \tag{4.1}$$

Quindi aggiungeremo in parallelo all'RLC un altro filtro, di struttura

identica, i cui coefficienti sono stati calcolati allo stesso modo ma partendo da:

$$T(s) = sC_{ft} \tag{4.2}$$

La struttura ottenuta è mostrata in figura 4.6.

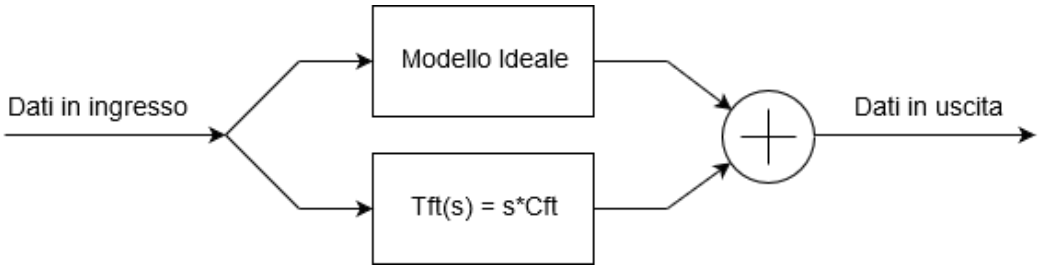


Figura 4.6: Schema semplificato del modello implementato per emulare la capacità di feedthrough C_{ft}.

Sommando le correnti in uscita, si riesce agevolmente a modellizzare questo effetto tipico dei MEMS. Possiamo vedere in figura 4.7 come la funzione di trasferimento venga modificata in base al valore della capacità di feedthrough. Per il nostro esempio abbiamo considerato capacità di 10 fF e 100 fF, valori ragionevoli che si possono incontrare in dispositivi reali; si ha la conferma di come capacità di feedthrough C_{ft} troppo elevate possano comportare un problema nella distinzione del picco di risonanza nella funzione di trasferimento.

