# Politecnico di Milano

V Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Elettronica

Dipartimento di Elettronica e Informazione



## *Misuratore laser a triangolazione a banda larga*


Relatore:
Prof. Michele Norgia

Tesi di Laurea di:
Reuf Qereshniku
Matr. N°755479



Anno Accademico 2011-2012

# Indice

# INTRODUZIONE

Le vibrazioni in ambito meccanico possono rendere più breve la vita di un pezzo o di un macchinario, possono essere dannose per l'utente e generalmente generano inquinamento acustico. Inoltre, nelle produzioni possono rendere inutilizzabili prodotti realizzati, a causa di queste, non secondo le specifiche.

Effettuare misure di frequenza, di ampiezza o comunque di grandezze a esse connesse è molto utile per tentare di capire la causa della vibrazione e quindi intervenire por attenuarla o se possibilie sopprimerla. Per questo in commercio esistono diversi tipi di misuratori di distanza con un elevata risoluzione destinati proprio alla diagnostica dei macchinari.

In questa tesi viene descritta la realizzazione e la progettazione di un misuratore di distanze basato sulla tecnica della triangolazione LASER, tecnica che permette l'utilizzo di componeni elettronici a banda inferiore rispetto ad altre tecniche (come quella interferometrica) che comporta un minore costo e un più semplice utilizzo. Questo tipo di strumento possiede il vantaggio di effettuare la misura senza avere il contatto fisico con l'oggetto, lasciando cosi il sistema imperturbato e ottenendo risultati ancor più precisi.

Per ottenere prestazioni più alte rispetto ai normali vibrometri a triangolazione si è deciso di sfruttare i vantaggi che il campionamento col seguente processing digitale comporta nell'elaborazione dei segnali utilizzando un DSP.

La realizzazione dello strumento è stata effettuata nel Laboratorio di Misure a Dipartimento di Elettronica e Informazione del Politecnico di Milano in seguito alla necessità di realizzazione di un vibrometro per le misure di eccentricità di una turbina.

Il sistema doveva essere in grado di misurare l’eccentricità di un rotore con accuratezza micrometrica, mentre il rotore stesso era in fase di verifica su supporto, come illustrato in **Figura 0.1**. Lo scopo del lavoro era minimizzare i tempi di misura e i rischi di danneggiamenti del sistema si misura e, soprattutto, della turbina.

Misuratore laser a triangolazione a banda larga

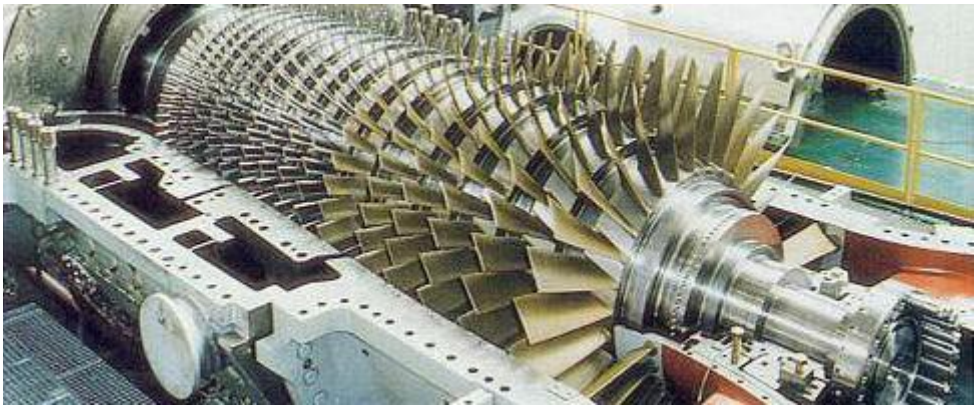


**Figura 0.1** Turbina in fase di verifica

Il progetto poi però ha preso strada propria, non accontentandosi della semplice soddisfazione delle specifiche per la misurazione dell'eccentricità, ma tentando di ottenere le massime prestazioni possibili.

# CAPITOLO 1

# Vibrazioni

## 1.1 Onde meccaniche

Un'onda meccanica è la propagazione di una perturbazione in un mezzo (gassoso, liquido o solido).

