ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

SECONDA FACOLTÀ DI INGEGNERIA CON SEDE A CESENA
CORSO DI LAUREA IN INGEGNERIA ELETTRONICA INFORMATICA E DELLE TELECOMINCAZIONI

TITOLO DELL' ELABORATO

Progetto di un sistema a microcontrollore per il controllo di uno shaker elettrodinamico

Elaborato in

Elettronica dei sistemi digitali

Relatore
Prof. Ing. Aldo Romani

Correlatore
Dr. Matteo Filippi

Presentata da
Fabio Busignani

Sessione II

Anno Accademico: 2011/2012

Ai miei nipoti Giulio e Pietro

Indice

Introduzione **1**

1 Il microcontrollore **3**
1.1 Generalità . . . 3
1.2 Scelta del PIC . . . 3
1.3 Peripheral Pin Select (PPS) . . . 4
1.3.1 Controllo del PPS . . . 6
1.4 Interrupt . . . 7
1.4.1 Interrupt Vector Table . . . 8
1.4.2 I livelli di priorità . . . 8
1.4.3 Traps . . . 10
1.5 Timer . . . 11

2 Generazione della forma d'onda analogica **15**
2.1 Introduzione . . . 15
2.2 Teoria . . . 15
2.2.1 DAC con rete a scala . . . 17
2.2.2 Parametri caratteristici . . . 17
2.3 LTC1450 . . . 18
2.3.1 Funzione dei pin . . . 18
2.3.2 Operazioni . . . 20
2.4 Algoritmo di conversione . . . 21

3 Periferiche di comunicazione **23**
3.1 Introduzione . . . 23
3.2 SPI . . . 24
3.2.1 Teoria . . . 24
3.2.2 Interfaccia SPI nei PIC24F . . . 25

4 Lettura delle accelerazioni **29**
4.1 Principio di funzionamento . . . 29
4.2 KXP84 . . . 30
4.2.1 Interrupt . . . 32
4.2.2 La comunicazione SPI . . . 32
4.2.3 Registri di controllo . . . 33
4.3 Lettura dell'accelerazione . . . 34
4.3.1 Codice in C per la gestione dell'accelerometro . . . 34

4.3.2 Algoritmo per il calcolo dell'accelerazione efficace 37
4.3.3 Visualizzazione sul display LCD 39

5 Regolazione dell'ampiezza del segnale 41
5.1 Potenziometro digitale 41
5.2 MCP41010 43
5.3 Algoritmo di controllo 46

6 Lettura della tensione 49
6.1 Convertitore analogico - digitale 49
6.1.1 Convertitore ad approsimazioni sucessive 50
6.2 Modulo ADC interno al PIC 52

Schema Circuitale 55

Conclusioni 57

Elenco delle figure 58

Riferimenti bibliografici 61

Introduzione

Il problema energetico oggigiorno risulta essere sempre più complesso. Difatti nel processo evolutivo tecnologico c'è stato un netto squilibrio tra lo sviluppo energetico e lo sviluppo dell'elettronica e delle telecomunicazioni.
Il mondo della ricerca si sta concentrando non solo su come diminuire i consumi dei dispositivi elettronici ma anche su come ricavare, ed immagazzinare, energia da fonti alternative o non convenzonali.
Col termine **Energy Harvesting** vengono indicati i sistemi elettronici in grado di immagazinare energia elettrica direttamente impiegabile dall'ambiente circostante, avvalendosi di queste fonti alternative che possono essere così suddivise:

- fonti termiche;
- fonti piezoelettriche (sonore, di movimento);
- fonti fotovoltaiche;
- fonti da onde elettromagnetiche.

Nell'ultimo decennio il campo dell'energy havesting ha avuto un importante incremento come si evince dal crescente numero di pubblicazioni scientifiche e di prototipi. Questo argomento, seppur ancora in fase di sviluppo, ricopre già un'ampia area di impiego che va dal settore commerciale a quello militare.
Il progetto esposto in questa tesi non rappresenta uno studio sulle nuove fonti energetiche, bensì si vuole porre al servizio della suddetta ricerca. Esso viene infatti impiegato per la caratterizzazione dei sensori piezoelettrici. Sensori che verranno successivamente utilizzati per la realizzazione di dispositivi energy harvesting da sorgenti piezoelettriche, ossia dispositivi che sfruttano le vibrazioni a bassa frequenza per la realizzazione di energia elettrica.
Nonostante questi device eroghino tensioni e correnti molto basse, trovano comunque un notevole impiego per l'alimentazione di sensori wireless a basso consumo di potenza (*Low-power device*) che sono quindi in grado di fornire le informazioni utili senza la necessità di utilizzare dei cavi o delle batterie, svincolando così il progettista da eventuali problemi sia di natura logistica che economica.
Per lo studio del comportamento del sensore piezoelettrico ci si avvale di uno shaker elettrodinamico che dovrà trasmettere una vibrazione al piezoelettrico stesso, avente un'accelerazione efficace nota e personalizzabile dall'utente. Il controllo di questa grandezza viene dunque affidato al sistema a microcontrollore descritto di seguito e avente lo schema a blocchi di (Fig. 1).
In tale schema si nota come il controllo è suddiviso sostanzialmente in due processi,

uno mirato all'acquisizione delle accerelazioni ed al calcolo dell'accelerazione efficace, mentre l'altro, si occupa di regolare l'ampiezza della forma d'onda fornita in ingresso allo shaker.
Il progetto deve andare a sostituire un sistema già esistente dove il processo di acquisizione delle accelerazioni e calcolo dell'accelerazione efficace viene affidato al personal computer grazie all'ausilio del software *Labview* (sviluppato dalla *National Istrument*), mentre il processo di regolazione d'ampiezza dell'onda, e dunque il controllo sull'accelerazione, viene affidato affidato all'utente stesso mediante l'utilizzo di un trimmer multigiri. Questo sistema però, com'è facilmente intuibile, non è né molto efficiente, né tantomeno *user friendly*. Infatti, l'utilizzatore deve stare molto attento a come effettuare la regolazione in quanto un aumento spropositato dell'ampiezza del segnale comporta lo stesso aumento in termini di accelerazione e, vista la fragilità dei sensori piezoelettrici, la loro conseguente rottura.
La necessità di avere un sistema a microcontrollore è data dal fatto che viene richiesto al circuito la possibilità di lavorare *stand-alone*, ossia senza l'ausilio di componenti software/hardware aggiuntivi.
Lo scopo finale è quello di realizzare un banco di prova che sia in grado di effettuare **misure ripetibili**. Solo in questo modo è infatti possibile definire sperimentalmente il comportamento di un componente o, generalizzando, di un processo.

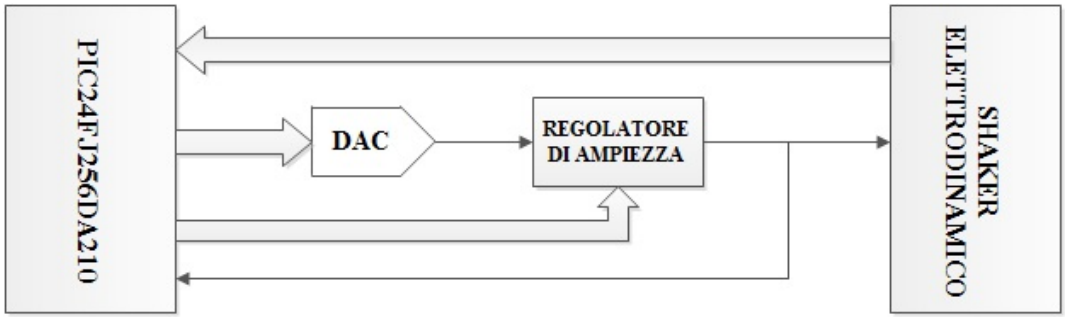


Figura 1: Schema a blocchi del progetto

Il sistema prende in ingresso delle forme d'onda memorizzate in una Flash esterna e un valore di accelerazione efficace stabilito dall'utilizzatore e provvede a far oscillare lo shaker elettrodinamico con queste specifiche.
Affinché ciò avvenga ci si avvale dapprima di un convertitore digitale - analogico (vedi capitolo 2) avente il compito di interpolare i punti descritti dai campioni digitali memorizzati e forniti dal microcontrollore al suo ingresso, così da ottenere una forma d'onda continua sia nel tempo che nei valori: una forma d'onda analogica, appunto.
L'onda, prima di arrivare all'ingresso dello shaker, passa attraverso il blocco denominato *regolatore di ampiezza* che, come verrà illustrato nel capitolo 5, ha il compito di modificare l'inviluppo dell'onda col fine di controllare l'accelerazione efficace prodotta dallo shaker. Il controllo sul blocco di regolazione è affidato al microcontrollore. Esso prende le sue decisioni sulla base dell'accerelazione efficace imposta dall'utente e quella calcolata con l'ausilio di un accelerometro solidale alla vibrazione imposta ai sensori (vedi capitolo 4).
Verranno ora descritti i singoli blocchi mostrati in (Fig. 1) e, per ognuno di essi, si analizzeranno le caratteristiche che hanno determinato le scelte progetturali.

Capitolo 1

Il microcontrollore

1.1 Generalità

Un microcontrollore, abbreviato **MCU** (*Micro - Controller Unit*), è un sistema programmabile che include in un unico package un microprocessore, memorie, periferiche digitali ed analogiche, oscillatori atti alla generazione dell'onda di clock ed altri componenti ancora che possono variare in base al tipo di microcontrollore scelto.
Da questa breve descrizione si evince che per potersi interfacciare col mondo circostante, i microcontrollori, necessitano di pochi componenti e, quindi, garantiscono una bassa complessità hardware. Per questo motivo, spesso, questi dispositivi vengono definiti come dei *mini - computer* veri e propri.
Esistono diverse aziende che producono microcontrollori ma, per questo progetto è stato scelto di affidarsi ai **PICmicro** (spesso abbreviati al semplice *PIC*) della *Microchip*. Questa scelta è dovuta alla notevole quantità di PIC messi a disposizione dall'azienda.
In base alla dimensione dei registri interni i PIC si dividono in tre grandi famiglie, 8, 16 e 32 bit. Ognuna di esse, oltre a distinguersi per la lunghezza dei registri, si distingue anche per la quantità di memoria, per i tipi di periferiche integrate, per la frequenza di clock alla quale può operare e per i consumi di potenza.

1.2 Scelta del PIC

In questo progetto le specifiche impongono che, nel caso in cui la forma d'onda non sia una sinusoide, ma una forma qualsiasi (*custom*), per il calcolo dell'accerelazione efficace occorra osservare l'andamento della grandezza per circa 10 secondi. Questo, da un punto di vista di tipo computazionale richiede una quantità di RAM non reperibile nei PIC ad 8 bit e, quindi, si è deciso di far affidamento alla famiglia a 16 bit.
Il microcontrollore impiegato è il PIC24FJ256DA210, nella sua *development board* distribuita dalla Microchip stessa ed illustrata in (Fig. 1.1).

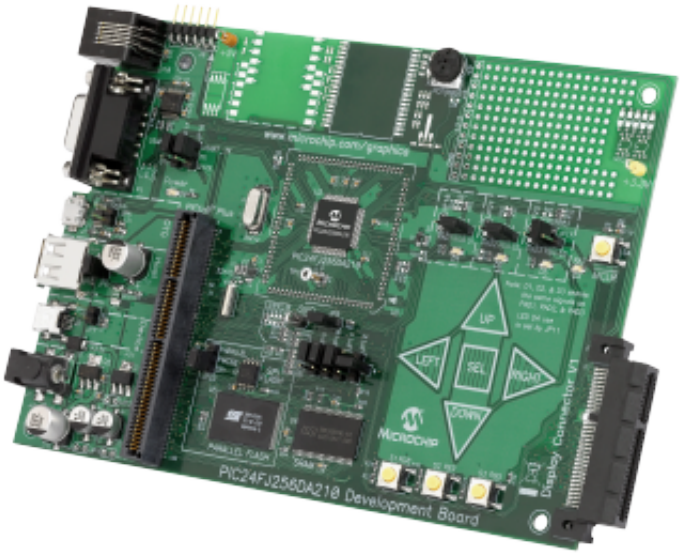


Figura 1.1: PIC24FJ256DA210 Development Board

La scheda, oltre al microcontrollore, presenta una serie di periferiche utili soprattuto per il progetto sviluppato parallelamente sulla quale si basa il sistema descritto in questa tesi come:

- i pulsanti;
- una memoria FLASH da $2\ MByte$;
- una periferica USB;
- *LDO* (Low - Dropout) a 3.3 e $5\ V$.

Le varie caratteristiche del PIC24FJ256DA210 saranno discusse man mano nel corso dell'elaborato, qui mi limiterò ad illustrare delle particolarità circa la gestione delle periferiche, degli interrupt e del timer che sono caratteristiche della famiglia di microcontrollori a 16 bit PIC24F.

1.3 Peripheral Pin Select (PPS)

A differenza di quanto accade nella famiglia dei PICmicro a 8 bit, quelli a 16 presentano un numero superiore di periferiche e queste non sono fisse. Facendo riferimento al diagramma dei pin illustrato in (Fig. 1.2) si può infatti notare come, ad esempio, manchino nelle funzionalità dei pin stessi le sigle che caratterizzano una comunicazione SPI. Ciò è duvuto al fatto che questa periferica rientra tra le numerose periferiche programmabili.
Il *Peripheral Pin Select* opera su un sottoinsieme fisso di pin digitali di ingresso/uscita. Grazie ad esso, il programmatore può mappare in maniera del tutto indipendente l'ingresso e/o l'uscita della maggior parte (non tutte) delle periferiche digitali presenti all'interno del microcontrollore.

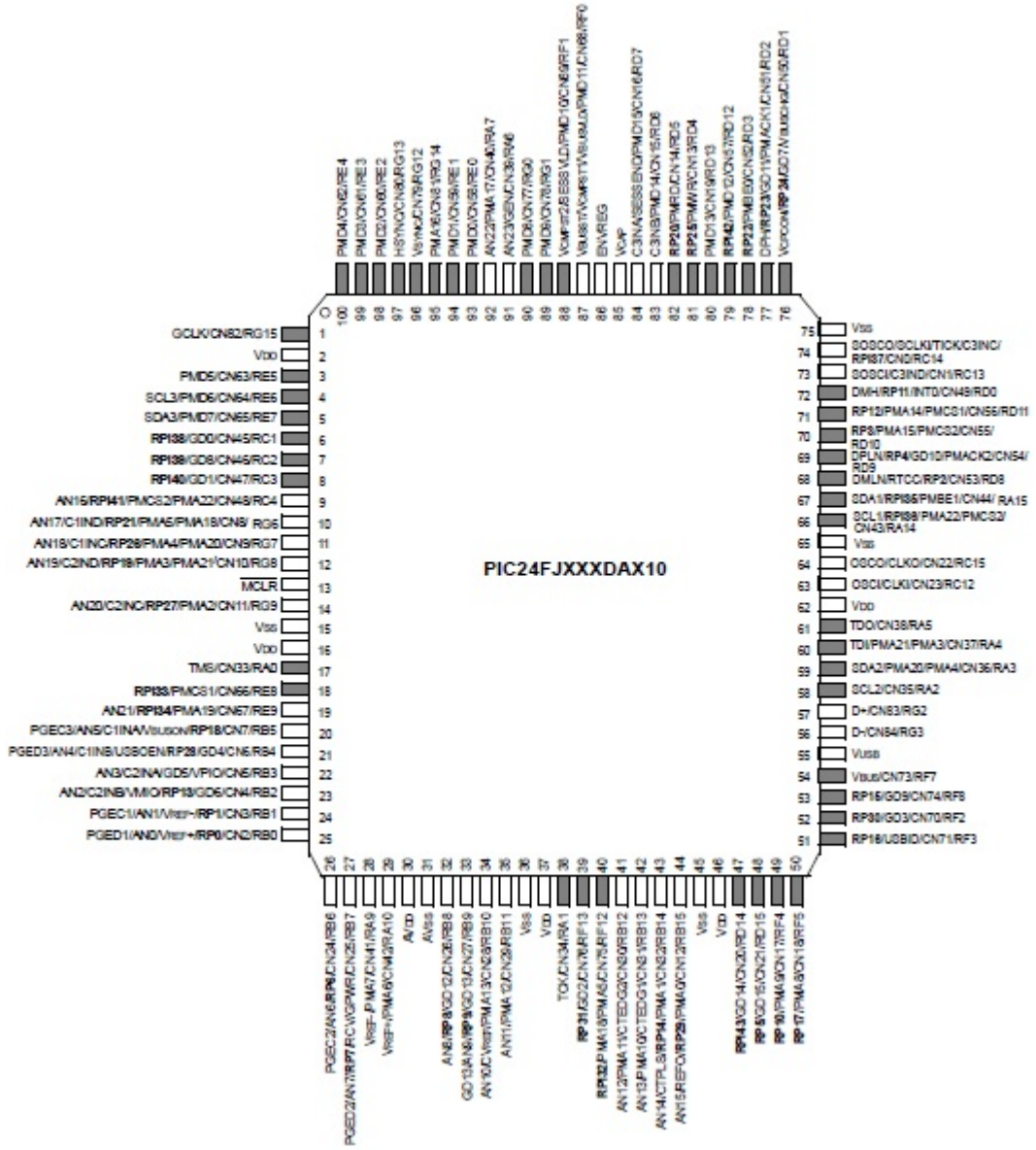


Figura 1.2: PIC24FJ256DA210, diagramma dei pin

Il sottoinsieme di pin sul quale è possibile mappare le varie periferiche digitali è riconoscibile in (Fig. 1.2) grazie alla denominazione **RPn** (dove con *n* si indica il numero del pin considerato). Alcuni pin non possono essere mappati come uscite, supportando dunque la sola funzione di ingresso e per essere distinti vengono indicati con **RPIn**.
Le periferiche gestite attraverso il PPS sono tutte solo di tipo digitale, quelle analogiche hanno dunque sempre un'assegnazione dei pin fissa. Tra queste periferiche sono presenti quelle atte alla comunicazione seriale (come la UART e SPI), gli interrupt esterni, gli ingressi ai quali viene fornito il segnale di clock dei timer e le altre periferiche relative ai timer.
Come si può notare dalla lista, non tutte le periferiche di tipo digitale vengono gestite dal *Peripheral Pin Select*, quelle escluse infatti non possono godere di questa rimappatura perchè richiedono una circuiteria specifica, tra quelle ad assegnazione fissa ritroviamo ad

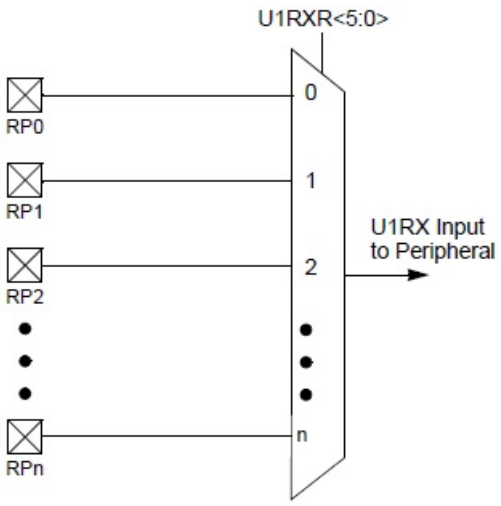


Figura 1.3: Peripheral Pin Select in ingresso, esempio effettuato su U1RX

esempio I^2C e USB.