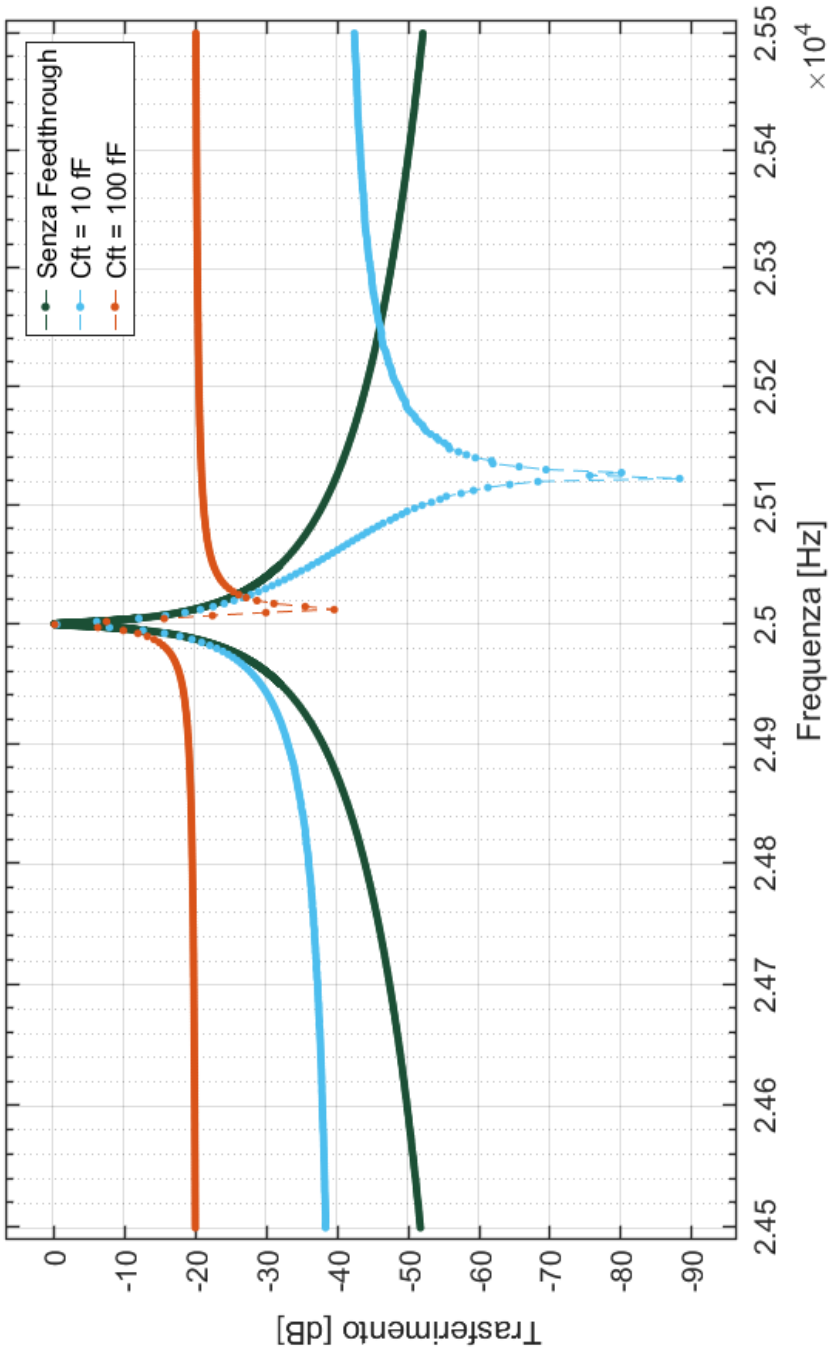


Figura 4.7: Verifica sperimentale dell'effetto di diversi valori della capacità di feedthrough C_{ft} sulla funzione di trasferimento.

4.4 Emulazione di MEMS includendo rumore termomeccanico

Il successivo elemento utile da emulare è il rumore termomeccanico; difatti, per taluni sistemi MEMS, tale contributo risulta essere il rumore dominante. In altri sistemi il rumore è addirittura un elemento fondamentale per il funzionamento del circuito (si pensi ad un oscillatore il cui start-up nasce proprio dal rumore).
Abbiamo già introdotto in precedenza (nel capitolo 2.5) la soluzione utilizzata per la sua generazione. In figura 4.8 è schematizzato il modello completo utilizzato per emulare il comportamento del MEMS nel caso si voglia includere questo fenomeno.

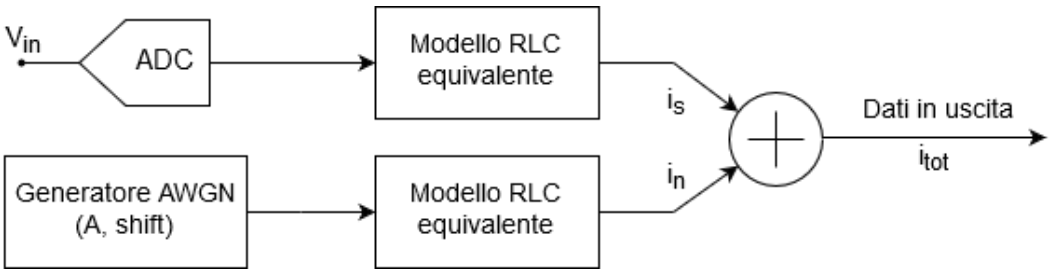


Figura 4.8: Schema semplificato del modello utilizzato per implementare il rumore termomeccanico.

Dove i_s è la corrente di uscita del modello ideale e i_n quella dovuta al rumore termomeccanico in ingresso. Quindi la corrente di uscita totale i_{tot} sarà:

$$i_{tot} = i_s + i_n \tag{4.3}$$

Per le misure sono stati utilizzati i seguenti MEMS emulati: (i) il giroscopio FM di riferimento (vedi dati in tabella 1.2) e (ii) un accelerometro con parametri elencati in tabella 4.1.

Per effettuare la misura del rumore si è posto l'ingresso a massa, in modo tale da annullare la "corrente di segnale" i_s isolare in uscita la componente dovuta al rumore i_n.

Parametro	Simbolo	Valore
Costante elastica	k	$3\,\mathrm{N/m}$
Frequenza di risonanza	f_0	$3\,\mathrm{kHz}$
Costante di trasduzione	η	$0.1 \times 10^{-6}\,\mathrm{V\,F/m}$
Tensione di bias	V_{dc}	$2\,\mathrm{V}$
Fattore di qualità	Q	2

Tabella 4.1: Valori numerici dell'accelerometro utilizzato nella misura di rumore.

Tramite l'analizzatore in modalità "spettro" possiamo misurare il profilo dello spettro del segnale di uscita: ci si aspetta che il rumore bianco segua la funzione di trasferimento del MEMS, essendo passato attraverso l'RLC serie. Questa configurazione può venire schematizzata nel modo rappresentato in figura 4.9.