Per formare un'onda meccanica servono

- una sorgente della perturbazione
- un mezzo che subisca la perturbazione
- una connessione tra la materia perturbata e quella adiacente che propaghi la perturbazione

Al passaggio di un'onda meccanica la materia subisce una deformazione elastica: le particelle che costituiscono il mezzo materiale si spostano rispetto alla loro posizione di riposo e vi ritornano quando l'ampiezza dell'onda diventa nulla.

Durante la propagazione di un'onda, lo spostamento, la velocità e l'energia meccanica di un elemento di massa o volume vengono trasmessi a quello adiacente. In questo modo le onde trasportano energia meccanica attraverso la materia.

Nelle onde longitudinali le particelle investite dall'onda subiscono spostamenti paralleli alla direzione di propagazione dell'onda.

Nelle onde trasversali le particelle investite dall'onda subiscono spostamenti ortogonali alla direzione di propagazione dell'onda.

Esistono anche onde nelle quali le particelle del mezzo subiscono spostamenti sia longitudinali che trasversali (come ad es. le onde marine).

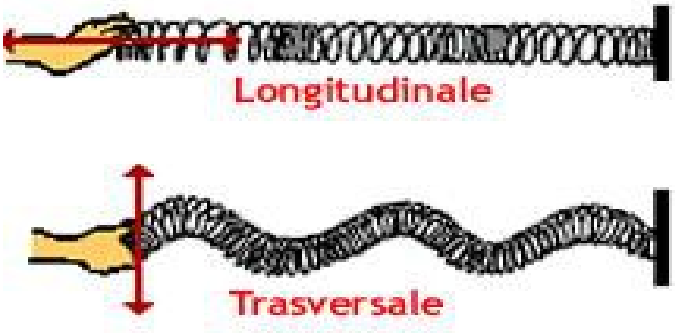



**Figura 1.1** Onda longitudinale e trasversale

Le onde possono propagarsi lungo una sola direzione (es. onda su una corda tesa), nel piano (es. onde provocate da un sasso lasciato cadere in uno stagno) o in tutto lo spazio (es. onde sonore generate da sorgenti puntiformi).

Quando le onde incontrano ostacoli o attraversano mezzi diversi subiscono riflessione e trasmissione.

Le onde possono essere descritte quantitativamente attraverso una funzione dello spazio e del tempo che assume la sua forma più semplice nel caso in cui la propagazione avviene in una sola direzione (onde unidimensionali). La generalizzazione della funzione d'onda al caso di onda tridimensionale è comunque immediata (onde sferiche).

Nelleonde sinusoidali l'ampiezza dell'onda varia in modo sinusoidale nel tempo e nello spazio. Questo tipo di onde riveste un'importanza particolare perché ogni altra forma d'onda periodica può essere ottenuta come sovrapposizione di onde sinusoidali (serie di Fourier).

Una caratteristica delle onde meccaniche è la loro velocità di fase, ossia la velocità alla quale si sposta un punto dell'onda di ampiezza fissata (ad esempio la cresta dell'onda). La velocità di fase dipende dal mezzo in cui si propaga l'onda. La velocità di un'onda sonora

in aria a 20°C è circa 340 m/s, mentre la stessa onda si propaga in acqua alla velocità di 1480 m/s. La velocità di propagazione di un'onda su una corda è direttamente proporzionale alla radice della tensione e inversamente proporzionale alla densità lineare di massa della corda. In generale, la potenza trassporata da un onda è proporzionale alla velocità dell'onda e al quadrato della sua ampiezza massima.

Se più onde si propagano in un mezzo le ampiezze delle onde si sommano (principio di sovrapposizione). La sovrapposizione di due onde di uguale frequenza può dar luogo ad interferenza costruttiva o distruttiva.

Quando due onde sinusoidali di uguale frequenza si propagano l'una verso l'altra sovrapponendosi si formano onde stazionarie. L'ampiezza delle onde stazionarie varia periodicamente nello spazio e in alcuni punti detti nodi è sempre nulla. Perciò le onde stazionarie non trasportano potenza.