1.3.1 Controllo del PPS

Il controllo del *Peripheral Pin Select* avviene attraverso due *Special Function Register*, uno si occupa dell'associazione della periferica nel caso in cui la funzione coinvolta sia di input, l'altra, invece, nel caso in cui sia una funzione di output. Il motivo per cui si utilizzano due registri diversi risiede nel fatto che, in questo modo, quando si considera una particolare periferica che possiede un pin che è sia un ingresso che un'uscita, allora questo può essere mappato su entrambe i registri senza particolari vincoli.
A livello logico vi è anche una sostanziale differenza tra il *mapping* di ingresso e quello d'uscita. Nel primo caso è la periferica che viene assegnata al pin, vedi (Fig. 1.3). Nel secondo caso è il pin che viene associato alla periferica, vedi (Fig. 1.4). Il registro utilizzato per mappare i pin di ingresso viene denominato con **RPINRx** (dove x è un numero) all'interno del quale ogni periferica mappabile viene rappresentata attraverso 6 bit. Coerentemente a quanto appena affermato, la programmazione assegna alla periferica interessata il pin *RPn* dove n è il valore decimale espresso attraverso i 6 bit di campo descritti pocanzi.
Contrariamente agli ingressi, per quanto riguarda le uscite il *Peripheral Pin Select* opera sulla base del pin. In questo caso, il registro utilizzato è il **RPORx**, entro il quale vengono dedicati 6 bit ai pin che possono essere soggetti all'operazione di mappatura. Il valore dei 6 bit descrive in manirera univoca una periferica d'uscita da mappare al pin
Questo mapping in uscita prevede anche il valore nullo (formato da tutti zeri), in questo modo il pin sarà un ingresso o un'uscita digitale (la sua natura viene dettata dallo stato del bit corrispondente nei registri TRIS). Seppur la configurazione di valore nullo rappresenti il *default state* di ogni pin, può capitare in realtà che questo non sia verificato e, se non impostato, comporta un malfunzionamento del pin stesso. Durante le prove del circuito presentato in questa tesi io stesso mi sono dovuto scontrare con questo fenomeno risolvibile ricordandosi di comprendere nel *Peripheral Pin Select* tutti i pin digitali utilizzati. Da quanto detto, si può notare come le due operazioni siano sostanzialmente

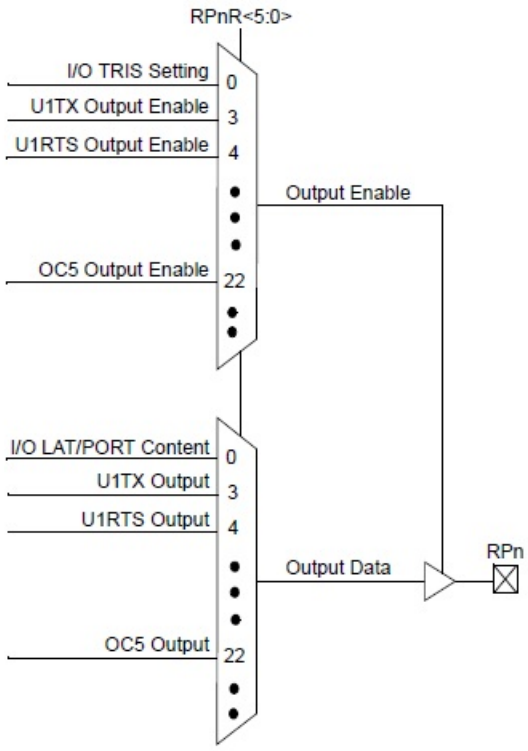


Figura 1.4: Peripheral Pin Select in uscita, esempio effettuato su OC5

una il duale dell'altra.

1.4 Interrupt

Un *interrupt* (interruzione se lo volessimo tradurre in italiano) è un evento che interrompe, appunto, la normale esecuzione del programma per servire una porzione di codice differente, chiamata *Interrupt Service Routine*, **ISR**. Dopo che la *ISR* è stata completata il programma ritorna al punto dov'era prima che l'*interrupt* sopprangiungesse.
Si può pensare all'*ISR* come ad un sottoprogramma (*subrutine*) che non viene invocato a livello software tramite il comando *assembly* **GoSub**, ma piuttosto da un evento. Quest'evento può essere esterno, come ad esempio quello di un pin d'ingresso che modifica il proprio livello logico, oppure può sorgere internamente al microcontrollore, come accade per gli interrupt generati da un timer che raggiunge il suo stato di fine conteggio e setta dunque il bit di *overflow*.
La tecnica ad interrupt contrasta quella a *polling*, quest'ultima prevede che, per accertarsi che un evento avvenga, il microcontrollore interroghi costantemente il *flag* o il pin corrispondente e poi agisca di conseguenza. Come è facile osservare gestire gli eventi a polling risulta essere maggiormente dispendioso.

1.4.1 Interrupt Vector Table

Nei PIC24F ad ogni sorgente di interrupt configurabile corrisponde un proprio *interrupt vector*. Tutti questi vettori vengono contenuti all'interno di una tabella chiamata *Interrupt Vector Table* (**IVT**) che risiede nella memoria di programma a partire dalla locazione $0x000004$. La tabella contiene ben 126 vettori. Ci sono 126 sorgenti di interrupt differenti tra cui 8 non sono mascherabili (ossia sono sempre attivi) e vengono denominati *trap interrupts*[1].
Il valore programmato all'interno di ogni *interrupt vector* rappresenta l'indirizzo iniziale della *ISR* ad esso associata.
In base alla locazione occupata in memoria, ad ogni interrupt è associato un numero identificativo indicato come **IRQ** (*Interrupt Request number*).
Il numero *IRQ* oltre ad identificare in maniera univoca un *interrupt vector*, consente di fornire anche una priorità naturale alla richiesta di interrupt. Infatti, se due interrupt hanno lo stesso livello di priorità, quello avente il numero *IRQ* più piccolo (e quindi quello all'indirizzo della locazione di memoria più basso) ha una maggior *priorità naturale*. Per maggiori delucidazioni in merito è bene fare riferimento alla (Fig. 1.5).

Alernate Interrupt Vector Table

Oltre alla tabella di interrupt descritta pocanzi, è presente anche una tabella alternativa. Questa tabella è del tutto identica alla precedente e viene utilizzata in casi particolari. Per abilitarla occorre agire sul bit *ALTIVT* del registro *INTCON2*. Se tale bit viene settato, gli interrupt non saranno più serviti in base ai valori contenuti nella tabella *IVT*, bensì il microcontrollore si riferirà alla sola tabella alternativa, *AIVT*.

1.4.2 I livelli di priorità

I PIC appartenenti alla famiglia dei 16 bit consentono al programmatore di poter impostare gli interrupt utilizzati in ben sette diversi livelli di priorità. In questo modo, se si verificassero contemporaneamente due interrupt aventi due diversi livelli di priorità, il microcontrollore gestirebbe questo conflitto servendo per prima l'interrupt a priorità più alta.
Di default i PIC24F operano sfruttando l'**annidamento di interrupt** (*Nesting*), così se mentre è in esecuzione una *ISR* associata ad un interrupt avente un determinato livello di priorità, si verifica un interrupt ad un livello di priorità maggiore, allora l'esecuzione del primo interrupt viene interrotta per consentire lo svolgimento del secondo interrupt. Questo comportamento può essere disabilitato ponendo a livello logico alto il bit *NSTDIS* contenuto nel registro *INTCON1*.
Disabilitando il nesting gli interrupt vengono serviti in sequenza e, il livello di priorità, è determinante solo nel caso in cui due richieste di interrupt avvengano contemporaneamente.
La priorità da associare ad un determinato interrupt avviene per mezzo di tre bit contenuti nei registri **IPCn** (*Interrupt Priority Control*). I livelli di priorità configurabili partono dal livello 1 che rappresenta la priorità più bassa e finiscono al livello 7 che rappresenta, invece, la più alta. Se i tre bit sono tutti a livello logico basso, ovvero si è ad un livello di

[1] vedi sezione 1.4.3

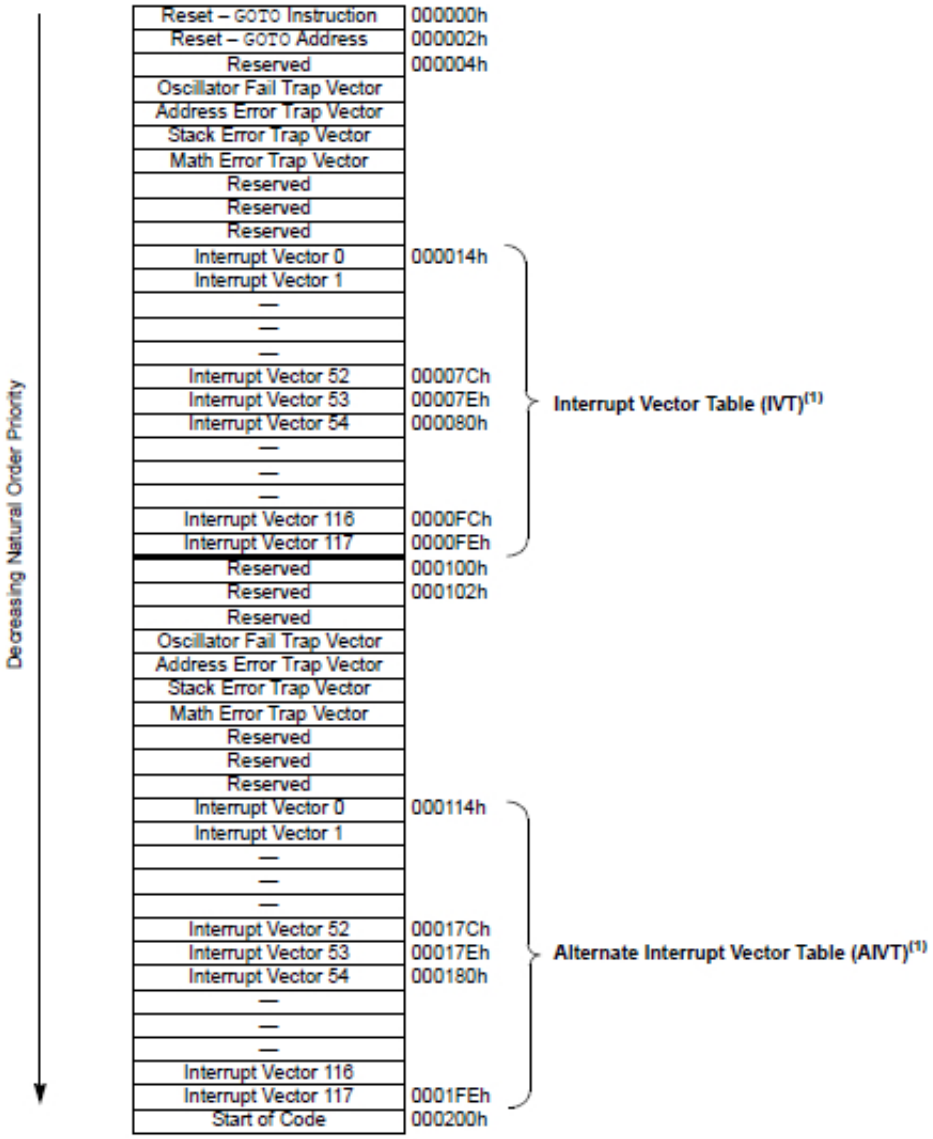


Figura 1.5: Interrupt Vector Table

priorità 0, allora l'interrupt ad essi associato è disabilitato.
Ricordandosi che esiste anche un livello di priorità naturale, nel caso in cui due richieste di interrupt avvengano contemporaneamente, la CPU andrà prima a vedere la priorità imposta in fase di programmazione e, se risultasse essere uguale per entrambe allora ricorrerebbe a quella dettata dalla posizione nella *Interrupt Vector Table*. Quindi il livello di priorità naturale viene usato unicamente per risolvere i conflitti tra due richieste di interrupt che avvengono simultaneamente.

CPU Priority Status

In questi PIC anche la CPU ha un livello di priorità, essa può assumere ben sedici diversi valori: da 0 a 15. Questo permette di mascherare gli interrupt indesiderati, in quanto gli interrupt aventi un livello di priorità non maggiore a quello della CPU vengono ignorati.
Ciò che è stato precedentemente affermato, ossia che per disabilitare una sorgente di interrupt basta porre a 0 il suo livello di priorità, trova qui riscontro. Infatti, tale livello di priorità non potrà mai essere superiore a quello della CPU, qualunque valore assuma quest'ultima.
Il valore corrente del livello di priorità della CPU è indicato attraverso i seguenti bit di stato:

- **IBL<2:0>** contenuti nel registro *SR*;
- **IPL3** situato nel registro *CORCON*.

I primi tre bit di stato, *IPL<2:0>*, possono essere scritti per mezzo del codice al fine di mascherare le sorgenti di interrupt configurabili dall'utente. Mentre l'ultimo bit, *IPL3* viene settato automaticamente dal microcontrollore se si verifica un *trap event*.

1.4.3 Traps

Come si può notare dalla (Fig. 1.5) ci sono otto vettori aggiuntivi che occupano le prime locazioni di memoria della *Interrupt Vector Table*. Questi vettori indicano particolari interrupt chiamati **traps** (trappole) e sono utilizzati per catturare speciali condizioni di errore.
La tabella di (Fig. 1.6) mostra nel dettaglio gli otto vettori di cui si sta parlando. Questi tipi di errore, che risultano essere abbastanza gravi, comportano generalmente delle conseguenze fatali che insidiano il corretto svolgimento dell'applicazione. Per questo motivo i *traps interrupt* hanno assegnato un livello di probabilità fisso ed al di sopra di tutti gli altri interrupt (compreso tra 8 e 15).
Siccome esistono degli eventi d'errore più gravi di altri e che quindi necessitano di un intervento più immediato, anche i traps interrupt sfuttano la tecnica dell'annidamento. Inoltre questi interrupt non sono soggetti al disattivamento del *nesting* e non sono mascherabili.
Studiando la tabella di (Fig. 1.6) si può notare come in realtà quattro di questi vettori siano inutilizzati ($Reserved$) e dunque implementati per un probabile uso futuro. Si ha inoltre che le traps, in ordine di priorità decrescente, sono causate da: un guasto dell'oscillatore ($Oscillator\ Failture$), un indirizzamento scorretto ($Address\ Error$),

Vector Number	IVT Address	AIVT Address	Trap Source
0	000004h	000104h	Reserved
1	000006h	000106h	Oscillator Failure
2	000008h	000108h	Address Error
3	00000Ah	00010Ah	Stack Error
4	00000Ch	00010Ch	Math Error
5	00000Eh	00010Eh	Reserved
6	000010h	000110h	Reserved
7	000012h	000112h	Reserved

Figura 1.6: Dettaglio sui traps vector

un errore dello stack ($Stack\ Error$) ed un errore di tipo matematico ($Math\ Error$).

Le due traps a priorità maggiore vengono chiamate **Hard Traps**, mentre alle due restanti viene dato il nome di **Soft Traps**. La differenza tra le due categorie è che le *Hard Traps* forzano la CPU ad interrompere l'esecuzione del codice dopo il completamento dell'istruzione che ha causato l'evento di trap, mentre le *Soft Traps* impiegano un ciclo di istruzione in più per la conferma della condizione di errore.

Il compilatore *MPLAB C30* associa ad ogni *trap vector* una routine di reset del processore. In ogni caso, è possibile andare a modificare la gestione di queste particolari interruzioni trattandole come normali interrupt.

1.5 Timer

Il progetto descritto in questa tesi prevede vari processi di acquisizione dati. Risulta utile, se non essenziale, fare in modo che questi processi avvengano con una cadenza prestabilita, e non in maniera irregolare. In particolare si desidera che i campioni acquisiti siano tra loro tutti equidistanti in modo tale da rendere agevole la definizione della finestra di osservazione, in quanto il tempo risulta frazionato.
La finestra di osservazione rappresenta quell'intervallo temporale entro il quale si eseguono un numero di letture dell'accelerazione predeterminate e, su queste letture verrà poi calcolato il valore di accelerazione efficace attraverso l'equazione:

$$a_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} a_i^2} \tag{1.1}$$

In (Eq. 1.1) si è indicato con a_i l'i-esimo campione di accelerazione letto e con N il numero di acquisizioni effettuate nella finestra di osservazione.
Risulta dunque evidente che se i campioni sono tra loro distanziati di un intervallo ΔT, allora l'ampiezza della finestra è:

$$T_W = \Delta T \cdot N \tag{1.2}$$

Per questo motivo si è dovuto ricorrere ad un **Timer**. Grazie ad esso è possibile gestire la parte dell'acquisizione dati ad interrupt e, in questo modo si riesce ad ottenere un ulteriore vantaggio in quanto si permette al microcontrollore di eseguire i calcoli nel tempo che intercorre tra una lettura e la sucessiva.
Un *timer* è un contatore programmabile interno al microcontrollore. Il PIC24FJ256DA210 possiede cinque timer a $16\ bit$ che vengono chiamati *Timer1*, *Timer2*, *Timer3*, *Timer4* e *Timer5*.
Eccezion fatta per il primo, è possibile far lavorare due timer a $16\ bit$ come se fossero uno solo a $32\ bit$ creando i *Timer2/3* e *Timer4/5*, questo può risultare utile per la generazione di conteggi lunghi, che però non servono in questo progetto.
Questi timer contano un certo numero di impulsi determinati dal programmatore, dopo di che si genera un interrupt dovuto alla saturazione del registro di conteggio.
La struttura interna del timer viene mostrata in (Fig. 1.7), dove si è fatto riferimento al Timer1. È possibile definire alcuni blocchi:

- **TMR1**, contenente il valore del conteggio;
- **T1CON**, registro di controllo del Timer1, consente di attivare il conteggio e di definire le sue modalità operative;
- **PR1**, viene utilizzato per produrre un reset periodico del timer.

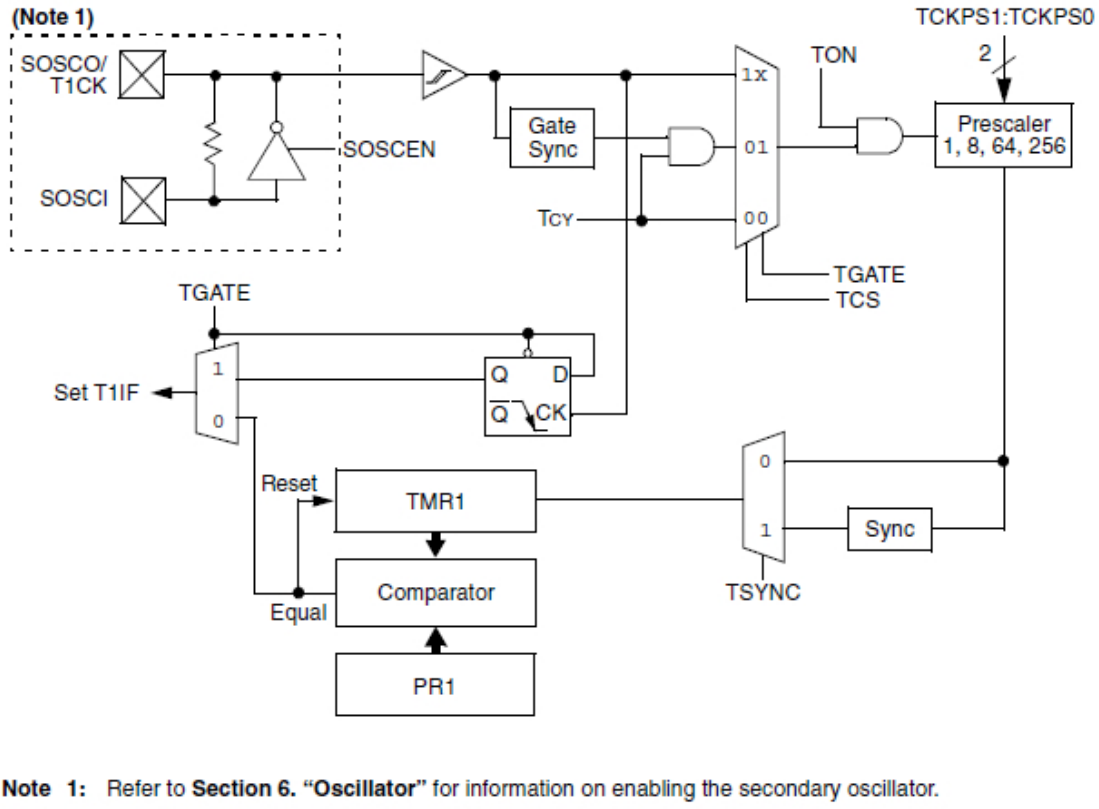


Figura 1.7: Schema a blocchi del Timer1

Per la generazione degli interrupt ogni ΔT secondi, occorre settare in maniera opportuna il registro *T1CON* e calcolare l'offset di partenza del conteggio che andrà poi precaricato nel registro *TMR1*.
Iniziamo con l'esaminare il registro di controllo del timer facendo riferimento allo schema a blocchi di (Fig. 1.7). Esso contiene:

- *TON*, bit di attivazione del conteggio;
- *TSIDL*, se a livello logico alto questo bit determina la sospensione del conteggio quando il microcontrollore entra in *Idle mode*;
- *TGATE*, questo bit viene ignorato quando *TCS* è ad 1 e serve per abilitare o meno il *Gate time accumulator mode*. Tale modalità prevede che il timer venga incrementato attraverso il clock interno (ossia ogni ciclo di istruzione T_{CY}) finchè il pin *T1CK* si mantiene allo stato logico alto. Quando, invece, questo pin si porta a 0 il conteggio si interrompe in quanto viene di fatto disabilitato il segnale di clock;
- *TKPS*, formato da due bit, serve per selezionare il valore del *Prescaler*, questo blocco serve per rallentare il segnale di clock fornito al suo ingresso;
- *TSYNC*, anche questo bit viene ignorato qualora *TCS* assuma un valore logico alto e, serve per abilitare la sincronizzazione del clock del timer con un segnale di clock fornito esternamente;
- *TCS*, è il bit di selezione della sorgente di clock, se posto a livello logico basso significa che il timer preleva il proprio clock da quello del microcontrollore, altrimenti lo preleva esternamente.