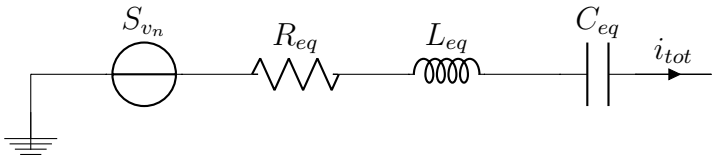


Figura 4.9: RLC serie ideale con generatore di rumore S_{v_n}.

Abbiamo visto in precedenza che il generatore equivalente di rumore termomeccanico d'ingresso ha un valore:

$$\sqrt{S_{v_n}} = \sqrt{4K_B T R_{eq}} \tag{4.4}$$

Basandoci su questa formula dimensioniamo la Power Spectral Density (PSD) del generatore di rumore bianco, ricavando da MATLAB il coefficiente di attenuazione A, introdotto nel capitolo 2.5. Perciò sul segnale di uscita i_{tot} ci attenderemo un valore di picco di:

$$S_{i_n}(\omega_0) = S_{v_n}/R_{eq} \tag{4.5}$$

Nel caso dell'accelerometro i risultati sono riportati in figura 4.10; possiamo vedere come il rumore, che all'ingresso era bianco, assuma all'uscita uno shaping dovuto al trasferimento del MEMS. Nella figura è stata anche tracciata la funzione di trasferimento del modello ideale del dispositivo.

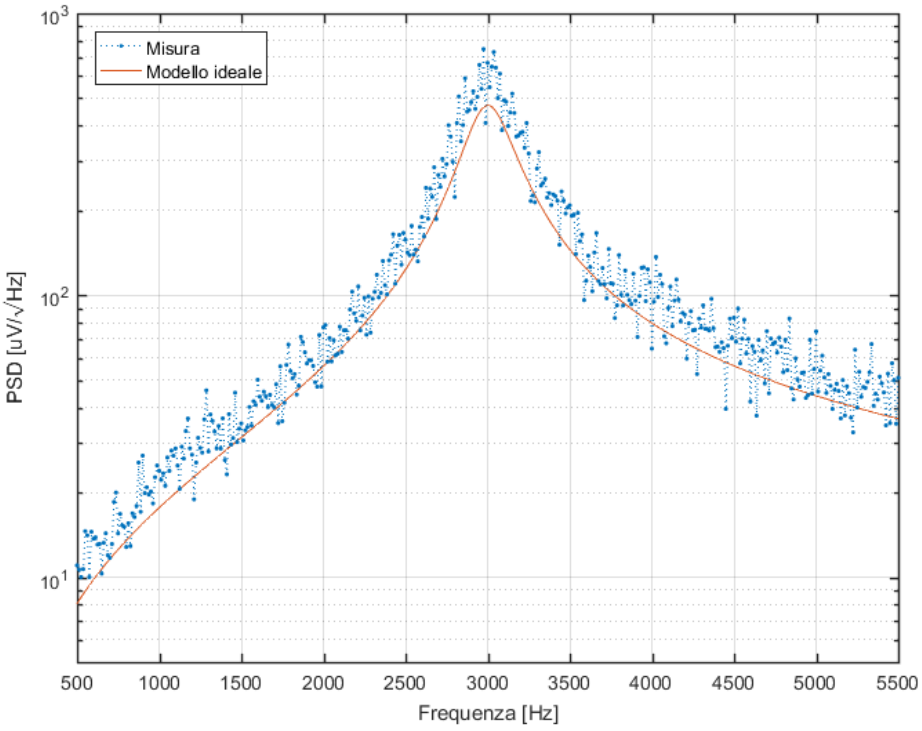


Figura 4.10: Spettro di rumore termomeccanico in uscita da MEMU configurato per emulare l'accelerometro descritto in tabella 4.1.

Per quanto riguarda il giroscopio FM considerato, se andiamo a calcolare la componente di corrente in uscita dovuta al rumore (possibile grazie ai valori forniti in tabella 1.2), è possibile notare una differenza di circa 6 ordini di grandezza tra il valore di corrente nominale e la componente dovuta al rumore. Questo fa sì che non sia possibile apprezzare questo fenomeno all'uscita a causa della limitata risoluzione del DAC a 16 bit, soprattutto fuori da risonanza dove vi è un'ulteriore attenuazione.