In una corda fissata agli estremi o in un tubo pieno d'aria possono formarsi soltanto onde stazionarie con frequenze che sono tutte multiple di una frequenza caratteristica detta fondamentale. Gli strumenti musicali sono dispositivi predisposti per formare onde stazionarie di particolare frequenza (note) in corde vibranti e tubi pieni d'aria.

## 1.2 Vibrazioni

La vibrazione è un fenomeno caratterizzato da moti alternati di piccola ampiezza ed alta frequenza spesso sovrapposti al normale movimento cinematicodegli organi delle macchine.
Problemi generati dalle vibrazioni nelle macchine e negli impianti:

•rotture per fatica;
•impossibilità di mantenere le prestazioni di progetto;
•accoppiamento vibroacusticocon emissione di rumore;

•possibili effetti dannosi sull'uomo.

Le vibrazioni sono fenomeni determinati da trasferimenti di energia potenziale elastica in energia cinetica. Nascono generalmente a causa di forze perturbatrici che agiscono sul sistema meccanico e la loro ampiezza dipende dalle proprietà elastiche del sistema (vibrazioni forzate). Il fenomeno si manifesta anche in assenza di forze esterne eccitatrici quando si perturba lo stato di moto (o di quiete) del sistema imponendo, ad esempio, condizioni iniziali non di equilibrio (vibrazioni libere).

Per avere un'idea più precisa del campo di frequenze e delle ampiezze caratteristiche che vengono generate da macchinari, da singoli componenti meccanici, da mezzi di trasporto ed anche da eventi naturali si riporta in **Figura 1.2** un normogramma frequentemente riportato in letteratura. Da tale diagramma possono essere ricavate, in maniera molto indicativa, le frequenze, le velocità e le accelerazioni caratteristiche di molti tipi di vibrazioni a cui siamo giornalmente esposti.

Dal punto di vista pratico, quando si parla di vibrazioni, si usa distinguere le vibrazioni propriamente dette dagli impulsi (o shock). Molti macchinari sono sottoposti a test per verificarne il comportamento sia di tipo vibratorio che di tipo impulsivo. In generale si parla di shockquando un macchinario è sottoposto ad una forma di eccitazione meccanica di durata relativamente breve. Viceversa si parla di vibrazioni, quando l'eccitazione è di lunga durata. Se le vibrazioni hanno proprietà statistiche (medie) che non variano nel tempo, queste vengono dette di tipo stazionario. Tuttavia anche quando le proprietà statistiche delle vibrazioni non consentano di classificarle come stazionarie, se la velocità con cui queste proprietà variano è relativamente bassa le tecniche e gli strumenti di misura possono essere molto simili (se non identici) a quelli che si utilizzano nel caso stazionario. In tutti i casi che non rientrano nelle casistiche precedenti si parla in generale di shock.

In effetti non esiste un criterio rigido che consenta di distinguere le vibrazioni dagli shock. Dal punto di vista pratico si definisce spesso shock una eccitazione di tipo transiente la cui durata è comparabile o inferiore al tempo di risposta (o di decadimento) della risposta all'impulso del sistema.

Tale definizione è comunque da ritenersi relativamente semplicistica in quanto, dal punto

di vista meccanico, sarebbe più corretto definire come eccitazione impulsiva o shock un fenomeno transitorio il cui contenuto in frequenza sia sufficientemente ampio, in relazione al sistema in analisi, da eccitare tutti i principali modi di vibrazione del sistema.