Di seguito saranno presentati i passaggi effettuati per generare un interrupt ogni millisecondo, quindi $\Delta T = 1\ ms$.
Siccome non viene richiesto alcun tipo di sincronizzazione con un segnale esterno e, non è necessario avvalersi del prescaler in quanto l'intervallo è sufficentemente piccolo, il registro di configurazione viene utilizzato unicamente per abilitare il conteggio.
La development board possiede un quarzo da $8\ MHz$ che grazie al *PLL* del microcontrollore consente a quest'ultimo di operare ad una frequenza di clock pari a $F_{OSC} = 32\ MHz$ (frequenza massima alla quale possono operarare i PIC24F). Questo significa che il periodo di clock è di $31.25\ ns$ in quanto:

$$T_{OSC} = \frac{1}{F_{OSC}} \tag{1.3}$$

Nella famiglia dei PIC24F inoltre, l'esecuzione di un istruzione richiede due cicli di clock. Per come è stato implementato il registro di configurazione del timer, si ha che il periodo con il quale avviene l'incremento corrisponde a quello del ciclo di istruzione e vale quindi $62.5\ ns$.

$$T_{TIMER} = T_{OSC} \cdot 2 \tag{1.4}$$

A questo punto occorre calcolare il numero di conteggi che il timer deve effettuare per impiegare un millisecondo. Questo numero viene fornito dalla seguente equazione:

$$N_{CONTEGGI} = \frac{\Delta T}{T_{TIMER}} \tag{1.5}$$

Quindi, il numero di conteggi è pari a 16000. Per avere dunque un interrupt ogni ΔT occorre precaricare nel timer, attraverso il *TMR1* il seguente valore:

$$TMR1_{PRECARICA} = 2^{16} - N_{CONTEGGI} \tag{1.6}$$

Quindi, in questo specifico caso si ha un valore di precarica pari a 49354.

Capitolo 2

Generazione della forma d'onda analogica

2.1 Introduzione

Per la generazione della forma d'onda che andrà a comandare i movimenti dello *shaker* occorre avvalersi di un convertitore digitale - analogico (**DAC**, Digital to Analog Converter).
I convertitori digitali - analogici abilitano l'interfacciamento dei sistemi digitali col mondo reale. Essi sono utilizzati per trasformare un codice binario fornito in ingresso in un segnale d'uscita analogico, sia sotto forma di tensione che di corrente.
L'uscita deve cambiare in funzione del tempo, esattamente come il valore digitale in ingresso cambia nel tempo. Siccome lo *shaker* necessita di un segnale in tensione, si è da subito escluso l'impiego di convertitori con l'uscita in corrente in quanto avrebbero richiesto una conversione del segnale in differenza di potenziale. Questo passaggio, che potrebbe introdurre degli errori dovuti dalla non idealità dei componenti, è di fatto ovviabile attraverso l'utilizzo di un convertitore D/A con uscita in tensione.
Per quanto riguarda l'ingresso, anche qui è possibile effettuare diverse scelte. Essendo una porta di tipo digitale, esistono infatti diversi modi attreverso i quali può comunicare[1]. Per usufruire della maggior frequenza di trasmissione possibile (che risulta essere un limite rispetto alla frequenza del segnale convertibile) la decisione più conveniete presa è stata quella di non utilizzare un convertitore ad interfaccia di ingresso seriale, bensì quella di avvalersi di uno con interfaccia di ingresso paralella.

2.2 Teoria

In base al range che il segnale analogico d'uscita può assumere i DAC si dividono in due macrocategorie. Ci sono infatti convertitori *bipolari* dove l'intervallo di valori assunti dalla forma d'onda d'uscita è simmetrico rispetto allo 0 e, quindi, il segnale analogico può assumere anche valori negativi. A questi, troviamo contrapposti i convertitori *unipolari* dove la forma d'onda d'uscita può assumere solo valori positivi.
In questo progetto sarà utilizzato un DAC unipolare in quanto lo shaker non ammette in

[1] Vedi capitolo 3

ingresso tensioni negative.
Quindi in questa sezione teorica ci si concentrerà sui convertitori unipolari e con l'uscita in tensione.
I DAC, in generale, richiedono due tensioni di riferimento, una positiva (V_{REF+}) ed una negativa (V_{REF-}). Siccome si sta analizzando il convertitore unipolare, esso richiede la sola tensione di riferimento positiva e sarà indicata semplicemente con V_{REF}. In questo modo, indicando con N il numero di bit di cui è composto il segnale d'ingresso del convertitore (osia la sua **risoluzione**), al bit meno significativo viene associata la tensione:

$$V_{LSB} = \frac{V_{REF}}{2^N} \tag{2.1}$$

La tensione d'uscita V_n può essere espressa in funzione della tensione di fondo scala e del codice applicato all'ingresso mediante la somma pesata di quei termini la cui corrispondente cifra binaria assume valore 1 secondo l'equazione (Eq. 2.2).

$$V_n = V_{FS} \sum_{r=0}^{N-1} \frac{B_r}{2^{M-r}} = V_{FS} \left(\frac{B_0}{2^N} + \frac{B_1}{2^{N-1}} + \cdots + \frac{B_{N-1}}{2} \right) \tag{2.2}$$

Le tipologie di circuito utilizzate per la realizzazione del convertitore sono due e differiscono tra loro dalla tipologia di rete di resistori utilizzata. Queste tipologie sono: *DAC con resistori di peso binario* e *DAC con rete a scala*. La prima è concettualmente la più semplice e si ricava osservando l'equazione (Eq. 2.2), da essa infatti è intuitivo l'utilizzo di un sommatore realizzato mediante un amplificatore operazionale e connesso ad una rete resistiva pesata, nella quale due valori sucessivi differiscono di un fattore 2. Si capisce che proprio quest'ultima condizione determina una difficoltà realizzativa circa il dimensionamento della rete, per questo motivo suddetta tipologia è poco utilizzata. La seconda tipologia di convertitori D/A, quella con rete a scala, è la più utilizzata (anche il DAC utilizzato in questo progetto appartiene a questa categoria[2]) quindi verrà studiata in maniera più approfondita, vedi (Sez. 2.2.1). Il grafico di (Fig. 2.2) rappresenta la funzione di trasferimento di un DAC ideale a 3 bit, da esso è semplice intuire come il componente stabilisca una corrispondenza unica tra l'ingresso digitale e la tensione d'uscita, che in realtà non assumerà una quantità infinita di valori, bensì un numero preciso e multiplo intero di V_{LSB}.

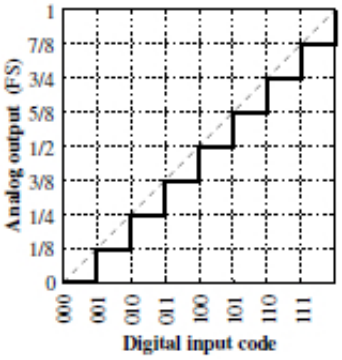


Figura 2.1: Funzione di trasferimento di un DAC a 3 bit

[2]Vedi sezione 2.3

2.2.1 DAC con rete a scala

Per implementare la funzione di trasferimento descritta dall'equazione (Eq. 2.2) si può utilizzare un'opportuna rete a scala composta solamente da due tipi di resistori uniti tra loro da un rapporto 2, ossia R e $2R$. Oppure la stessa rete a scala può essere effettuata utilizzando un solo resistore R questo grazie all'utilizzo di un decoder di tipo $N-to-2^N$. Un esempio di questa topologia circuitale viene fornita in (Fig. 2.2). In questo modo grazie al decoder che comanda la rete resistiva si è in grado di partizionare la tensione di riferimento utilizzando 2^N resistori. Si realizzano dunque tutti i valori assumibili dalla tensione d'uscita avvalendosi di un semplice partitore di tensione, al quale va aggiunto in cascata un operazionale connesso in modalità buffer per separare il circuito da tutto ciò che segue.

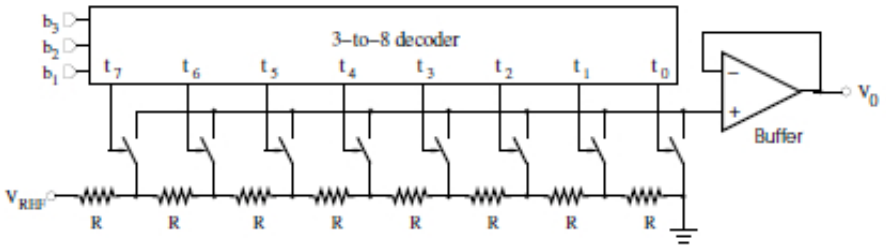


Figura 2.2: Scala di resistori di un convertitore D/A a 3 bit

2.2.2 Parametri caratteristici

Nella scelta del convertitore A/D appropriato, occore conoscere bene il significato dei parametri caratteristici. Fra questi due sono già stati illustrati e sono la **grandezza fisica d'uscita** e la **risoluzione**. A questi occorre aggiungere l'**accuratezza**, ossia la differenza tra il valore d'uscita del convertitore e quello desiderato. Questo parametro viene identificato attraverso alcune possibili sorgenti di errore come:

- **Reference voltage error**, rappresenta la dispersione dell'effettivo valore della tensione di riferimento attorno al valore nominale, essa viene generalmente espressa da diversi parametri che la descrivono in funzione della temperatura e della tensione;
- **Nonlinearity error**, è l'errore dovuto alla non linearità della caratteristica di trasferimento del componente. A causa di questo errore la conversione può apparire a passo variabile e non più fisso. La massima differenza tra due gradini successivi viene indicata con *DNL* e chiamata *errore di non linearità differenziale*. Bisogna prestare attenzione a questa sorgente d'errore, in quanto, se ecessiva, può compromettere la monotonia della caratteristica di ingresso - uscita (Fig. 2.2);
- **Output buffer amplifier error**, spesso i convertitori A/D hanno in uscita uno stadio amplificatore implementato attraverso un amplificatore operazionale, questo componente può introdurre ulteriori errori dovuti alla sua non idealità;

Un ultimo parametro utile è il **rapporto segnale - rumore** (*SNR*, Signal to Noise Ratio), il rumore è un disturbo indesiderato sovrapposto al segnale utile e questo rapporto indica quanto il nostro componente è immune ai disturbi.

2.3 LTC1450

Il componente LTC1450 è un convertitore digitale - analogico a 12 bit che gode di alcune importanti caratteristiche come l'alimentazione singola, la tensione d'uscita *rail - to - rail* ed un package a 24 pin dove il significato di ognuno di essi viene riporatato in (Fig. 2.4) e spiegato successivamente.
Il fatto che l'uscita sia *rail - to - rail* implica che la sua dinamica è approssimabile a quella ideale. Ossia, la tensione analogica può assumere valori molto vicini a quelli di riferimento.

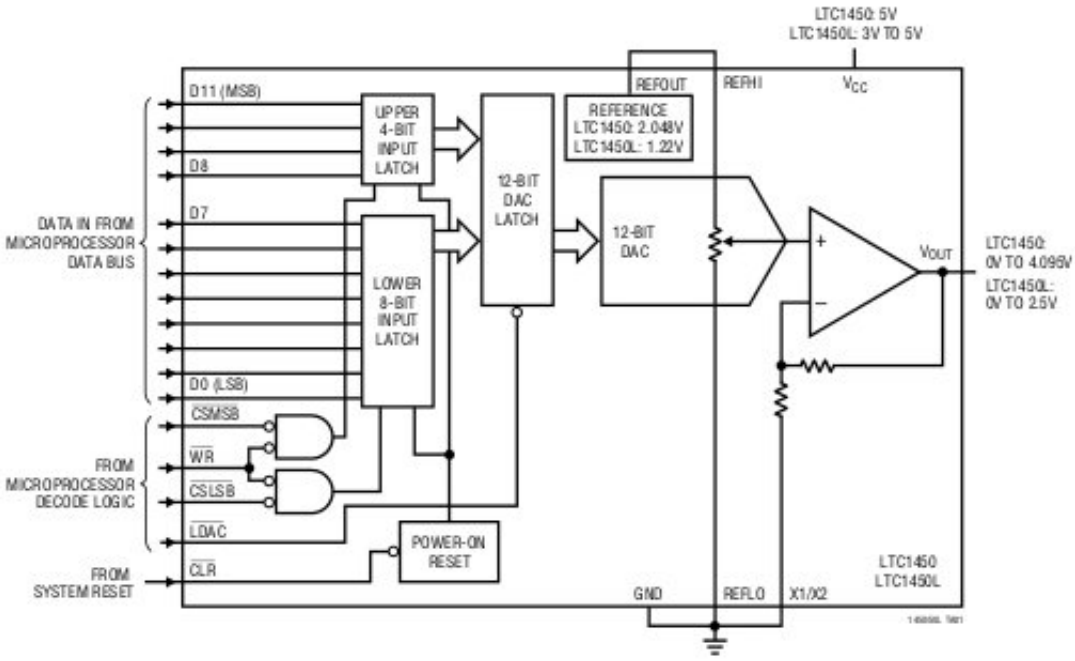


Figura 2.3: Schema a blocchi del LTC1450

Come mostrato in (Fig. 2.3) il DAC considerato comprende un buffer d'amplificazione in uscita ed un doppio buffer parallelo all'ingresso.

2.3.1 Funzione dei pin

Il componente dall'esterno può essere visto come mostrato in (Fig. 2.4), dove i pin assumono i seguenti significati:

- $\overline{\mathbf{WR}}$, *Write Input*: è un segnale attivo basso. La sua funzione è facilmente intuibile dallo schema a blocchi del componente (Fig. 2.3), esso assieme ai segnali $\overline{CSMSB}$ e $\overline{CSLSB}$ ha il compito di abilitare o meno la lettura dei dati in ingresso. Quando questi tre segnali sono mantenuti bassi i registri d'ingresso entrano in modalità trasparente.

- $\overline{\mathbf{CSLSB}}$, *Chip Select Least Significant Byte*: è un segnale attivo basso. Esso viene utilizzato con $\overline{WR}$ per caricare i dati dentro gli otto registri d'ingresso che compongono il byte meno significativo.

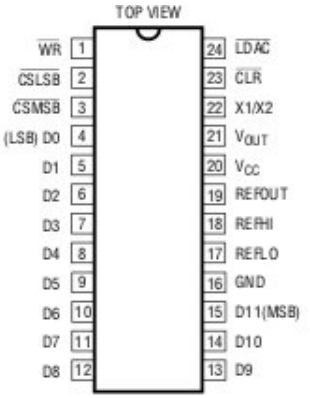


Figura 2.4: LTC1450, denominazione dei pin

- $\overline{\mathbf{CSMSB}}$, *Chip Select Most Significant Byte*: è un segnale attivo basso. Il suo funzionamento è esattamente analogo a quello descritto in precedenza, infatti esso viene utilizzato con $\overline{WR}$ per caricare i dati dentro i quattro registri d'ingresso che compongono la parte più significativa. Nel progetto qui descritto i pin $\overline{CSMSB}$ e $\overline{CSLSB}$ sono collegati assieme, in modo da consentire il caricamento simultaneo nei registri d'ingresso di tutti i 12 bit che compongono il dato da convertire.

- $\mathbf{D0 - D7}$, rappresentano l'ingresso dati del Least Significant Byte.

- $\mathbf{D8 - D11}$, rappresentano l'ingresso dati del Most Significant Byte.

- **GND**, *Ground*, massa.

- **REFLO**, rappresenta il riferimento a potenziale inferiore della rete resistiva che compone il convertitore. In questo specifico caso tale riferimento è la massa (GND).

- **REFHI**, rappresenta il riferimento a potenziale maggiore della rete resistiva che compone il convertitore. Nel progetto questo segnale è stato collegato al pin d'uscita REFOUT.

- **REFOUT**, è un segnale d'uscita in tensione a 2.048 V e, come precedentemente anticipato, viene utilizzato per fornite il riferimento al DAC interno.

- $\mathbf{V_{CC}}$, è ingresso attraverso il quale viene fornita l'alimentazione positiva (5 V).

- $\mathbf{V_{OUT}}$, è il segnale d'uscita in tensione avente l'andamento analogico.

- $\mathbf{X1/X2}$, denominato *Gain Setting Resistor Pin*. In base a dove viene connesso si determina il guadagno dell'amplificatore d'uscita. In particolare, connettendolo a V_{OUT} si ha un guadagno unitario, mentre connettendolo a GND il guadagno dell'amplificatore è pari a 2. Siccome si è assunto un riferimento a 2.048 V per poter avere in uscita a tale componente un segnale avente un'escursione maggiore di tale valore, si è scelto di connettere tale pin a massa, garantendo così un limite superiore del segnale d'uscita pari a 4.095 V.

- $\overline{\mathbf{CLR}}$, *Clear Input*: è un segnale attivo basso. Se attivato avvia il reset dei latch interni.

- $\overline{\text{LDAC}}$, *Load DAC*: è un segnale attivo basso. Viene utilizzato in maniera asincrona per caricare sui latch dell'ADC il dato presente nei registri d'ingresso e, conseguentemente, avviare la conversione digitale - analogica ottenendo dunque una forma d'onda al pin V_{OUT}.

2.3.2 Operazioni

Interfaccia parallela

Il dato fornito in ingresso al DAC, attraverso i 12 bit $D0$ - $D11$, viene caricato all'interno dei registri d'ingresso quando i *Chip Select* e il *Write Input* sono a valore logico basso.
Il dato in ingresso viene campionato e memorizzato dentro i suddetti registri in corrispondenza del fronte di salita o del *Write Input* o del *Chip Select* [3]. Prima di poter essere sottoposto al processo di conversione, il dato deve essere caricato dentro dei registri successivi, denominati *DAC Latch*, ciò avviene quando l'apposito segnale di *Load DAC* viene mantenuto basso e, in corrispondenza di un suo fronte di salita il processo ha inizio.