Capitolo 5

Conclusioni e Prospettive

5.1 Conclusioni

Durante questo lavoro di tesi è stato sviluppato, presso il Dipartimento di Elettronica, Informazione e Bioingegneria del Politecnico di Milano, un emulatore di dispositivi MEMS basato su FPGA.

La progettazione può essere sostanzialmente suddivisa in: (i) sviluppo di un firmware per FPGA in VHDL che implementi un filtro digitale e il protocollo di comunicazione e (ii) design una scheda per il condizionamento dei segnali in termini di ampiezza e la conversione da analogico a digitale e viceversa.

Come da specifiche iniziali si è riusciti ad emulare dispositivi MEMS con un ampio range di fattori di qualità Q (0.1-60 000), potendone variare i parametri e riscontrando una perfetta corrispondenza tra comportamento analogico e comportamento emulato.
Le frequenze a cui sono state effettuate le misure, ed in cui MEMU ha mostrato lo stesso comportamento del corrispettivo "analogico", vanno da 10 Hz a 100 kHz.

Come da obiettivo di partenza, è stato possibile tenere in considerazione nel modello anche alcune non idealità tipiche dei dispositivi MEMS, in particolare la capacità di feedthrough C_{ft} e il rumore termomeccanico della struttura.

La scheda elettronica realizzata consente la selezione di diversi range in

ingresso, in modo da non avere eccessiva perdita di dinamica dell'ADC in caso di MEMS che lavorino a tensioni molto diverse fra loro; questa costituisce il fattore limitante della realizzazione attuale, ad esempio la risoluzione in uscita del DAC è, a volte, non abbastanza raffinata per poter rappresentare fedelmente il rumore termomeccanico all'uscita del dispositivo.

Dato che il filtro, realizzato con prewarping sul picco, conserva in modo fedele il rapporto della fase a risonanza, con l'attuale uscita in tensione si dovrebbe poter chiudere l'anello con device compatibili, tipicamente MEMS con sense piezoresistivo, i quali hanno uscita in tensione.

5.2 Prospettive future

Test in anello chiuso

Il prossimo step per lo sviluppo di MEMU è rappresentato dal test del suo funzionamento in anello chiuso; come già detto la soluzione attuale con uscita in tensione non è compatibile con molti device, per consentire il funzionamento in anello chiuso sarebbe necessario progettare uno stadio di uscita in corrente, dato che la maggior parte dei dispositivi funziona in questo modo.

Come step intermedio verrà utilizzata l'emulazione di un MEMS a lettura piezoresistiva NEMS [23], che esce direttamente in tensione, accoppiando MEMU nella sua versione attuale ad una scheda elettronica già presente in laboratorio per tale tipo di sensore.

Aumento del range dei fattori di qualità emulabili

Vi sono pubblicazioni [6,7] che trattano di dispositivi MEMS con fattori di qualità Q estremamente elevati, nell'ordine di 1 milione. Riuscire, tramite MEMU, ad emulare valori di Q sempre più elevati sarebbe uno sviluppo certamente interessante ma soprattutto utile, ampliando, per esempio, le calibrazioni possibili in caso di utilizzo con strumentazione dedicata ai MEMS.

Ricostruzione dell'impedenza

In MEMU il nodo di drive viene connesso all'ADC, che ha un'impedenza d'ingresso elevata, ben diversa dall'impedenza d'ingresso di un MEMS reale.
Questa infatti è costituita da [24]: (i) RLC equivalente della sola parte di drive, (ii) capacità tra statore e massa e (iii) capacità di feedthrough.
Volendo estendere il funzionamento di MEMU sarebbe necessario ricostruire l'impedenza in ingresso Z_{in}. Questo verrebbe fatto con una struttura che effettui un "sink", dal nodo in ingresso, di una corrente del valore corretto. Un primo esempio di quanto detto può essere schematizzato con la struttura riportata in figura 5.1.