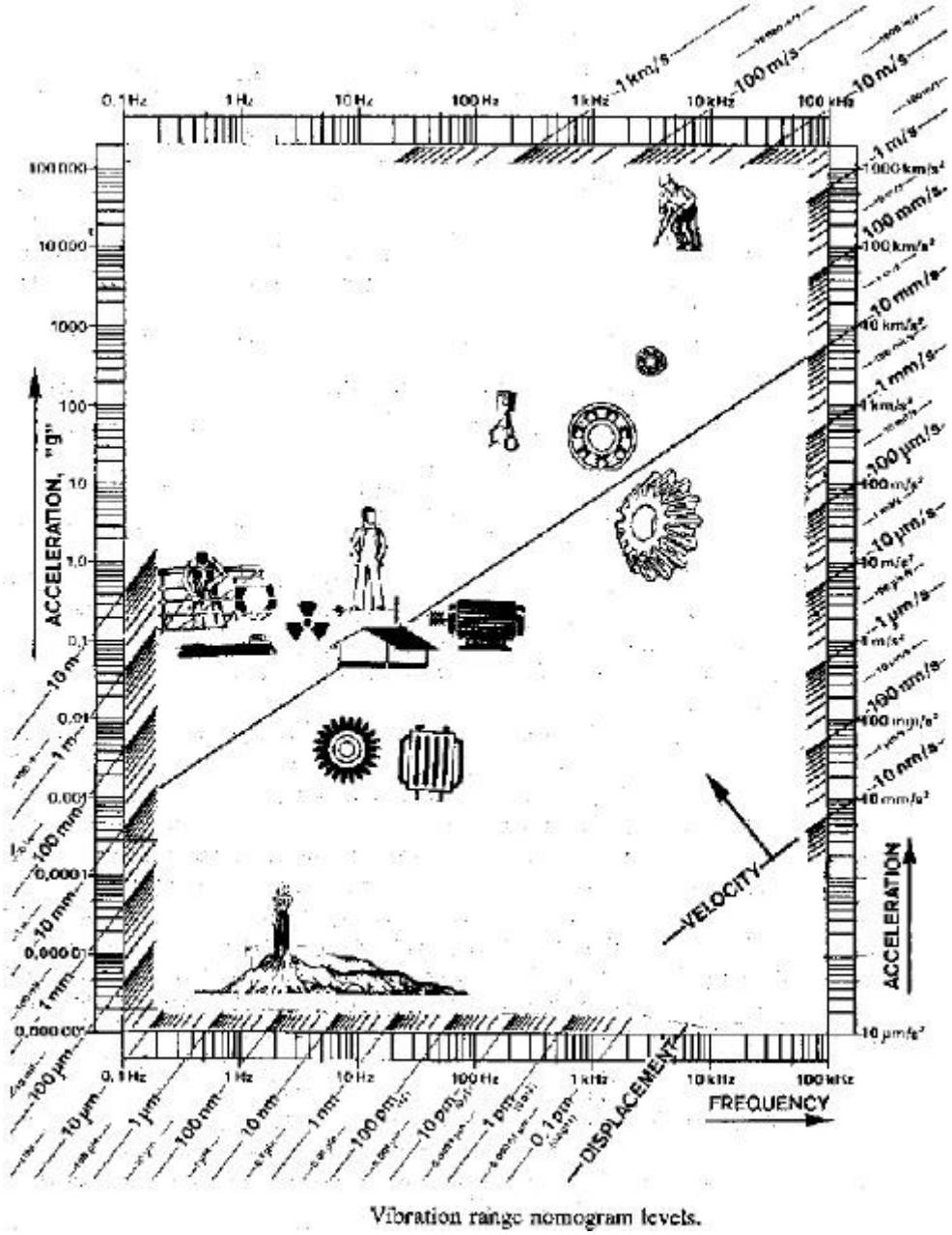



**Figura 1.2** Normogramma dei livelli di vibrazione

La classificazione del livello di vibrazione non è unica. Varia con il tipo di standard adottato, con il range di frequenza e con altri fattori. Neanche la grandezza di osservazione è univocamente accettata: i parametri che possono essere presi in considerazione sono l'ampiezza della vibrazione, la sua velocità o la sua accelerazione.

•Per le vibrazioni nel campo di frequenze tra 10 e 1000 Hz, la velocità di vibrazione è il parametro di gran lunga più utilizzato.

•Per vibrazioni di tipo puramente armonico, sia il valore di picco che il valore medio quadratico (definito nella pagina seguente) sono indicativi del livello di vibrazione.

•Per vibrazioni di tipo più complesso i due indici darebbero indicazioni discordanti (per velocità di rotazione da 600 a 12000 rpm-10÷200 Hz-solo il valore medio quadratico della velocità di vibrazione risulta indicativo del livello di vibrazione).

Per tale motivo la InternationalStandard Organisation(ISO) ha introdotto come parametro di misura del livello di vibrazione il **'vibration severity'**.