Riferimento di tensione

L'integrato LTC1450 include al proprio interno 2.048 V di tensione di riferimento che donano al componente stesso un valore di fondoscala della tesione d'uscita pari a 4.095 V, quando si è configurato a 2 il guadagno d'uscita.
Tale riferimento non è internamente connesso al convertitore, il collegamento deve essere effettuato esternamente. Questa scelta da parte del costruttore è stata mirata ad offrire una maggior flessibilità, in quanto si ha la possibilità di collegare un riferimento esterno. Come è stato precedentemente detto, in questo caso, ho preferito utilizzare il riferimento interno. Infine, per diminuire il rumore sul riferimento mi sono avvalso di un condensatore di bypass.

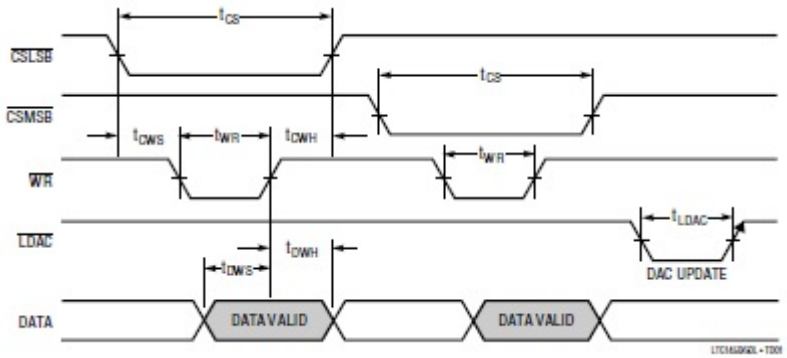


Figura 2.5: LTC1450, diagramma temporale

[3] Per maggiori informazioni circa le tempistiche si faccia riferimento alla (Fig.2.5)

2.4 Algoritmo di conversione

La conversione D/A, così come accade per il processo di acquisizione dell'accelerazione, viene avviato ogni millisecondo grazie all'interrupt generato dal Timer1. In questo modo si definisce una frequenza di campionamento di $1\ KHz$. Questa frequenza di campionamento rispetta ampiamente la condizione di Shannon necessaria per previnire il fenomeno dell'*aliasinng*. Infatti, ad un campionamento del segnale nel dominio del tempo corrisponde una ripetizione periodica, con periodo f_C (frequenza di campionamento), nel dominio delle frequenze. Se la frequenza di campionamento non è sufficientemente elevata si ha una sovrapposizione delle componenti spettrali del segnale determinando l'impossibilità di una sua ricostruzione nel dominio del tempo.
Affinchè il DAC sia in grado di ricostruire fedelmente la forma d'onda campionata, occorre dunque che sia rispettata la seguente disequazione, chiamata *teorema del campionamento di Shannon*:

$$f_C \geq 2 \cdot f_M \tag{2.3}$$

Dove con f_M si è indicata la massima frequenza alla quale il segnale ha ancora un'ampiezza apprezzabile.
Ciò che ci consente di affermare che $1\ KHz$ è una frequenza che rispetta il teorema espresso dalla (Eq. 2.3) è dato dal fatto che il campo di impiego dei sensori piezoelettrici viene caratterizzato dalle basse frequenze. A sostegno di questa affermazione vi è lo spettro di una tipica vibrazione alla quale questi sensori sono sottoposti (Fig. 2.6), come si può notare, la frequenza massima del segnale è nell'intorno dei $100\ Hz$.

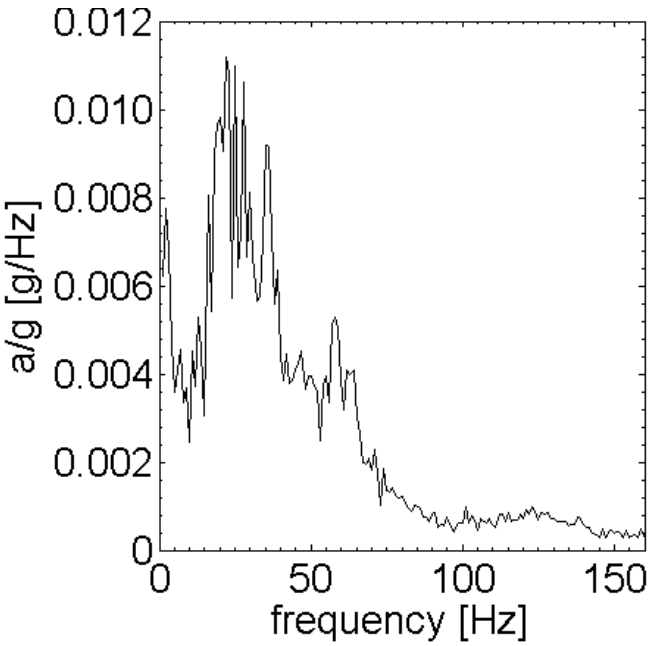


Figura 2.6: Spettro di una vibrazione causata dal treno

L'algoritmo di conversione si basa sul fatto che le forme d'onda, salvate in una memoria esterna, vengono rappresentate come dei vettori dove ogni elemento rappresenta un campione.

Ogni millisecondo dunque un campione viene caricato sui dodici pin d'uscita del microcontrollore che rappresentano la comunicazione parallela con il convertitore digitale - analogico, attraverso dodici maschere. Successivamente ha inizio la conversione vera e propria attraverso il seguente codice:

```
void ConvertiDAC(void)
{
        /*
         * Rendo i registri del Byte meno significativo
         * trasparenti
         */
        CSLSB = BASSO;
        /*
         * Rendo i registri del Byte piu' significativo
         * trasparenti.
         */
        CSMSB = BASSO;
        /*
         * In quanto e' nullo il tempo di hold dichiarato
         * dal costruttore si ha:
         */
        WR = BASSO;
        Nop();
        Nop();
        /*
         * Una volta che sono passati almeno 40 ns
         * produco un fronte di salita di WR al
         * fine di campionare tutti e 12 i bit caricati
         * nei registri di ingresso
         */
        WR = ALTO;
        /*
         * Posso riportare immediatamente a livello alto
         * i due pin che controllano i registri di ingresso
         * in quanto i datasheet dichiarano nullo il tempo di
         * setup
         */
        CSLSB = ALTO;
        CSMSB = ALTO;
        /*
         * Fatto cio' non mi resta che avviare la conversione
         * caricando i 12 bit sui latch del DAC, per far
         * cio' si genera un fronte di salita sul segnale LDAC.
         */
        LDAC = BASSO;
        Nop();
        Nop();
        LDAC = ALTO;

}
```

Capitolo 3

Periferiche di comunicazione

3.1 Introduzione

Un ruolo di fondamentale importanza in questo progetto viene svolto dalla comunicazione esistente tra il microcontrollore e le periferiche. Quindi, prima di procedere con l'analisi dei blocchi successivi è giusto dedicare un capitolo per descrivere come questa comunicazione avviene.
Il PIC24FJ256DA210 offre ben 11 periferiche di comunicazione progettate per assisterlo nelle comuni applicazioni di controllo embedded.
Tralasciando la periferica parallela ci si può concentrare sui protocolli di comunicazione seriali. Essi sono:

- 4 x **UART** (Universal Asynchronous Receiver-Transmitter);
- 3 x **SPI** (Serial Peripheral Interface);
- 3 x **I^2C** (Inter Integrated Circuit).

Queste periferiche possono essere raggrupate in due categorie differenti, vi sono infatti quelle *sincrone* (SPI e I^2C) e quelle *asincrone* (UART), in base al modo in cui l'informazione viene passata dal trasmettitore al ricevitore nel tempo.
Le periferiche di comunicazione sincrona hanno bisogno di apposite linee fisiche da dedicare al segnale di clock, in quanto esso viene condiviso tra i dispositivi interessati determinando, appunto, una sincronizzazione fra essi. Mentre, nella comunicazione asincrona ogni dispositivo coinvolto avrà un proprio segnale di clock interno, la sincronizzazione quindi non viene effettuata con la sua condivisione bensì attraverso delle informazioni aggiuntive trasmesse con i dati.
Quando siamo in presenza di una comunicazione seriale sincrona è uso comune chiamare *master* il device che origina il segnale di clock, e *slave* il device che si sincronizza con esso. In generale è possibile avere più di un master e/o più di uno slave, nel primo caso bisogna avere l'accortezza che solo un master alla volta controlli la comunicazione al fine di evitare conflitto elettrico.

In particolare, nel progetto ivi descritto si è in presenza di una situazione *master - multi slave* in quanto il ruolo di master viene ricoperto unicamente dal microcontrollore, mentre gli slave interessati sono due, l'accelerometro e il potenziometro digitale.

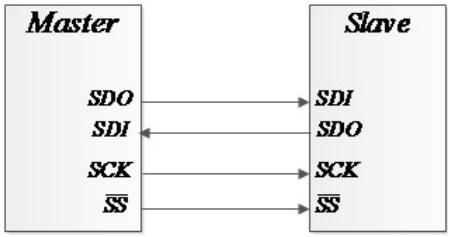


Figura 3.1: Schema a blocchi di una comunicazione SPI

3.2 SPI

La periferica di comunicazione seriale utilizzata nel progetto è la SPI (il perchè verrà esposto di seguito).

3.2.1 Teoria

L'interfaccia SPI fu inventata dalla Motorola (ora Freescale) e descrive la comunicazione tra un unico master e uno o più slave. Essa (a differenza dell'altro protocollo sincrono precendetemente citato, I^2C) prevede una scissione della linea dati in due linee fisicamente separate: una dedicata al flusso di dati in uscita dal master, l'altra dedicata invece ai dati in ingresso al master stesso. Tutto ciò consente di affermare che tale protocollo consente una comunicazione di tipo *full - duplex*, ossia è possibile sia trasmettere che ricevere dati in uno stesso intervallo di tempo.

Come mostrato nella (Fig. 3.1) una comunicazione SPI master - slave, è caratterizzata da 4 linee fisiche, per questo motivo tale protocollo viene anche chiamato *4 Wire Interface*. La denominazione di ogni linea utilizzata dalla Microchip è la seguente:

- **SDO**, Serial Data Out, chiamato anche *MOSI* (Master Output Slave Input);
- **SDI**, Serial Data In, chiamato anche *MISO* (Master Input Slave Output);
- **SCK**, Serial Clock, chiamato anche *SCLK* ;
- **SS**, Slave Select, chiamato anche *CS* (Chip Select).

Il master è il device incaricato ad avviare la trasmissione, può capitare anche che uno slave richieda l'avvio della comunicazione attraverso una *interrupt line* (linea d'interruzione). Ricevuta questa richiesta, il master andrà poi a leggere la periferica che l'ha generata.
Il master può gestire in parallelo più slave[1], per questo si necessita della presenza dello *Slave Select* attraverso il quale viene selezionato il device con cui avviare la comunicazione.

In (Fig. 3.2) è stato messo in evidenza come l'interfaccia SPI implementi un registro a scorrimento (*Shift Register*) al suo interno[2]

[1] Come nel caso mostrato in (Fig. 4.3)
[2] Nella figura si è in presenza di una comunicazione SPI ad 8 bit

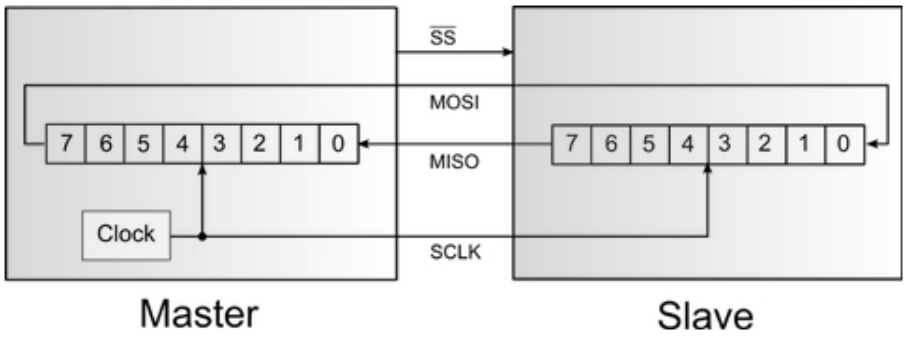


Figura 3.2: Comunicazione SPI mettendo in evidenza gli shift register

Funzionamento

Analizzando *step by step* la comunicazione SPI si nota che, prima di avviare il trasferimento dei dati, il master abbassa il livello logico della linea di *SS* relativa allo slave interessato. Successivamente, il master gli fornisce anche il clock attraverso la linea *SCK* alla frequenza con cui avverà la trasmissione. Ora, il trasferimento vero e proprio può avere inizio, i bit interni dentro lo *shift register* del master vengono trasmessi allo slave attraverso la linea *SDO* a partire dal bit più significativo, *MSB* (Most Significant Bit). Il bit trasmesso entra nello *shift register* dello slave, il quale, in maniera analoga, svuota in contemporanea il proprio contenuto inviando i bit attraverso *SDI* a partire sempre da quello più significativo.
La comunicazione termina quando anche l'ottavo bit viene trasmesso.
Le tempistiche dipendono dal settaggio effettuato per tale interfaccia e, quindi, verranno trattate nella sezione sucessiva (Sez. 3.2.2).

3.2.2 Interfaccia SPI nei PIC24F

Come già accennato il PIC24FJ256DA210 possiede ben 3 moduli SPI funzionalmente identici. Ogni modulo supporta due modalità operative differenti, lo *Standard mode* e l'*Enhanced Buffer mode*. Nel primo caso il dato sia in trasmissione che in ricezione passa attraverso un singolo registro, mentre nel secondo caso passa anche attraverso un registro ad 8 livelli di tipo *FIFO* (First Input First Output).
Lo schema a blocchi di un modulo SPI ivi contenuto è illustrato in (Fig. 3.3).

L'interfaccia SPI è dunque formata dai seguenti registri:

- **SPIxBUF** è il registro entro il quale il microcontrollore scrive i dati da trasmettere, nonché quello dove vengono scritti i dati ricevuti e resi accessibili attraverso il *bus dati* interno;
- **SPIxCON1** e **SPIxCON2** sono due registri di controllo attraverso il quale il modulo viene configurato per operare come desiderato;
- **SPIxSTAT** è un registro di stato (*status register*) dove vengono indicate le condizioni operative;
- **SPIxSR** è lo *shift register* attraverso il quale transitano i dati trasmessi e ricevuti.

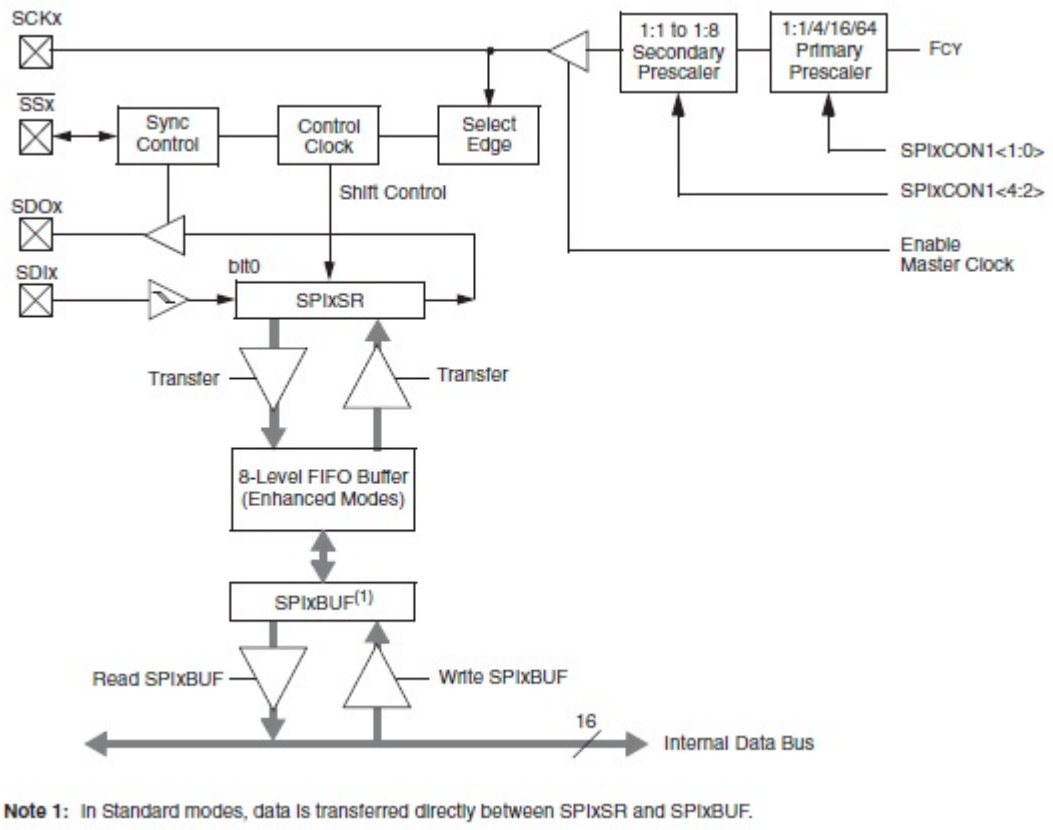


Figura 3.3: Schema a blocchi del modulo SPI interno ai PIC24F

SPI Status Register

Attraverso questo registro è possibile abilitare il modulo SPI, decidere se il modulo deve continuare ad operare anche quando entra nello stato di riposo (*Idle mode*), se si opera in *Enhanced Buffer mode* si possono settare 8 diversi eventi che determinano un'interruzione.
Questo registro, oltre alle funzioni appena esposte, risulta essere essenziale per sapere lo stato della trasmisione. Infatti, leggendo determinati bit si è in grado di sapere quanti dati sono in attesa di essere trasmessi (se il microcontrollore è settato per operare in modalità master, altrimenti esso indica il numero di dati che devono ancora essere letti), se lo shift register di trasmissione è vuoto, se si è verificato un *overflow*, se la trasmissione si è avviata e se il dato è stato ricevuto.

SPI Control Register 1

Questo registro consente di impostare la modalità operativa del PIC, in particolare consente di impostare il ruolo del suddetto in tale comunicazione, o è il device master o è lo slave, se la comunicazione avviene a 8 o a 16 bit e altre opzioni ancora che vengono illustrate di seguito.
Nella discussione che segue, si intenderà che il microcontrollore operi a 8 bit e in modalità master.