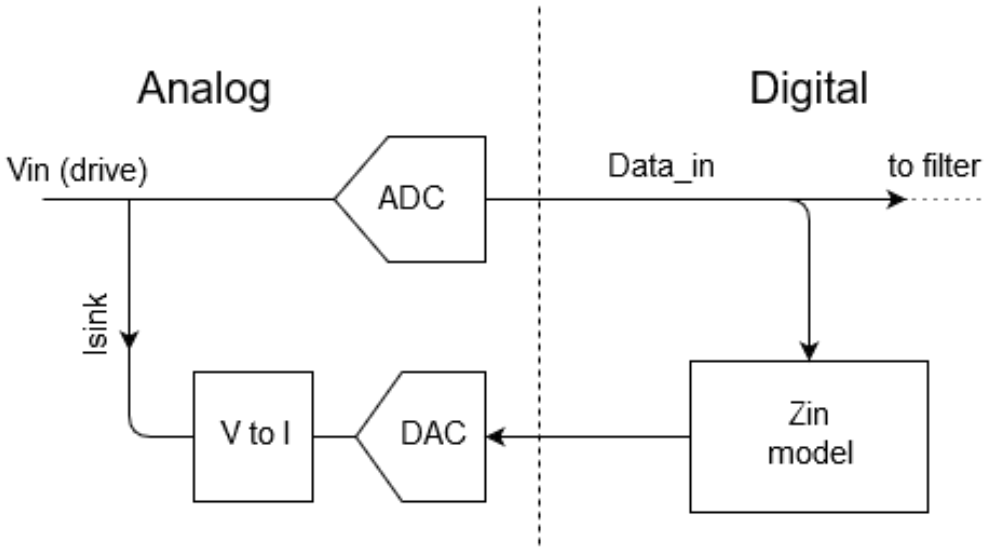


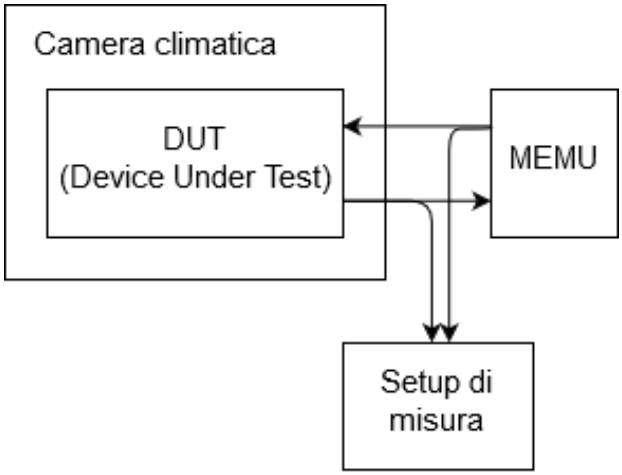
Figura 5.1: Sketch della struttura per la ricostruzione dell'impedenza in ingresso del MEMS emulato.

Drift dei parametri

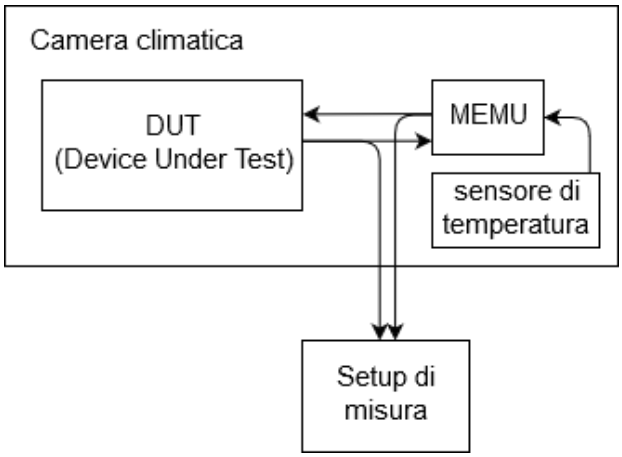
Un'altra feature utile sarebbe quella di poter configurare un drift (o anche un cambiamento repentino) dei parametri nel tempo.
Questo consentirebbe ad esempio di caratterizzare cambi nella struttura del device o in variabili esterne come la temperatura ambientale. In caso di aggiunta di nuove funzioni, come quella appena considerata, il fattore limitante è l'area; la problematica è infatti costituita dalla necessità di avere un interfaccia con il computer, in modo che sia possibile gestire il drift sul parametro e alla velocità desiderati.

Caratterizzazione in temperatura

Una tipica misura che offre dei dati utili è la caratterizzazione del sistema in temperatura. Se si volesse effettuare un test di questo tipo su di un ASIC in congiunzione all'utilizzo di MEMU, le soluzioni più semplici sarebbero due: (i) simulare il drift in temperatura come considerato in 5.2, (ii) aggiungere a MEMU un sensore di temperatura che verrebbe utilizzato real-time per il calcolo del modello aggiornato (figura 5.2).



(a) Emulazione della variazione di temperatura.



(b) Misura della variazione di temperatura.

Figura 5.2: Caratterizzazione in temperatura.