Come già detto, è il microcontrollore a fornire il clock agli slave e, tale clock è determinante per la velocità di trasmissione dei bit.
Il clock fornito viene prelevato da quello del PIC stesso e, prima di essere offerto allo slave attraversa due *prescaler*, in modo tale che esso venga rallentato. Così la frequenza alla quale avviene la comunicazione è il risultato della seguente equazione:

$$F_{SCK} = \frac{F_{CY}}{Primo\ Prescaler \cdot Secondo\ Prescaler} \tag{3.1}$$

Dove con F_{CY} viene indicata la frequenza di un ciclo di istruzioni, ossia la frequenza operativa del microcontrollore (F_{OSC} divisa per il numero di colpi di clock necessari affinchè un'istruzione elementare venga eseguita. Per la famiglia PIC24F vale la seguente relazione:

$$F_{CY} = \frac{F_{OSC}}{2} \tag{3.2}$$

Ad esempio, nell'impostazione del modulo SPI preposto alla comunicazione con l'accelerometro, quest'ultimo non è in grado di supportare una trasmissione dati a frequenza superiore ad $1\ MHz$. Siccome il PIC24FJ256DA210 viene fatto operare ad una frequenza pari a $F_{OSC} = 32\ MHz$ (quindi $F_{CY} = 16\ MHZ$) risulta necessario effettuare un'operazione di *prescale*. In particolare, in questo caso, per ottenere la frequenza di clock desiderata si può agire in due modi identici tra loro: o fare in modo che il *prescaler* primario riduca di un fattore 16 la frequenza in ingresso mentre il secondario la lascia immutata, oppure settare entrambe i *prescaler* affinché riducano di un fattore 4 le rispettive frequenze introdotte. Così si ha:

$$F_{SCK} = \frac{16\ MHz}{16 \cdot 1} = \frac{16\ MHz}{4 \cdot 4} = 1\ MHz$$

Modalità di trasmissione Sempre attraverso il registro *SPI Control Register 1* è possibile impostare la modalità di trasmissione. Esistono infatti ben quattro modi differenti riferiti al *Serial Clock* ed altri due modi relativi al campionamento del dato in ingresso.
Attraverso i due bit denominati **CKE** (*SPI Clock Edge Select bit*) e **CKP** (*Clock Polarity Select bit*) si è in grado di agire sull'andamento del *Serial Clock*, definendo così le sue quattro modalità operative.
Il primo bit, se è a livello logico alto, fa si che il dato in uscita presente sul *SDO* cambi durante il passaggio del clock dal *active state* al *Idle state*. Vice versa, se impostato a livello logico basso, il dato in uscita cambia sul passaggio del clock dal *Idle state* al *active state*.
Il secondo bit, invece, influisce sulla polarità del segnale di clock e, se impostato a 0 lo lascia immutato, mentre quando è imposto a 1 lo inverte. Questo determina direttamente i livelli logici del *Idle state* e del *active state*: se *CKP* è a livello logico alto, *Idle state* corrisponderà ad un 1 logico, mentre *active state* corrisponderà ad uno 0 logico. Naturalmente nel caso in cui il suddetto bit è posto a livello logico basso si verificherà l'esatto contrario.
Per quanto riguarda i due modi relativi al campionamento del dato in ingresso è **SMP** il bit da configurare nel registro di controllo. Se esso è imposto a 0 il campionamento avviene sul fronte di discesa del clock. Diversamente, se imposto a livello logico alto, il campionamento del dato avviene sul fronte di salita del segnale di clock.

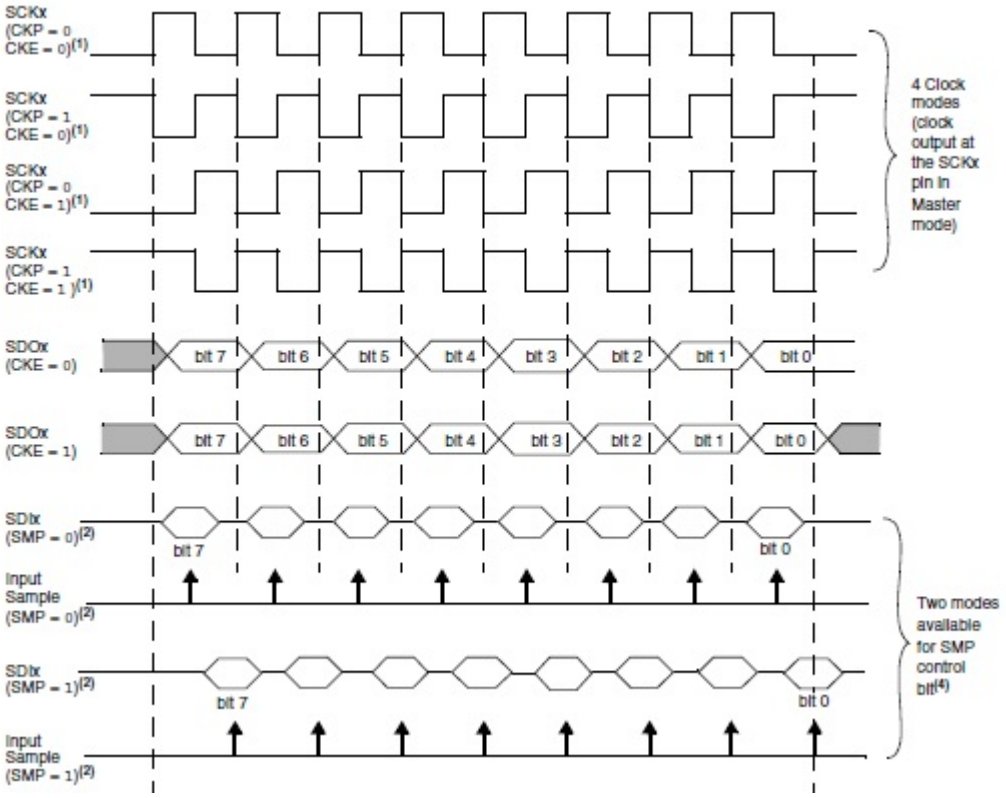


Figura 3.4: Diagramma temporale della comunicazione SPI

Le configurazioni selezionate non possono essere fatte casualmente, infatti per garantire una corretta comunicazione, occorre che slave e master abbiano la stessa modalità di trasmissione.
La (Fig. 3.4) evidenzia ciò che è appena stato esposto, evidenziando l'andamento temporale delle forme d'onda che caratterizzano la comunicazione in funzione di come sono stati configurati i tre bit *CKE*,*CKP* e *SMP*.

SPI Control Register 2

Questo secondo registro di controllo permette di abilitare o meno l'*Enhanced Buffer* ed altre impostazioni circa la sincronizzazione dei dati trasmessi.

Capitolo 4

Lettura delle accelerazioni

Nello sviluppo del progetto il processo di lettura delle accelerazioni e calcolo del valore efficace (o *RMS*, Root Mean Squared) è stato indispensabile. Infatti, sulla base di questo valore di accelerazione efficace il microcontrollore agirà di conseguenza regolando il valore del potenziometro digitale [1].
A tal fine è stato necessario ricorrere all'**accelerometro**.
L'accelerometro, così come dice la parola stessa, è un dispositivo in grado di rilevare un'accelerazione, ovvero una variazione di velocità nel tempo.

4.1 Principio di funzionamento

L'accelerometro è un sensore inerziale e, come la maggior parte di essi, è di tipo meccanico, ossia l'elemento sensibile risulta essere una massa (detta *proof - mass*) in grado di compiere un qualche movimento all'interno del dispositivo stesso. Questa microstruttura mobile viene realizzata mediante tecniche di **micromachining**.
Per consentire mobilità alla massa ci si avvale di un elemento elastico che, al tempo stesso, consente ad un sensore di rilevare lo spostamento rispetto alla sua condizione di riposo. In presenza di un'accelerazione, la massa (dotata di una propria inerzia) si sposta, il sensore trasforma questo spostamento meccanico in un segnale elettrico che poi, mediante un apposito circuito di condizionamento, verrà elaborato a dovere.
In base al principio di funzionamento del sensore di posizione è possibile fare una suddivisione tra i diversi accelerometri. In particolare, possiamo distinguerli in *accelerometri piezoresistivi*, *piezoelettrici*, *laser*,di tipo *tunneling* e di tipo *capacitivo*.
Gli accelerometri **piezoresistivi** sono stati tra i primi accelerometri ad essere realizzati e, come dice la parola stessa, il loro principio di funzionamento è basato sulla variazione del parametro resistenza causato da una deformazione della superficie del sensore.
Gli accelerometri **piezoelettrici** si basano sulla proprietà di *piezoelettricità*, caratteristica posseduta da alcuni materiali dielettrici (come il quarzo) che consiste nella generazione di cariche elettriche causate da una deformazione del materiale stesso. Quindi, in questo caso si fa uso di tale proprietà per convertire direttamente la forza indotta sulla *proof - mass* in un segnale elettrico.
Gli accelerometri **laser** sono caratterizzati da un laser che puntando verso la *proof - mass*

[1] vedi capitolo 5

interna al componente è in grado, istante per istante, di rilevare le posizioni che essa assume. In questo modo, l'accelerometro sfrutta le leggi della fisica classica percui l'accelerazione viene data dalla derivata seconda della posizione del punto rispetto al tempo. Questi accelerometri sono i più precisi ma, al tempo stesso sono anche i più costosi e i più espansivi, infatti al loro interno dovranno contenere anche un elaboratore per poter effettuare l'operazione di derivata.
Gli accelerometri di tipo **tunneling** sfruttano il legame che si instaura tra la *corrente di tunnel* generata da due elettrodi posti ad una determinata distanza. Questa corrente scorrerà solo se i due elettrodi sono sufficientemente vicini tra loro, in modo tale da consentire l'*effetto tunnel* e, ha un intensità che è funzione di questa distanza.
Gli accelerometri **capacitivi** sono la famiglia di accelerometri più diffusa in commercio, ne fa parte lo stesso KXP84 utilizzato in questo progetto (Sez. 4.2). In questi dispositivi la trasduzione dello spostamento della *proof - mass* avviene attraverso la variazione di una capacità. Il motivo della loro popolarità risiede negli innumerevoli vantaggi da essi posseduti, come l'alta resistenza al rumore, la bassa sensibilità alla temperatura, ottima precisione, costi contenuti ed altro ancora.
Il device presenta un condensatore variabile per ogni asse sul quale è possibile calcolare la componente di accelerazione. Questo condensatore può essere realizzato o come un condensatore variabile a due armature , dove una è collegata alla *proof - mass* e quindi mobile, oppure realizzato come un condensatore differenziale, composto da tre armature e, quindi è equivalente a due condensatori in serie. Questa volta l'armatura connessa alla massa mobile è quella in comune.

La fisica classica ci insegna che un corpo, libero di muoversi e per quanto complesso possa essere, risulta avere sei gradi di libertà. Gli accelerometri possono rilevare movimenti di rotazione, di traslazione o di torsione in base a come si muove la massa, movimento che viene vincolato lungo direzioni ben determinate ed ortogonali tra loro. Queste vengono chiamate direzioni di movimento o assi di sensibilità. In base al numero di questi assi si individuano gli accelerometri ad un asse, a due assi o a tre assi.

4.2 KXP84

Come ci viene detto dai datasheet del componente stesso, il KXP84 è un accelerometro triassiale che consente di rilevare e, conseguentemente misurare un'accelerazione compresa nell'intervallo $\pm 2g$ (dove con g si è indicata l'accelerazione gravitazionale e, quindi tale intervallo è equivalete a $\pm 19.6\ m/s^2$). L'elemento sensibile, la *proof - mass*, è realizzato dalla *Kionix* utilizzando un processo tecnologico a micromachining.
Esso risulta essere un accelerometro di tipo capacitivo, in quanto la rilevazione dell'accelerazione si basa sul principio di variazione della capacità indotta dal moto della massa inerziale. Il condizionamento del segnale è offerto da un dispositivo ASIC separato che inoltre offre anche una funzione di *self - test*. Questo dispositivo ASIC esterno risulta essere molto importante anche per il fatto che esso gestisce l'attivazione dei segnali di interrupt qualora la soglia di accelerazione venisse superata in uno qualsiasi dei tre assi (**motion interrupt**), o se la stessa accelerazione scende al di sotto di una certa soglia (**free - fall interrupt**). Come verrà mostrato conseguentemente, queste due soglie sono

Parameters		Units	Min	Target	Max
Supply Voltage (V_{dd})	Operating	V	2.7	3.3	5.25
I/O Pads Supply Voltage (V_{IO})		V	1.7	-	V_{dd}
Current Consumption	Operating	mA	0.6	1.0	1.7
	Standby	µA	-	-	10
Input Low Voltage		V	-	-	0.2 * V_{IO}
Input High Voltage		V	0.8 * V_{IO}	-	-
Input Pull-down Current		µA		0	
Analog Output Resistance(R_{out})		kΩ	24	32	40
Power Up Time[1]		ms		5*R_{out}*C	
A/D Conversion time		µs		200	
SPI Communication Rate[2]		MHz		1	
I^2C Communication Rate		KHz		400	

Figura 4.1: Specifiche elettriche del KXP84

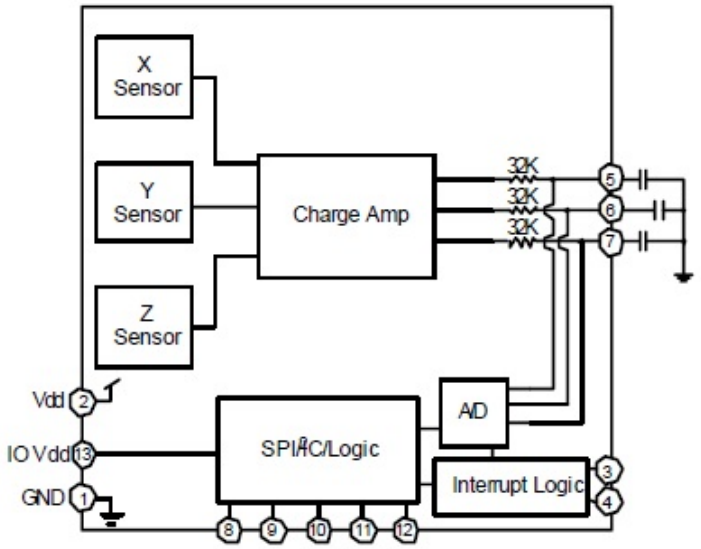


Figura 4.2: Diagramma funzionale del KXP84

sono configurabili tramite precise istruzioni.
Le interfacce di comunicazioni seriali disponibili sono quelle sincrone di tipo I^2C e SPI, in questo progetto si è deciso di utilizzare quest'ultima interfaccia in quanto, come mostrato nella tabella di (Fig. 4.1) garantisce una maggior velocità di trasmissione. La stessa tabella riporta inoltre le specifiche elettriche del componente.

Lo schema a blocchi funzionale del componente è visibile in (Fig. 4.2) dove si può notare come siano presenti tre sensori incaricati di rilevare l'accelerazione nei tre assi, il segnale uscente viene poi ampilificato tramite un amplificatore differenziale, questo consente di rimuovere il rumore di modo comune.
L'uscita dell'amplificatore viene poi fornita in ingresso ad un convertitore analogico - digitale, dopo esser passata attraverso un filtro passa - basso opzionale. Una volta che l'informazione sull'accelerazione è stata resa digitale viene fornita sia all'*interrupt logic* (che ha il compito di generare i due segnali di interrupt accennati pocanzi) sia all'interfaccia SPI, pronta per essere trasportata all'esterno.

Come detto pocanzi, il segnale d'uscita dall'amplificatore può esser sottoposto ad un processo di filtraggio grazie all'ausilio di tre condensatori (C_1, C_2 e C_3) collegati tra i pin 5, 6 e 7 del componente e la massa. La scelta della capacità da attribuire a questi tre condensatori è legata alla frequenza di taglio del filtro passa - basso (f_c) dalla seguente relazione:

$$C_1 = C_2 = C_3 = \frac{4.97 \cdot 10^{-6}}{f_c} \quad F \tag{4.1}$$

Al fine di eliminare il rumore causato dalla linea di alimentazione, la stessa *Kionix* , nella sua evaluation board, ha optato per una frequenza di taglio appena al di sotto dei 50 Hz, e quindi per questi tre condensatori il valore capacitivo attribuito è 0.1 μF. Facendo questa scelta si ottiene infatti una frequenza di taglio pari a:

$$f_c = 49.7 \quad Hz.$$

4.2.1 Interrupt

Come detto precedentemente il KXP84 fornisce due interrupt differenti, uno denominato *motion interrupt* (MOT) e l'altro denominato *free - fall interrupt*. I due interrupt sono indipendenti tra loro, ossia sono configurabili separatamente tra loro attraverso l'utilizzo di alcuni registri ad 8 bit interni al componente.

Motion Interrupt

Il *motion interrupt* si avvia quando viene rivelato un evento di *high-g*. Questo evento si verifica quando l'accelerazione su uno qualsiasi degli assi supera il valore della soglia alta per un certo tempo. Infatti, per evitare che sbalzi momentanei determino un interrupt, l'accelerazione deve mantenersi al di sopra di questa soglia per almeno un numero di passi di conteggio consecutivi impostabili dall'utente. Risulta essere un interrupt molto utile in quanto consente di interrompere la comunicazione qualora si stia facendo lavorare lo strumento a ridosso della saturazione.

Free-fall Interrupt

analogamente il *free-fall interrupt* si avvia quando viene rilevato un evento di *free-fall*. Questo evento si verifica quando l'accelerazione su tutti e tre gli assi dell'accelerometro scende simultaneamente al di sotto della soglia bassa per un certo tempo. Anche qui, una volta che questa soglia è stata attraversata si avvia un conteggio, se l'accelerazione si mantiene al di sotto di tale soglia per un certo numero di passi (sempre impostabili tramite i registri di controllo), l'interrupt si avvia.

4.2.2 La comunicazione SPI

Una volta convertita l'informazione sull'accelerazione attraverso un ADC interno al componente, essa risulta essere pronta per il trasferimento verso il microcontrollore. A tal fine si utilizza l'interfaccia SPI. In questa comunicazione l'accelerometro KXP84 risulta essere sempre lo slave e quindi non può aprire direttamente la comunicazione,

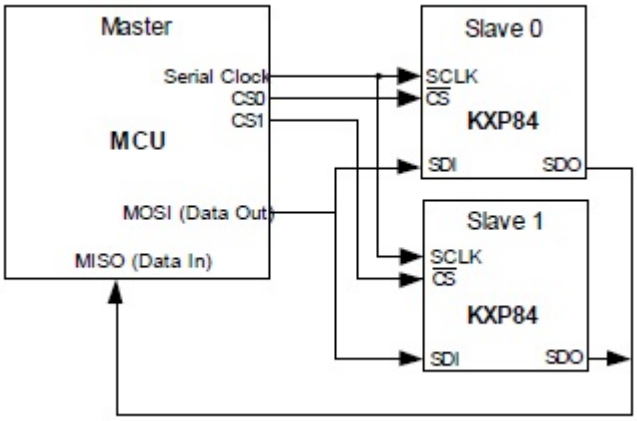


Figura 4.3: Connessione in parallelo di più KXP84

bensì può forzare il master a farlo sotto determinate condizioni. Infatti gli interrupt di cui si discuteva nella sezione precedente (Sez. 4.2.1) hanno il compito di generare un evento su due pin esterni che se collegati al microcontrollore e configurati a dovere, determinano l'apertura della comunicazione da parte di quest'ultimo.
Il KXP4 presenta un *Buffer Tristate* nel pin *Slave Data Output* (SDO), grazie ad esso è possibile collegare, qual'ora lo si desiderasse, più di questi componenti in parallelo (Fig. 4.3). Infatti, quando il device non è selezionato il corrispettivo SDO rimane nello stato di alta impedenza. Questo serve per evitare conflitti sul bus e le conseguenti degradazioni dei segnali trasmessi.

4.2.3 Registri di controllo

I registri di controllo contenuti all'interno del componente hanno 8 bit di indirizzamento, di questo dato occorre tenerne conto durante la configurazione dell'interfaccia SPI del microcontrollore in quanto, essendo a 16 bit, tutti i registri interni possiedono questa lunghezza e quindi anche i buffer d'ingresso e uscita del SPI.
Ogni volta che viene fornita l'alimentazione , il Master deve scrivere in maniera appropriata i registri di controllo dell'accelerometro, ogni registro di controllo necessita della trasmissione di due byte per essere settato. Il primo byte inizializza la scrittura sul registro desiderato (contiene dunque i bit di indirizzamento) mentre il secondo contiene i dati veri e propri che devono essere scritti sul registro e quindi i settaggi imposti dall'utilizzatore. Per capire se si vuole leggere o scrivere un determinato registro si va ad osservare il suo bit più significativo (MSB), qualora esso fosse a valore logico basso, 0, si intende scrivere sul registro, mentre se fosse a livello logico basso, 1, si desidera leggere il registro. La (Fig. 4.4) mostra il diagramma temporale per la scrittura di un registro di controllo, in particolare si può notare come siano necessari 16 cicli di clock e come tutti i comandi sono inviati a partire dal bit più significativo.

La (Fig. 4.5) mostra invece il diagramma temporale per la lettura di un registro di controllo. Le considerazioni che si ricavano sono intuitive, nonché le stesse della fase di scrittura.

Figura 4.4: Diagramma temporale per la scrittura di un registro di controllo del KXP4

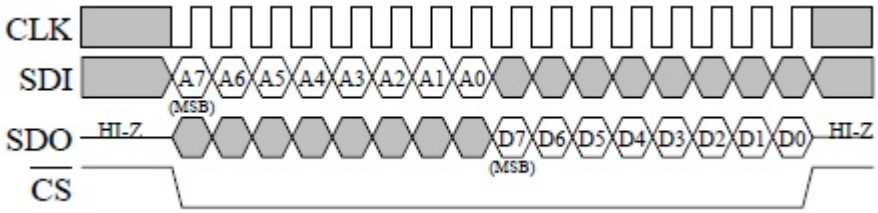


Figura 4.5: Diagramma temporale per la lettura di un registro di controllo del KXP4

4.3 Lettura dell'accelerazione

Il KXP84 contiene al suo interno un convertitore analogico - digitale a 12 che ha il compito di campionare, convertire e fornire il segnale letto dal sensore all'interfaccia SPI in modo tale da rendere possibile la sua lettura. Come già detto la comunicazione avviene attraverso registri ad 8 bit e quindi il dato per essere traferito necessita di due letture consecutive, una rivolta al registro contenente la parte più significativa, l'altra verso il registro contenente la parte meno significativa.
Come mostrato in (Fig. 4.6) una volta inviato il comando di lettura dell'accelerazione, il componente procede con la conversione del segnale contentenente l'informazione. Questa conversione, seppur veloce, non è immediata, infatti richiede almeno 200 μs, dopo di che è possibile leggere il dato.

4.3.1 Codice in C per la gestione dell'accelerometro

Per poter gestire in maniera opportuna il componente è stato necessario definire ed utilizzare tre funzioni differenti, una per l'accensione e l'inizzializzazione dell'accele-

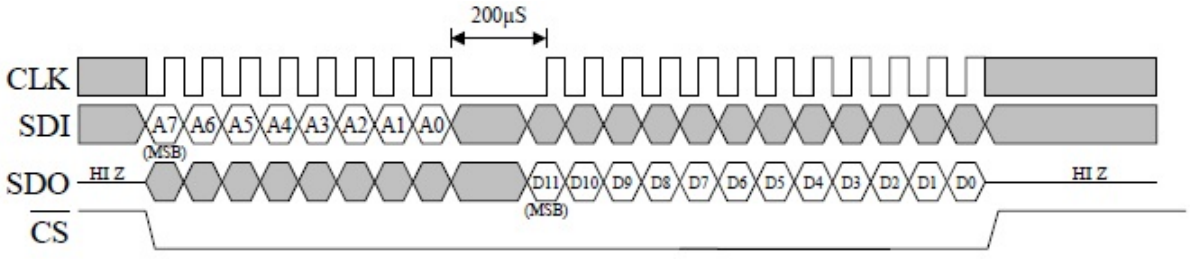


Figura 4.6: Diagramma temporale del processo di conversione analogico - digitale interna al KXP4

rometro, una per la lettura dell'accelerazione ed una per lo spegnimento del device. A tal fine è stato necessario avvalersi della tabella di (Fig. 4.7) dove vengono indicati gli indirizzi dei 13 registri interni accessibili all'utilizzatore.

Register Name	Type Read/Write	Read Address Hex	Read Address Binary	Write Address Hex	Write Address Binary
XOUT_H	R	0x80	1000 0000	xxxx	xxxx xxxx
XOUT_L	R	0x81	1000 0001	xxxx	xxxx xxxx
YOUT_H	R	0x82	1000 0010	xxxx	xxxx xxxx
YOUT_L	R	0x83	1000 0011	xxxx	xxxx xxxx
ZOUT_H	R	0x84	1000 0100	xxxx	xxxx xxxx
ZOUT_L	R	0x85	1000 0101	xxxx	xxxx xxxx
FF_INT	R/W	0x86	1000 0110	0x06	0000 0110
FF_DELAY	R/W	0x87	1000 0111	0x07	0000 0111
MOT_INT	R/W	0x88	1000 1000	0x08	0000 1000
MOT_DELAY	R/W	0x89	1000 1001	0x09	0000 1001
CTRL_REGC	R/W	0x8A	1000 1010	0x0A	0000 1010
CTRL_REGB	R/W	0x8B	1000 1011	0x0B	0000 1011
CTRL_REGA	R	0x8C	1000 1100	xxxx	xxxx xxxx

Figura 4.7: Indirizzi dei 13 registri interni al KXP84

```
void AccTurnOn ( void )
{
  /* Invio il comando di reset per cancellare il contenuto dei registri interni*/
  AccReset = 1;
  __delay_ms(1);
  AccReset = 0;

  /* Inizializzo l'SPI con prescaller a 16 */
  Open_SPI1(16);
  /* Abbasso lo slave select per abilitare la comunicazione con l'accelerometro*/
  SSacc = 0;
  /* Setteggio del registro REGB */
  Write_SPI1 (0x0b);
  Write_SPI1 (0x04);
  __delay_us(100);
  /* Settaggio del registro REGC */
  Write_SPI1 (0x0a);
  Write_SPI1 (0x03);
  __delay_us(100);
  /* Settaggio del registro FF_INT freefall interrupt */
  Write_SPI1 (0x06);
  Write_SPI1 (0x00);
  __delay_us(100);
  /* Settaggio del registro MOT_INT */
  Write_SPI1 (0x08);
  Write_SPI1 (0x4d);
  __delay_us(100);
  /* Settagio del registro MOT_Delay */
  Write_SPI1 (0x09);
  Write_SPI1 (0x14);
  /* Alzo lo slave select per terminare la comunicazione con l'accelerometro */
```

```
    SSacc = 1;
    Nop();
34  Nop();
    Nop();

}
```

Quel che compie questa funzione rappresenta la scelta fatta circa il funzionamento dell'accelerometro. In particolare tramite essa si precisa che l'unico interrupt che il device deve generare è quello dovuto ad un *motion event*, mentre deve trascurare quello determinato da un *free-fall event*. Tale evento verrà generato nel pin 3 del componente stesso. La frequenza con la quale il device va a controllare se sono presenti, o meno, le condizioni necessarie a far generare un interrupt (chiamata semplicemente *Interrupt Frequency*) è pari a $16\ KHz$, ossia il massimo impostabile. I tre cicli di $Nop()$ (*No Operation*) servono per assicurarsi che lo *Slave Select* resti alto per almeno $130\ ns$, come richiesto dalle specifiche.
Per quanto riguarda la seconda funzione, ossia quella dedicata alla lettura dell'accelerazione, il codice in $linguaggio\ C$ viene riportato di seguito:

```
#define LeggoH 0x80
#define LeggoL 0x81
3 #define AsseX 0x00
#define AsseY 0x02
#define AsseZ 0x04

int ReadAcc( char asse)
8 {
  /*Inizializzo la variabile che conterra' il valore della lettura*/
  unsigned int Dato;
  /* Abbasso lo slave select per abilitare la comunicazione con l'accelerometro*/
  SSacc = 0;
13 /*Invio il comando di lettura del byte piu' significativo dell'asse scelto*/
  Write_SPI1(LeggoH | asse);
  __delay_us(200);
  /*Carico in dato il MSByte*/
  Dato = Read_SPI1();
18 Dato = Dato<<8;
  Dato = Dato & 0xff00;
  /*Concateno al MSByte il LSByte, ottenuto mediante una seconda lettura*/
  Dato = Dato | Read_SPI1();
  /*Rimuovo i 4 bit privi di informazione*/
23 Dato = Dato >> 4;
  /* La lettura dell'accelerazione e' ora contenuta nella varidabile Dato*/
  Dato = Dato & 0x0fff;
  /* Alzo lo slave select per terminare la comunicazione con l'accelerometro */
  SSacc = 1;
28 Nop();
  Nop();
  Nop();
  return Dato;
```

}

Come si nota, questa funzione richiede inizialmente la lettura del $MSByte$, il processo, come si può notare, richiede un attesa di $200\mu s$. Questo intervallo è indispensabileall'accelerometro per convertire l'informazione da analogica a digitale. Questo $MSByte$ viene caricato in una variabile, mediante un secondo processo di lettura si accoda al primo $byte$ il $LSByte$ inerente alla prima lettura. Siccome il componente già da solo autoincrementa il registro di trasmissione, non è necessario indirizzare il registro contenente il $LSByte$ in quanto il suo dato verrà già caricato sullo *shift register* della perifetica SPI.
Lo $shift$ di quattro posizioni verso destra (riga 25 del codice) è necessario per rimuovere i 4 bit meno significativi che sono stati caricati nella variabile ma sono privi di informazione.
Infine, l'ultima funzione utile per controllare l'accelerometro è data dal suo spegnimento, o meglio dalla sua entrata in modalità standby (dove i consimi in termini di potenza sono molto bassi).

```
void AccTurnOff( void )
{
  /*Inizializzo l'SPI con prescaller a 16*/
  Open_SPI1(16);
  /* Abbasso lo slave select per abilitare la comunicazione con l'accelerometro*/
  SSacc = 0;
  /*Imposto il KXP84 in modalita' a bassa potenza*/
  Write_SPI1(0x0B);
  Write_SPI1(0x40);
  __delay_us(100);
  /* Settaggio del registro REGC */
  Write_SPI1(0x0A);
  Write_SPI1(0x03);
  SSacc = 1;
  Nop();
  Nop();
  Nop();
}
```

Questa funzione è simile alla prima. Si può notare che diversamente a quanto accadeva nell'accensione, qui oltre ad impostare il KXP84 a bassa potenza, si disabilitano anche gli interrupt che esso può generare.

4.3.2 Algoritmo per il calcolo dell'accelerazione efficace

L'algoritmo per il calcolo dell'accelerazione efficace è rappresentato dallo schema a blocchi di (Fig. 4.8), dove si può vedere che ogni millisecondo il dato contenente l'informazione sull'accelerazione presente sull'asse Z (l'asse presso il quale aviene il movimento) viene memorizzato all'interno di un vettore, come suo elemento i-esimo ($a(i)$). Successivamente il microcontrollore aggiorna il valore della variabile b, contenente al suo interno la somma dei quadrati dei valori del'accelerazione, ossia:

$$b = b + a(i)^2 \tag{4.2}$$

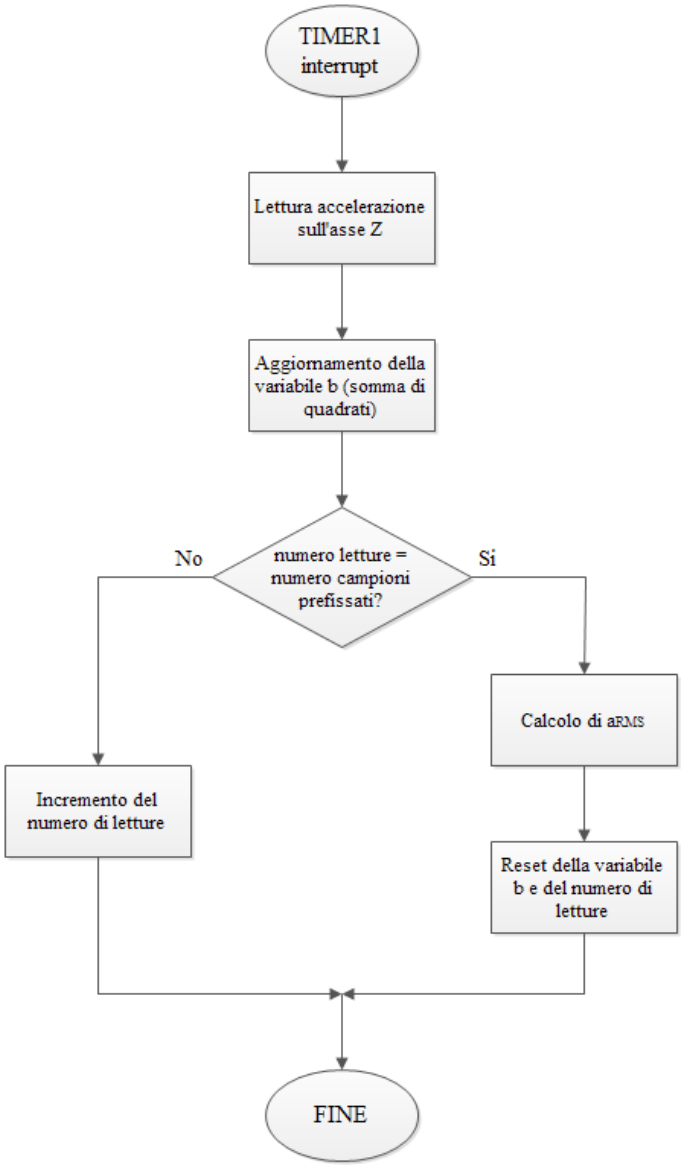


Figura 4.8: Diagramma a blocchi dell'algoritmo di calcolo dell'accelerazione efficace

Dopodiché viene effettuata una verifica sul numero di accelerazioni lette, se queste eguagliano il numero N di campioni prefissati sul quale calcolare il valore efficace allora si avvia il calcolo di a_{RMS} e si resettano i valori di i e b, dove i rappresenta l'indice del campione corrente e quindi il numero di letture. Altrimenti, se il numero di letture prefissate non fosse stato ancora raggiunto, l'algoritmo prevede l'incremento dell'indice i.Per calcolare a_{RMS} non si fa altro che dividere b per il numero di campioni e, succesivamente effettuare la radice quadrata. Per ottimizzare il codice, si può fare in modo che il numero di camioni sia esprimibile come una potenza del due ($N = 2^m$), così, anziché dividere per N è sufficiente effettuare il shift di m posizioni verso destra.

4.3.3 Visualizzazione sul display LCD

Come è stato sottolineato nell'introduzione, questo progetto si prefissa l'obiettivo di poter operare *stand-alone*. Per far ciò è necessario fornire un'informazione leggibile circa l'accelerazione efficacie imposta dal movimento dello shaker. A tal scopo si è utilizzato un display LCD 4x20 con controllore **HD44780** (prodotto da *Hitachi*).
Il dato calcolato tramite l'algoritmo precedente ci fornisce l'informazione riguardante l'accelerazione efficace, ma non risulta essere direttamente leggibile. Prima di poter scriverlo sul display LCD, questa informazione necessita dunque di alcuni passaggi per il suo condizionamento.
Come si nota dal codice che segue, il primo passaggio effettuato è la rimozione dell'offset, infatti anche se lo shaker non sta compiendo alcun movimento, l'accelerometro segnala la presenza di un'accelerazione pari a $1g$ (dovuta all'accelerazione gravitazionale). I datasheet ci forniscono un intervallo entro il quale si troverà questo valore. Anziché porre fiducia al dato di *target* dichiarato dalla casa produttrice, ho preferito ricavare suddetto valore sperimentalmente. Una volta rimoso l'offset occorre esprimere la grandezza per unità di g, in questo caso, non essendo possibile effettuare alcuna verifica per via sperimentale, mi sono dovuto fidare di quanto scritto sui documenti del componente.

```
void ScriviAcc(float aRMS)
{
  /* Rimozione dell'offset di 1g*/
  aRMS = aRMS - 2120;
  /*Esprimo la grandezza per unita' di g*/
  aRMS = aRMS / 819;
  LCD_PUTSN(aRMS);
}
```

Capitolo 5

Regolazione dell'ampiezza del segnale

Una volta che è stata misurata l'accelerazione efficace (a_{RMS}), essa viene confrontata con il valore imposto dall'utilizzatore e, se risultasse differente occorre modificare l'ampiezza del segnale fornito in ingresso allo shaker di conseguenza.
Per far ciò il microcontrollore dovrà regolare il fattore di partizione di un partitore posto in cascata al convertitore digitale - analogico. Questo partitore viene realizzato utilizzando un potenziometro digitale ed un amplificatore operazionale connesso in modalità buffer.

5.1 Potenziometro digitale

Il potenziometro è un dispositivo elettronico a tre terminali rappresentabile mediante un partitore di tensione. La sua particolarità risiede nel fatto che, tramite una sollecitazione esterna, il componente è in grado di modificare il rapporto di partizione e quindi appare come un divisore di tensione variabile. Spesso questo device viene chiamato anche *resistore variabile* o *reostato* ma in realtà questi termini non dovrebbero essere confusi. Infatti, la resistenza variabile, o reostato, è un componente a due terminali avente il compito di modificare il parametro resistenza fra di essi. Naturalmente, per come è fatto, il potenziometro può operare come resistenza variabile semplicemente cortocircuitando tra loro due terminali, il *wiper* (che tradotto in italiano vuol dire tergicristallo e viene indicato con W) e uno tra i due restanti.

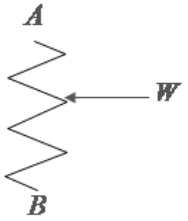


Figura 5.1: Rappresentazione di un potenziometro

Facendo riferimento alla (Fig. 5.1) possiamo affermare che il parametro resistenza tra il terminale A ed il terminale B (R_{AB}) resta costante, mentre i valori di resistenza tra i terminali A e W (R_{AW}), e tra B e W (R_{BW}) cambiano in funzione della posizione assunta dal cursore.
Quando si parla di **potenziometro digitale** il concetto a *black - box* rimane sempre

lo stesso, quello che cambia è il modo in cui il dispositivo interagisce e in cui esso è internamente costituito. Infatti essi modificano il loro valore basandosi sul messaggio di controllo ricevuto.
I potenziometri digitali sono disponibili in una moltitudine di interfacce di controllo come il *1 - wire*, I^2C, *SPI*, parallela ed altre ancora. Alcuni di essi possiedono al proprio interno una memoria non volatile, essa può risultare utile in quanto consente di memorizzare l'ultima posizione assunta dal device prima che l'alimentazione gli venisse rimossa.

Una delle principali differenze tra il potenziometro meccanico ed il potenziometro digitale risiede nella *risoluzione*. In un potenziometro meccanico il rapporto di partizione varia con continuità tramite la rotazione. Un potenziometro digitale, invece, gode di una maggior accuratezza grazie ad una stringa di resistori e a degli interruttori, connessi come mostrato in (Fig. 5.2). Conseguentemente, il valore commuta a passi, *steps*.
Facendo riferimento sempre al circuito di (Fig. 5.2) si possono definire alcune grandezze che, aggiunte a quelle già introdotte per la descrizione generale di potenziometro, risulteranno essenziali nella progettazione del blocco ivi studiato, il *regolatore d'ampiezza del segnale*:

- $\mathbf{R_S}$, è la *step resistance*. Rappresenta il cambiamento resistivo tra uno step ed il successivo, è definita anche come il rapporto tra il valore resistivo R_{AB} e la risoluzione del componente.
- **Risoluzione**, è il numero di resistenze R_S presenti all'interno del potenziometro digitale, nonché il numero di step.
- $\mathbf{R_W}$, è la *wiper resistance*. Essa Rappresenta la resistenza parassita introdotta dagli interruttori, ossia la resistenza presente tra la *Resistor Ladder* (scala di resistori, ossia la serie di tutti i resistori R_S) e il terminale di wiper.