Bibliografia

[1] L. Gaffuri Pagani, G. Langfelder, P. Minotti, and N. Aresi, "A programmable emulator of MEMS inertial sensors," in *2017 IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, March 2017, pp. 142–143.

[2] ITmems, *MCP MEMS Characterization Platform*, www.itmems.it.

[3] ZurichInstruments, *HF2LI Lock-in Amplifier*, www.zhinst.com.

[4] Si-Ware, *SWS61111 Inertial Sensor Development Platform*, www.si-ware.com.

[5] G. Langfelder, M. Vandi, N. Aresi, and T. Frizzi, "Fast and reliable closed-loop wafer-level measurement of gyroscopes drive quality factor," in *Inertial Sensors and Systems, 2016 IEEE International Symposium on*. IEEE, 2016, pp. 119–120.

[6] A. A. Trusov, I. P. Prikhodko, S. A. Zotov, A. R. Schofield, and A. M. Shkel, "Ultra-high Q silicon gyroscopes with interchangeable rate and whole angle modes of operation," in *Sensors, 2010 IEEE.* IEEE, 2010, pp. 864–867.

[7] D. Senkal, M. J. Ahamed, M. H. A. Ardakani, S. Askari, and A. M. Shkel, "Demonstration of 1 Million Q-Factor on Microglassblown Wineglass Resonators With Out-of-Plane Electrostatic Transduction," *Journal of Microelectromechanical Systems*, vol. 24, no. 1, pp. 29–37, 2015.

[8] R. Abbiati, S. Scarpaci, A. Geraci, and G. Ripamonti, "Configurable Digital Emulation of Radiation Sources," in *Nuclear Science Symposium Conference Record, 2006. IEEE*, vol. 2. IEEE, 2006, pp. 959–963.

[9] A. Abba, F. Caponio, A. Cusimano, and C. Tintori, "Dual channel fast digital detector emulator with analog input," in *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2015 IEEE*. IEEE, 2015, pp. 1–3.

[10] G. Langfelder, "Lezioni del corso MEMS and Microsensors."

[11] A. Cenk and A. Shkel, *MEMS Vibratory Gyroscopes*. Springer, 2008.

[12] V. Kaajakari, *Practical Mems: Design of Microsystems, Accelerometers, Gyroscopes, RF Mems, Optical Mems, and Microfluidic Systems*. SMALL GEAR PUB, 2009.

[13] P. Minotti, G. Mussi, S. Dellea, A. Bonfanti, A. L. Lacaita, G. Langfelder, V. Zega, C. Comi, S. Facchinetti, and A. Tocchio, "A 160 μa, 8 mdps/$\sqrt{}$hz frequency-modulated mems yaw gyroscope," in *2017 IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, March 2017, pp. 1–4.

[14] R. W. S. Alan V. Oppenheim, *Discrete-Time Signal Processing*. ADDISON WESLEY PUB CO INC, 2009.

[15] K. J. Astrom and B. Wittenmark, *Computer Controlled Systems: Theory and Design (Prentice Hall Information and System Sciences Series)*. Prentice Hall, 1990.

[16] "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Std 754-1985*, 1985.

[17] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, Aug 2008.

[18] F. Ferrandi, "Digital System Design Methodologies Course."

[19] P. Alfke, "Efficient shift registers, LFSR counters, and long pseudo-random sequence generators," *http://www. xilinx. com/bvdocs/appnotes/xapp052. pdf*, 1998.

[20] *Encyclopaedia of Mathematics (Set).* SPRINGER VERLAG GMBH, 1994.

[21] LinearTechnology, *16-Bit, 1Msps, Low Power SAR ADC with 97dB SNR.* [Online]. Available: http://cds.linear.com/docs/en/datasheet/237816fa.pdf

[22] TexasInstruments, *DAC8811 16-Bit, Serial Input Multiplying Digital-to-Analog Converter.* [Online]. Available: http://www.ti.com/lit/ds/symlink/dac8811.pdf

[23] S. Dellea, G. S. Strazzeri, P. Rey, and G. Langfelder, "Ultra-low-voltage gyroscopes based on piezoresistive NEMS for drive-motion and coriolis-motion sensing," in *2017 IEEE 30th International Conference on Micro Electro Mechanical Systems (MEMS)*, Jan 2017, pp. 21–24.

[24] C. T. C. Nguyen and R. T. Howe, "An integrated CMOS micromechanical resonator high-Q oscillator," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 4, pp. 440–455, Apr 1999.