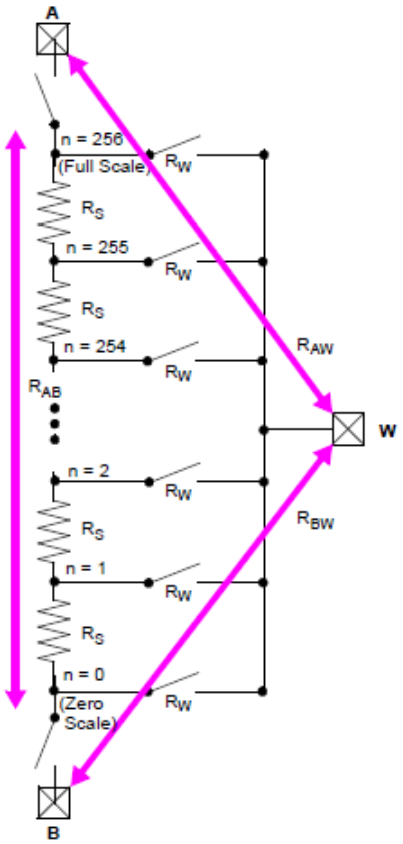


Figura 5.2: Rete resistiva di un potenziometro ad 8 bit

5.2 MCP41010

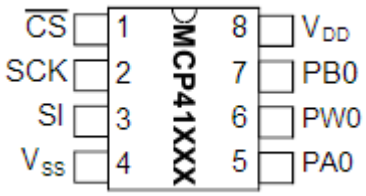


Figura 5.3: Diagramma dei pin del MCP1010 nel package PDIP

Il potenziometro digitale che è stato utilizzato è il *MCP41010* prodotto dalla *Microchip*. Questo componente presenta 256 step e la resistenza tra i due terminali *A* e *B* è di $R_{AB} = 10\ K\Omega$.
La resistenza di wiper è $R_W = 52\Omega$, mentre per quanto detto finora è facile ricavare la resistenza di step come:

$$R_S = \frac{10 \cdot 10^3}{256} = 39.0625\Omega \tag{5.1}$$

La memoria posseduta dal MCP41010 è di tipo volatile, questo vuol dire che una volta rimossa l'alimentazione il componente resetta tutti i suoi registri. Quando viene fornita

l'alimentazione, i registri interni vengono caricati in modo tale che la posizione del wiper sia esattamente a metà della sua escursione, quindi $R_{AW} = R_{BW}$
In prima approsimazione si può assumere che la posizione del wiper varia linearmente, essa è inoltre controllata tramite l'interfaccia SPI.
Facendo riferimento all'immagine di (Fig. 5.3) si definiscono qui di seguito i vari pin del componente, esponendo per ognuno di essi la sua funzionalità:

- $\overline{\mathbf{CS}}$, *Chip Select*, è la porta utilizzata nella comunicazione SPI per selezionare il potenziometro;
- $\mathbf{SCK}$, *Serial Clock*, è l'ingresso del clock fornito dal master nella comunicazione SPI;
- $\mathbf{SI}$, *Serial Input*, è l'ingresso seriale entro il quale transitano i dati trasmessi con la periferica SPI;
- $\mathbf{V_SS}$, *Ground*, è la massa;
- $\mathbf{V_DD}$, *Power Supply*, è l'alimentazione;
- $\mathbf{PB0}$, *Potentiometer terminal B*, è il terminale *B* del potenziometro;
- $\mathbf{PW0}$, *Potentiometer Wiper connection* , è il terminale *W* del potenziometro;
- $\mathbf{PA0}$, *Potentiometer terminal A*, è il terminale *A* del potenziometro.

Sfruttando il diagramma a blocchi di (Fig. 5.4) possiamo studiare la sua struttura interna, nonché il suo comportamento.

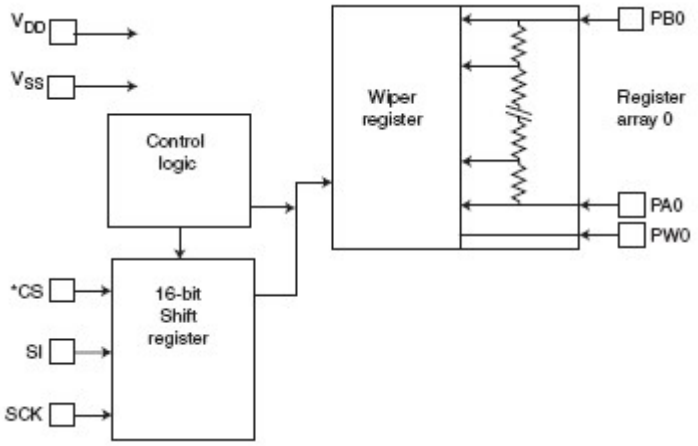


Figura 5.4: Diagramma a blocchi del MCP1010

Come si può notare il componente ha un funzionamento molto semplice, i dati trasmessi tramite la periferiche SPI vengono salvati in un registro denominato *Wiper register*. Il valore contenuto in questo registro è quello che determina la chiusura o l'apertura degli interruttori che connettono il terminale di wiper con la rete di resistori.
Lo *Shift register* con la quale la comunicazione SPI avviene è a $16\ bit$, quindi è possibile sia impostare il modulo SPI del PIC24FJ256DA210 affinchè avvii una comunicazione 16

bit, sia affinchè venga avviata a 8 bit, in questo secondo caso naturalmente serviranno due cicli di scrittura.
I 16 bit vengono suddivisi in due $Byte$, il primo viene denomito *Command Byte*, mentre il secondo *Data Byte*.

Command Byte

Il *Command Byte* contiene due bit per la selezione del comando da inviare ed uno per la selezione del potenziometro. Nei dispositivi MCP2XXX i bit di selezione del potenziometro sono due in quanto ne integrano due anziché uno.
I comandi che possono essere selezionati sono illustrati nella tabella di (Fig. 5.5), dove $C1$ e $C2$ corrispondono rispettivamente al sesto e quinto bit.

Command Selection Bits			
C1	**C0**	**Command**	**Command Summary**
0	0	None	No command executed.
0	1	Write Data	Write the data contained in the instruction byte to the potentiometer(s) selected by the selection bits.
1	0	Shutdown	Potentiometer(s) selected by the selection bits enter shutdown mode. Data bits for this mode are "don't care."
1	1	None	No command executed.

Figura 5.5: Command Byte del MCP1010

I comandi selezionabili sono dunque tre:

1. Scrittura di un nuovo valore che determina il posizionamento del wiper;
2. Spegnimento del dispositivo;
3. Nessun operazione.

Data Byte

Il *Data Byte* permette di selezionare in maniera univoca una delle 256 posizioni assunte dal wiper, dove, quando il suo valore è 0 vuol dire che il wiper è connesso al terminale *B*, mentre se vale 255 è connesso al terminale *A*. Quindi al variare di questo parametro, variano i valori resistivi R_{AW} e R_{BW} secondo le seguenti equazioni:

$$R_{AW} = R_{AB} \cdot \frac{256 - D_n}{256} + R_W \tag{5.2}$$

$$R_{BW} = R_{AB} \cdot \frac{D_n}{256} + R_W \tag{5.3}$$

Dove con D_n si è indicato il dato inviato attraverso il *Data Byte* e memorizzato nel *Wiper register*.

5.3 Algoritmo di controllo

Partitore resistivo Il partitore resistivo rappresenta una delle tipologie di circuito base dell'elettronica, utilizzando il potenziometro digitale precedentemente descritto, esso viene rappresentato dalla configurazione circuitale di (Fig. 5.6) dove si è supposto che al terminale *A* venga applicata la tensione di ingresso (V_1), al terminale di wiper venga estratta la tensione d'uscita (V_2) ed al terminale *B* viene connessa direttamente la massa. In questo modo il legame presente tra V_1 e V_2 è definito dal parametro A, attenuazione.

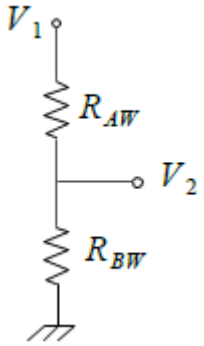


Figura 5.6: Partitore resistivo

$$A = \frac{V_2}{V_1} = \frac{R_{BW}}{R_{AW} + R_{BW}}. \tag{5.4}$$

Ricordandosi dello schema a blocchi del sistema (Fig. 1) si può capire lo scopo di questo blocco di attenuazione. Il convertitore D/A viene fatto lavorare sempre a fondo scala, ciò significa che la tensione d'uscita ha una dinamica che va da 0 a $4,095$ volt. Spesso quest'escursione, se fornita direttamente all'ingresso dello shaker determina un'accelerazione efficace superiore a quella desiderata. Quindi, utilizzando il partitore resistivo, si fa in modo che tale escursione risulti un limite superiore.
Il modo in cui viene utilizzato il potenziometro digitale è mostrato in (Fig. 5.7) ed è così espostro:
il potenziometro digitale è la prima periferica ad essere inizializzata e, si carica come valore di partenza $D_n = 0$, in questo modo la funzione di trasferimento del partitore resistivo avrà il valore minimo, ossia:

$$A = \frac{V_2}{V_1} = \frac{R_{BW}}{R_{AW} + R_{BW}} = \frac{R_W}{R_{AB} + 2R_W} \simeq 5.173 \cdot 10^{-3} \tag{5.5}$$

Si ha quindi, che inizialmente, la dinamica dell'onda analogica fornita in ingresso allo shaker avrà un massimo dato da qualche decina di millivolt ($\simeq 21.183\ mV$).
Una volta calcolato il valore di a_{RMS}, qualora risultasse inferiore di quello impostato dall'utente si aumenta di un'unità il dato trasmesso al potenziometro, aumentando quindi la funzione di trasferimento del blocco di regolazione.
Analogamente, per qualche motivo può capitare che l'accelerazione efficace impostata sia inferiore di quella calcolata, in questi casi si procede in maniera duale, ossia si riduce di un'unità il dato trasmesso al potenziometro, causando una diminizuione della dinamica dell'onda fornita in ingresso allo shaker.

Si è scelto di effettuare passi di un singolo step per volta perchè è preferibile impiegare più tempo per arrivare a regime e non rischiare di oltrepassare di molto la soglia imposta, rispetto a impiegare meno tempo ma correndo il rischio di esporre il sensore piezoelettrico ad un accelerazione eccessiva che porterebbe alla sua rottura.

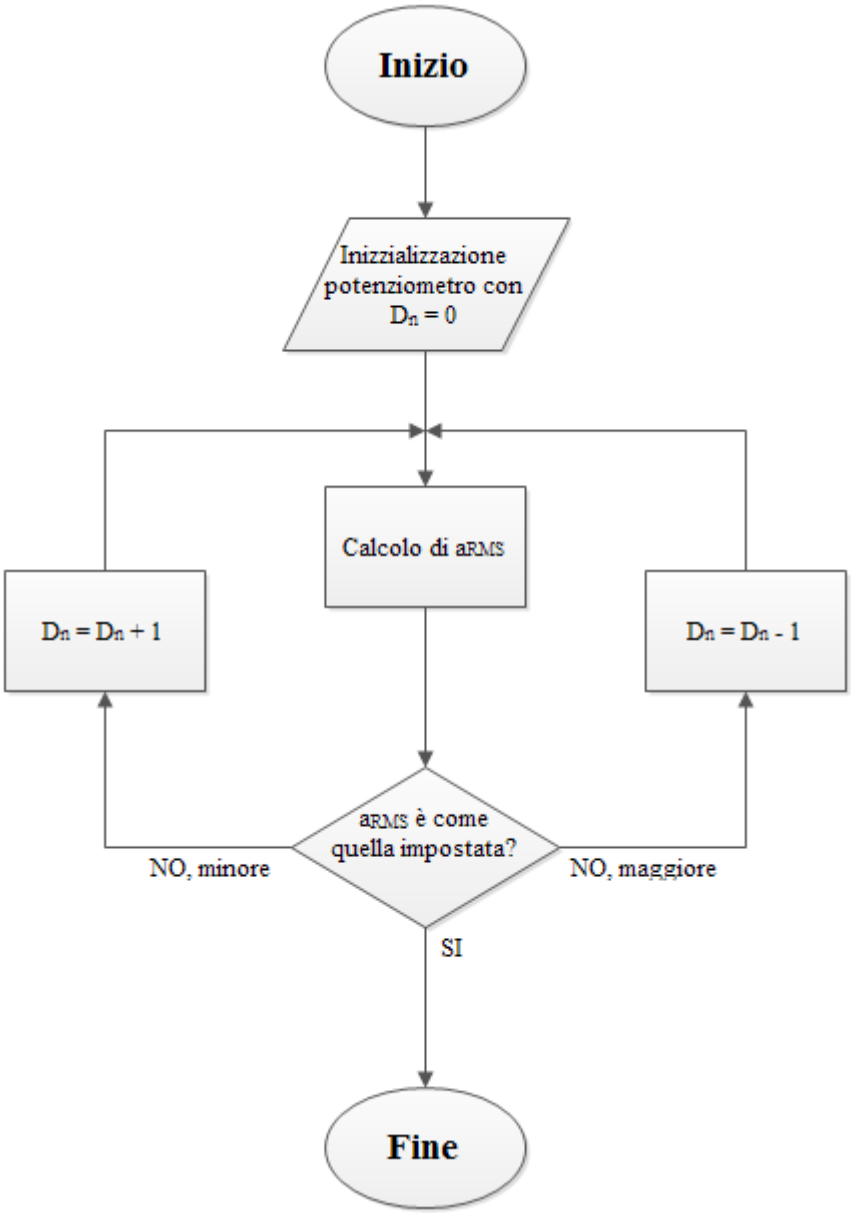


Figura 5.7: Schema a blocchi del controllo in retroazione

Capitolo 6

Lettura della tensione

L'ultima funzione descritta che va a concludere il progetto è data dalla lettura della tensione analogica fornita in ingresso allo shaker.
Questo dato può risultare utile per diversi motivi, come ad esempio la conoscenza dell'ampiezza della tensione di una sinusoide (ad una data frequenza) che determina una accelerazione efficace nota.
Per la realizzazione di questo blocco è stato necessario avvalersi del convertitore analogico - digitale interno al microcontrollore.

6.1 Convertitore analogico - digitale

Il processo di conversione A/D rappresenta tipicamente il primo passaggio per l'elaborazione di un segnale, questo perchè, il mondo che ci circonda è caratterizzato da grandezze analogiche mentre un elaboratore, qualsiasi esso sia (microcontrollore, DSP, FPGA, ecc...), è in grado di operare solamente con le grandezze digitali.
Il **convertitore analogico - digitale** (*ADC*, Analog to Digital Converter) è il componente duale al DAC visto nel capitolo 2.
La conversione di un segnale analogico in uno digitale viene effettuata attraverso due sottoprocessi:

1. *quantizzazione*;
2. *codificazione in binario*.

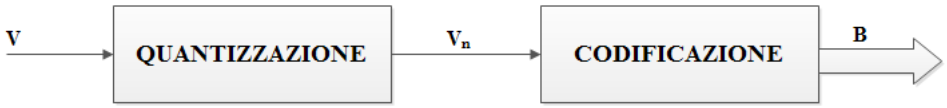


Figura 6.1: Processo di conversione A/D visto come cascata di due sottoprocessi

Quantizzazione

L'operazione di quantizzazione ha il compito di trasformare una variabile continua (V) in una discreta (V_n), ossia in una variabile che può assumere un numero finito di

valori. Questo sottoprocesso consiste quindi nella suddivisione della variabile continua in tante parti, il numero di queste parti è legato alla risoluzione del convetirtore, e il loro insieme definisce il *livello del convertitore A/D*.
Per compiere questa discretizzazione il processo definisce un **passo di quantizzazione** (V_q) e, misura la variabile continua con questo passo, associando al risultato della misura il suo intero più prossimo, ossia:

$$V_n = INT\left\langle \frac{V}{V_q} \right\rangle \cdot V_q = n \cdot V_q \tag{6.1}$$

Il processo di quantizzazione è per sua natura un processo non reversibile e, per come viene definito implica intrensicamente una perdita di informazione proporzionale al passo di quantizzazione. Tale perdita viene denominata **errore di quantizzazione** (ΔV_ε) e, la sua conoscenza in termini assoluti equivale alla conoscenza del valore assunto in quell'istante dall'ingresso analogico. Si sa con certezza l'intervallo di valori in cui cadrà l'errore di quantizzazione, esso è $\Delta V_\varepsilon \in \left[-\frac{V_q}{2}; \frac{V_q}{2}\right]$.
Si nota come una diminuzione del passo di quantizzazione implichi una diminuzione dell'errore ad essa associato.
Anche nel caso dei convertitori A/D si ha la suddivisione in unipolari e bipolari. Nel caso di mio interesse, come precedentemente esposto, le grandezze in gioco sono positive e quindi limiterò lo studio che segue al caso dei soli ADC unipolari. Quindi, questi convertitori necessitano della sola tensione di riferimento positiva, la *tensione di fondo scala* (V_{FS}). Si ha dunque che il passo di quantizzazione è definito come:

$$V_q = \frac{V_{FS}}{2^m} \tag{6.2}$$

Dove con m si è inicato il numero di bit, e quindi la risoluzione, del convertitore in esame.

Codificazione in binario

Il sottoprocesso di codificazione prende in ingresso un livello di tensione e ne fornisce una codifica binaria. La codifica, a differenza della quantizzazione, è invece un processo reversibile, la sua funzione inversa viene infatti offerta dal DAC e la caratteristica ingresso - uscita che deve rispettare è data, appunto, dalla (Eq. 2.2).

Le tre variabili che definscono e rappresentano in maniera univoca il funzionamento di un ADC sono dunque V_q, V_{FS}, m. Il valore di fondo scala viene spesso chiamato *variabilità dell'ADC* in quanto in generale, la grandezza analogica d'ingresso può variare entro un range di valori ampio V_{FS}.

6.1.1 Convertitore ad approsimazioni sucessive

I convertitori analogico - digitali per quanto complessi possano essere, vengono suddivisi in tre differenti categorie:

- ADC che effettuano la conversione in maniera quasi istantanea, compromettendo però la qualità del processo (convertitori flash);

- ADC che effettuano la conversione impiegando un determinato intervallo di tempo entro il quale la grandezza d'ingresso debba essere mantenuta costante (convertitori ad approsimazioni sucessive);

- ADC che effettuano la conversione impiegando un tempo prefissato, normalmente più lungo dei precedenti, entro il quale la grandezza di ingresso viene integrata (convertitori ad integrazione).

I microcontrollori possiedono al loro interno un *convertitore ad approsimazioni successive* e quindi questa categoria sarà qui esposta.

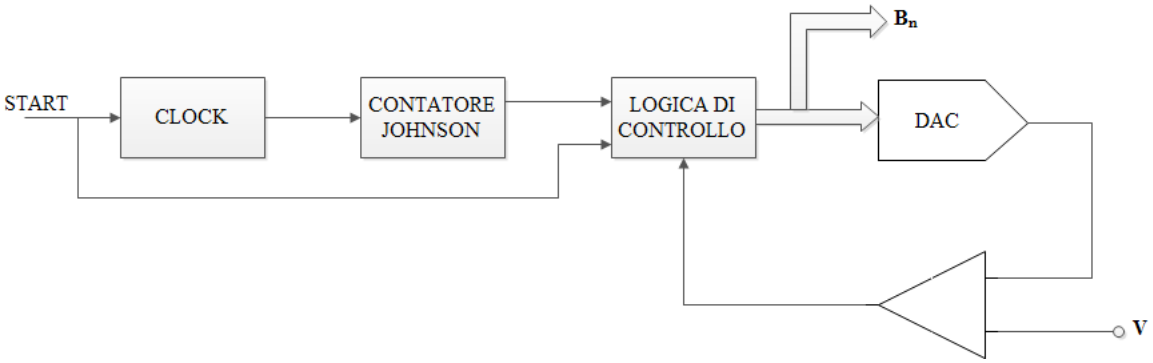


Figura 6.2: Schema a blocchi del convertitore ad approsimazioni sucessive

Il convertitore analogico - digitale ad approsimazioni successive (*SAR*, Successive Approximation Register) viene rappresentato dallo schema a blocchi di (Fig. 6.2). Il suo funzionamento è alquanto semplice, esso effettua la conversione avvalendosi del controllo in retroazione nella quale vengono confrontate la tensione d'uscita del DAC e quella V posta in ingresso all'ADC.
Prima di esporre i passaggi effettuati da questo componente per compiere il processo di conversione, occorre conoscere come opera il *contatore Johnson*. La particolarità di questo contatore è data dal fatto che ad ogni impulso di clock esso porta a livello logico alto una cifra binaria in uscita a partire dal bit più significativo.

Funzionamento

La comprensione del funzionamento di questo convertitore viene ricondotta a quella dell'algoritmo implementato dalla *logica di controllo*. In particolare, la conversione può essere suddivisa nella seguente successione di operazioni:

- L'inizio della conversione avviene attraverso l'apposito comando di $START$ che abilita il contatore Johnson ad $m\ bit$. Così, in corrispondenza del primo impulso di clock il bit più significativo all'uscita del contatore si porta a livello logico alto, mentre i restanti $m-1$ bit restano a livello logico basso. Per come è definito il DAC, questo dato al suo ingresso comporta un'uscita in tensione avente ampiezza $V_{FS}/2$.

- Quest'ampiezza viene confrontata con quella della tensione posta in ingresso al convertitore grazie ad un comparatore. Se, tale tensione risultasse inferiore a quella d'uscita dal convertitore D/A, il bit più significativo verrebbe memorizzato dalla logica di controllo come uno 0. Viceversa se il risultato della comparazione avesse decratato che è la tensione V ad essere maggiore, allora il bit più significativo sarebbe stato salvato come 1.

- Il secondo colpo di clock determina la transizione allo stato logico alto del bit successivo a quello più significativo, all'uscita della logica di controllo questa condizione si andrà ad aggiungere a quella memorizzata precedentemente.

- A tale codice corrisponde un livello di tensione in uscita al DAC che verrà sottoposto nuovamente al processo di comparazione. Analogamente a quanto accadeva precedentemente, il risultato di questo passaggio permette di determinare se questa seconda cifra binaria è corretta. Se così non fosse, la logica di controllo provvederà a memorizzare come bit successivo a quello più significativo uno 0.

- Questo processo si itera nella maniera sopra esposta per tutti i restanti bit.

Per come è stato descritto il processo si ha che, considerando un convertitore a m bit di risoluzione, il ciclo di conversione necessita di m impulsi di clock. Quindi si può affermare che avvalendosi di un convertitore SAR è possibile ottenere un'elevata velocità di conversione.
Siccome prima dell'avvio di un nuovo ciclo di clock si necessita che il dato abbia compiuto il ciclo di operazioni per il controllo in retroazione, la frequenza di clock risulta inferiormente limitata. Il periodo di clock deve risultare superiore alla somma del tempo richiesto per l'assegnamento dell'uscita del DAC e il tempo di risposta del comparatore. Tutta la trattazione del funzionamento circuitale si basa sulla condizione per cui la tensione in ingresso resti costante durante tutto il processo di conversione. Per assicurarsi che tale condizione venga mantenuta, spesso a monte del convertitore ad approsimazioni successive si utilizza un circuito di *sample and hold* (campiona e mantieni).

6.2 Modulo ADC interno al PIC

Vista l'importanza ricoperta dal convertitore analogico - digitale in un qualsiasi circuito elettronico, moltissimi microcontrollori possiedono al loro interno almeno un modulo ADC.
Il PIC24FJ256DA210 vanta ben 24 ingressi analogici dal quale è possibile selezionare il segnale che dovrà essere sottoposto al processo di conversione, mediante un convertitore ad approsimazioni sucessive (Sez. 6.1.1). Il convertitore A/D interno è a 10 bit ed è possibile utilizzare una tensione di riferimento esterna, fornita mediante due pin analogici aggiuntivi.
Il modulo ADC interno al PICmicro viene rappresentato dallo schema di (Fig. 6.3), dove è possibile vedere i suoi registri di controllo. In essa si nota come i 24 ingressi analogici vengono collegati a due multiplexer (*MUXA* e *MUXB*) usati per la selezione del canale che deve essere inviato al sample and hold (S&H). Ogni multiplexer fornisce due uscite differenti chiamate V_{INL} e V_{INH}, esse rapresentano rispettivamente la tensione

di ingresso negativa e positiva del S&H. La tensione positiva è dunque data dal pin selezionato come ingresso per la conversione A/D, mentre la tensione negativa viene connessa al riferimento negativo fornito al modulo, che in questo caso altro non è che GND.
Sempre dalla solita (Fig. 6.3) si evince come all'ingresso dei due multiplexer, oltre ai pin analogici, sono collegati anche altri quattro canali: uno d'ingresso denominato V_{CAP} (terminale positivo dell'alimentazione dell'alimentazione del core) e tre rifermenti interni: V_{BG}, $V_{BG}/2$ e $V_{BG}/6$. Dove con V_{BG} viene indicata la tensione di riferimento di *band gap*.
Il modulo fornisce il risultato della conversione attraverso un buffer composto da 16 registri.

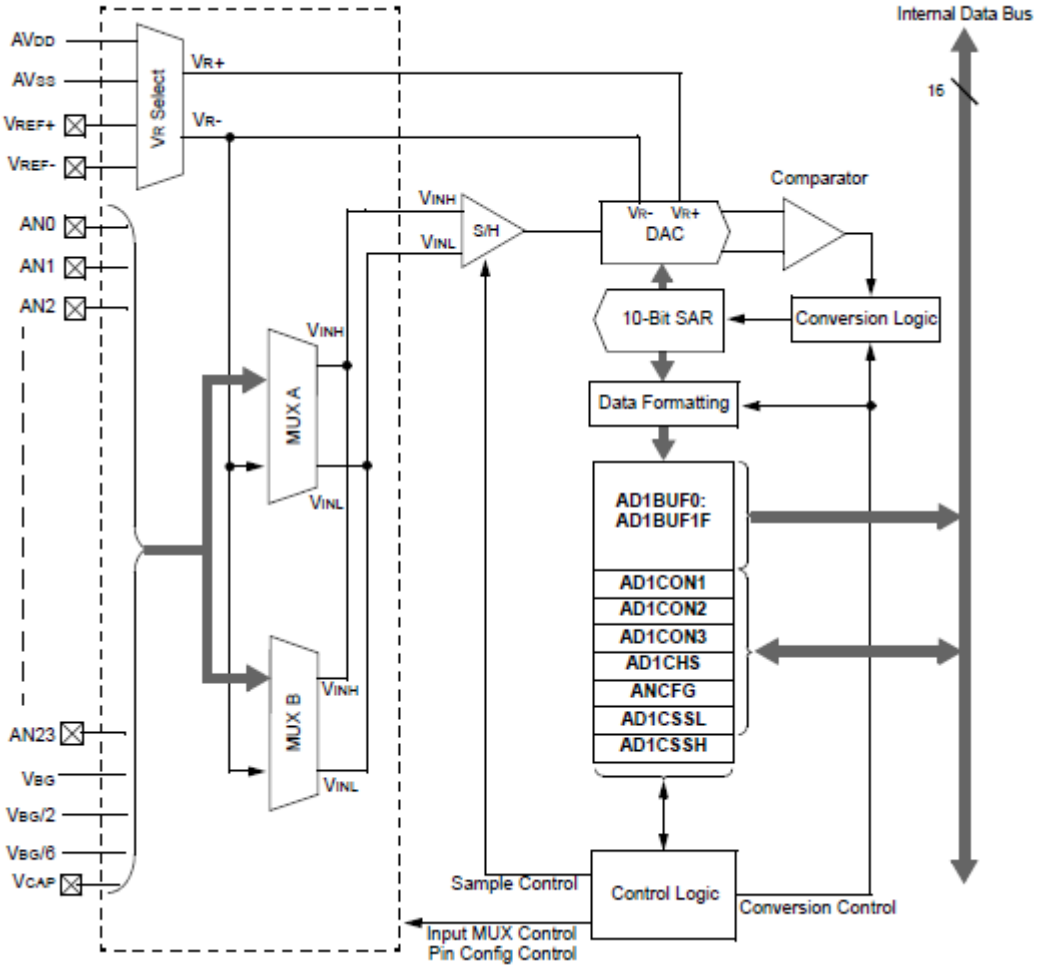


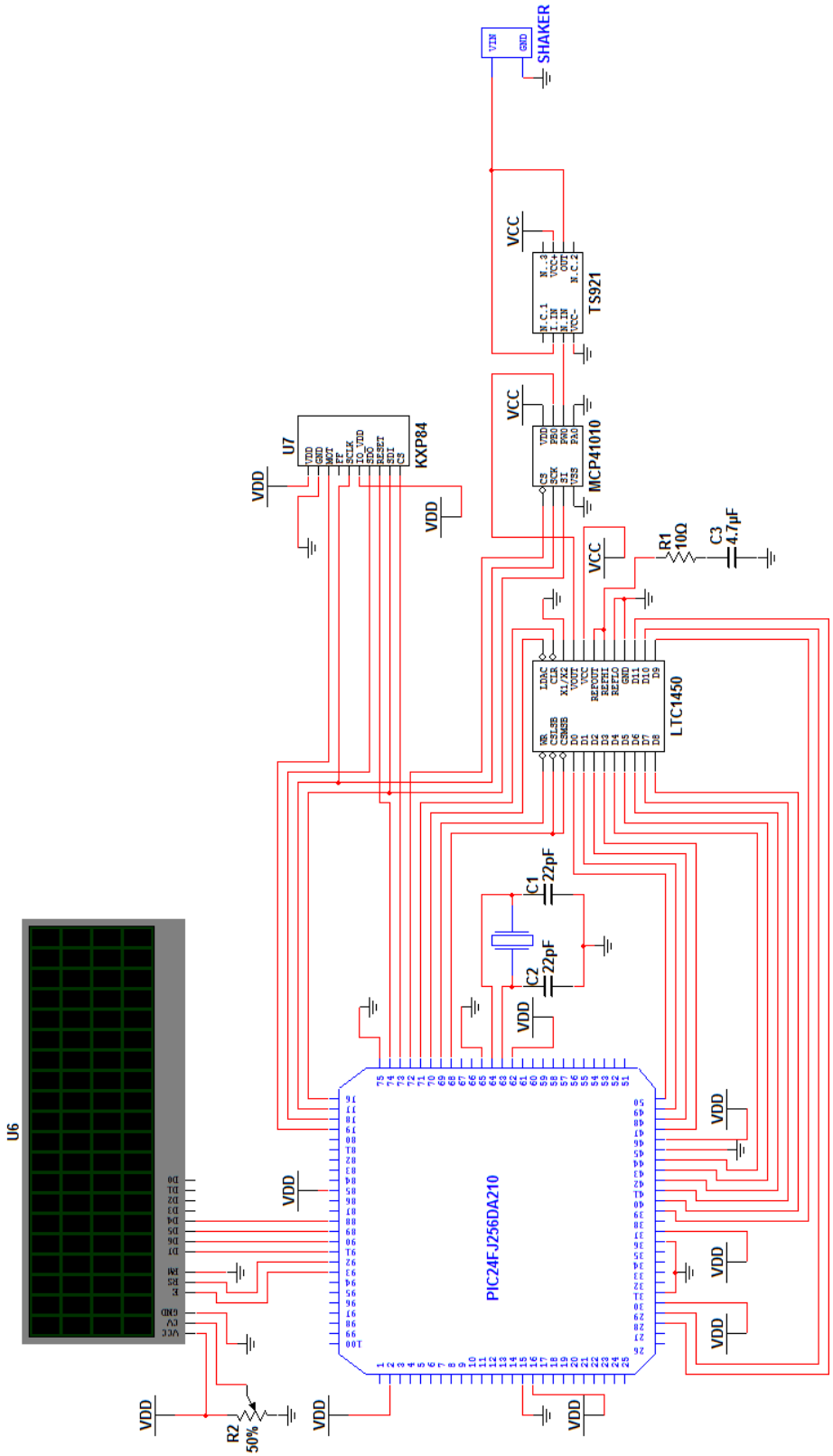
Figura 6.3: Modulo ADC interno al PIC16FJ256DA210

L'inizzializzazione del convertitore A/D avviene attraverso diversi registri che saranno esposti di seguito:

- **ANCFG** è il registro che controlla l'abilitazione delle tensioni di riferimento di band gap;
- **AD1SSL** è il registro attraverso il quale è possibile attivare la scansione dei pin d'ingresso da $AN0$ fino ad $AN15$. Il processo di scansione dei canali è una funzionalità offerta dai PIC24 ed è disponibile solo utilizzando il MUXA, se abilitato, una volta che si avvia la conversione A/D, tale conversione verrà effettuata su tutti i canali attivati, fino ad un massimo di 16 canali abilitati (in quanto il buffer d'uscita è formato da 16 word);
- **AD1SSH**, analogamente al registro precedente, attraverso il suddetto è possibile attivare la scansione dei pin d'ingresso da $AN16$ ad $AN24$ e ai riferimenti di band gap;
- **AD1CON1** è il primo registro di configurazione, esso consente di attivare la conversione, impostare qual'è levento che determina l'avvia del processo, esso può essere dato dal fine conteggio del timer, e permette la scelta del tipo di variabile in uscita dal convertitore.
- **AD1CON2** è il secondo registro di configurazione, utilizzato per la gestione degli interrupt generati dal modulo ADC, dalla scelta delle tensioni di riferimento positive e negative e, dalla scelta del multiplexer utilizzato per selezionare l'ingresso analogico del convertitore.
- **AD1CON3** è il terzo, ed ultimo,registro di configurazione, serve per selezionare la sorgente del clock da utilizzare per il modulo convertitore e per impostre i periodi di campionamento del S&H a monte ed il tempo di conversione dell'ADC.

L'acquisizione dei campioni della forma d'onda fornita in ingresso allo shaker viene anchessa gestita ad interrupt e, anche in questo caso, occorre fare in modo che i campioni siano tra loro poco distanziati. Difatti bisogna sempre rispettare il teorema del campionamento di Shannon.

Schema Circuitale



Conclusioni

Montando il circuito descritto in questo elaborato e rappresentato dallo schema in figura precedente è stato testato come il sistema di controllo a microcontrollore sia nettamente più preciso del preesistente sistema che si basava sul software *Labview* installato all'interno di un personal computer.
In particolare, per quanto riguarda le letture dell'accelerazione, il microcontrollore permette di raggiungere una frequenza di trasferimento dati attraverso l'interfaccia SPI di un megahertz, ossia il massimo supportabile dal componente KXP84. Mentre avvalendosi del precedente sistema, questa frequenza di trasferimento rimaneva sempre al di sotto del centinaio di kilohertz.
Siamo dunque ora in grado di effettuare letture più rapide che consentono di compiere i dovuti calcoli senza sorpassare il millisecondo.
Tra i vantaggi di questo sistema occorre sottolineare la possibilità di utilizzo in *stand-alone* rimuovendo così la dipendenza dal computer.
Il computer verrà utilizzato tramite un progetto parallelo ed integrativo a questo, per consentire all'utilizzatore di caricare delle forme d'onda personalizzabili, nonché ottenere l'andamento dell'accelerazione che caratterizza la vibrazione e della tensione fornita in ingresso allo shaker.
Inoltre, sempre rispetto al sistema preesistente, questo progetto introduce un controllo a catena chiusa essenziale per poter ripetere la caratterizzazione di un sensore piezoelettrico. È quindi tale controllo in retroazione a garantire la *condizione di ripetibilità* che è forse la specifica più importante. Infatti, prima era l'utilizzatore che agendo su un trimmer multigiri, riusciva ad imporre l'accelerazione voluta alle vibrazioni, naturalmente, in un secondo momento era impossibile riportare il sistema alle stesse condizioni e quindi risultava impossibile definire il comportamento del sensore piezoelettrico.
Concludendo, il progetto è riuscito a rispettare tutte le specifiche che erano state imposte senza dunque scendere a compromessi.

Elenco delle figure

1 Schema a blocchi del progetto 2

1.1 PIC24FJ256DA210 Development Board 4
1.2 PIC24FJ256DA210, diagramma dei pin 5
1.3 Peripheral Pin Select in ingresso, esempio effettuato su U1RX 6
1.4 Peripheral Pin Select in uscita, esempio effettuato su OC5 7
1.5 Interrupt Vector Table 9
1.6 Dettaglio sui traps vector 11
1.7 Schema a blocchi del Timer1 12

2.1 Funzione di trasferimento di un DAC a 3 bit 16
2.2 Scala di resistori di un convertitore D/A a 3 bit 17
2.3 Schema a blocchi del LTC1450 18
2.4 LTC1450, denominazione dei pin 19
2.5 LTC1450, diagramma temporale 20
2.6 Spettro di una vibrazione causata dal treno 21

3.1 Schema a blocchi di una comunicazione SPI 24
3.2 Comunicazione SPI mettendo in evidenza gli shift register 25
3.3 Schema a blocchi del modulo SPI interno ai PIC24F 26
3.4 Diagramma temporale della comunicazione SPI 28

4.1 Specifiche elettriche del KXP84 31
4.2 Diagramma funzionale del KXP84 31
4.3 Connessione in parallelo di più KXP84 33
4.4 Diagramma temporale per la scrittura di un registro di controllo del KXP4 34
4.5 Diagramma temporale per la lettura di un registro di controllo del KXP4 34
4.6 Diagramma temporale del processo di conversione analogico - digitale interna al KXP4 34
4.7 Indirizzi dei 13 registri interni al KXP84 35
4.8 Diagramma a blocchi dell'algoritmo di calcolo dell'accelerazione efficace 38

5.1 Rappresentazione di un potenziometro 41
5.2 Rete resistiva di un potenziometro ad 8 bit 43
5.3 Diagramma dei pin del MCP1010 nel package PDIP 43
5.4 Diagramma a blocchi del MCP1010 44
5.5 Command Byte del MCP1010 45
5.6 Partitore resistivo 46
5.7 Schema a blocchi del controllo in retroazione 47

6.1 Processo di conversione A/D visto come cascata di due sottoprocessi . 49
6.2 Schema a blocchi del convertitore ad approsimazioni sucessive 51
6.3 Modulo ADC interno al PIC16FJ256DA210 53

Riferimenti bibliografici

- Shashank Priya, Daniel J. Inman, *Energy Harvesting Technologies*, Springer, 2008
- Microchip, *PIC24FJ256DA210 Family Data Sheet*, www.microchip.com
- Microchip, *PIC24FJ256DA210 Development Board User's Guide*, www.microchip.com
- Microchip, *PIC24F Family Reference Manual, Sect. 12 I/O Ports with Peripheral Pin Select*, www.microchip.com
- Microchip, *PIC24F Family Reference Manual, Sect. 08 Interrupts*, www.microchip.com
- Microchip, *PIC24F Family Reference Manual, Sect. 14 Timers*, www.microchip.com
- Gaetano Iuculano, Domenico Mirri, *Misure Elettroniche*, Cedam, 2004
- Tertulien Ndjountche, *CMOS Analog Integrated Circuits: High-Speed and Power-Efficient Design*, CRC Press, 2011
- Linear Technology, *Datasheet of LTC1450* , www.linear.com
- Lucio Di Jasio, *Programming 16-Bit PIC Microcontrollers in C: Learning to Fly the PIC 24*, Newnes - Embedded Technology Series™, 2007
- Microchip, *PIC24F Family Reference Manual, Sect. 23 Serial Peripheral Interface (SPI)*, www.microchip.com
- Mauro Laurenti, *Tutorial - L'interfaccia SPI*, www.laurtec.com
- Kionix, *Datasheet of KXP84*, www.kionix.com
- Jack Smith, *Programming the PIC Microcontroller with MBASIC*, Newnes - Embedded Technology Series™, 2005
- Microchip, *AN1080 - Understanding Digital Potentiometers Resistor Variations*, www.microchip.com
- Microchip, *Datasheet of MCP41010*, www.microchip.com
- Microchip, *PIC24F Family Reference Manual, Sect. 17 10-Bit A/D Converter*, www.microchip.com