Tale parametro è definito come il più alto valore quadratico medio della velocità di vibrazione nel rangedi frequenza da 10 a 1000 Hz, misurato sul macchinario in punti prefissati.

Tali misure sono generalmente triassialiin corrispondenza dei supporti o del basamento.

Una volta effettuata la registrazione della velocità istantanea di vibrazione (naturalmente campionando il segnale) il suo valore quadratico medio può essere ricavato tramite la seguente formula:

$$v_{rms} = \sqrt{\frac{1}{T}\int_0^T v^2(t)\,dt} \qquad \textbf{(1.1}$$

Per vibrazioni di tipo armonico alla frequenza $\omega_i$, la velocità di vibrazione è esprimibile come:

$$v_i = \hat{v}_i \cos(\omega_i t) \qquad \textbf{(1.2}$$

Misuratore laser a triangolazione a banda larga

Lo standard ISO 2372 si riferisce a macchine rotanti con velocità tra i 10 e 200 rps caratterizzati da rotori sia rigidi che flessibili a condizione che le misure di vibrazioni in prossimità dei supporti siano indicative delle vibrazioni del rotore.

Il livello di vibrazione definito in precedenza si applica al rangedi frequenze comprese almeno tra il 30 ed il 300% della frequenza di rotazione del rotore (coprendo siale frequenze tipiche delle eccitazioni sincrone -sbilanciamento-che le loro principali armoniche e le principali asincrone -whirldel rotore).

**I componenti di tale tipo sono inoltre suddivisi in 4 classi:**

**Classe I**-Singoli componenti collegati integralmente al macchinario completo nelle normali condizioni operative (motori elettrici fino a 15 kW);

**Classe II**-Macchine di media taglia (motori elettrici da 15 a 75 kWe motori fino a 300 kWsu basamenti speciali )

**Classe III**-Motori di grandi dimensioni montati su basamenti pesanti e rigidi;

**Classe IV**-Motori di grandi dimensioni montati su basamenti relativamente elastici (flessibili) o strutture di tipo leggero.

Il livello di vibrazione è suddiviso in 4 intervalli classificati con le lettere da A (buono) a D (inaccettabile) in ordine crescente di importanza.

Il rangedi livello di vibrazioni deve essere scelto dall'utente sulla base di considerazioni che riguardano:

•il tipo e la taglia del macchinario;

•il tipo di servizio che deve assicurare;

•il tipo di basamento

•gli effetti che le vibrazioni possono provocare sul personale, sugli strumenti e sui macchinari vicini.

Criterio ISO per la classificazione del livello di vibrazione digrandi motori (classi III e IV) in funzione del tipo di supporti:

**Supporti rigidi:**la prima frequenza naturale del macchinario sui supporti è superiore alla principale frequenza dell'eccitazione;

**Supporti flessibili:**la prima frequenza naturale del macchinario sui supporti è inferiore alla principale frequenza dell'eccitazione.

| RMS velocity ranges of vibration severity | | Vibration severity for separate classes of machines | | | |
|---|---|---|---|---|---|
| mm/sec | in./sec | Class I | Class II | Class III | Class IV |
| 0.28 | 0.01 | | | | |
| 0.45 | 0.02 | A | | | |
| 0.71 | 0.03 | | A | | |
| 1.12 | 0.04 | B | | A | |
| 1.8 | 0.07 | | B | | A |
| 2.8 | 0.11 | C | | B | |
| 4.5 | 0.18 | | C | | B |
| 7.1 | 0.28 | D | | C | |
| 11.2 | 0.44 | | D | | C |
| 18 | 0.71 | | | D | |
| 28 | 1.10 | | | | D |
| 45 | 1.77 | | | | |

**Tabella 1.1** Criteri per la classificazione del livello di vibrazione

| rms velocity vibration severity | | Support classification | |
|---|---|---|---|
| mm/sec | in./sec | Rigid supports | Flexible supports |
| 0.46 | 0.018 | | |
| 0.71 | 0.028 | Good | |
| 1.12 | 0.044 | | Good |
| 1.8 | 0.071 | | |
| 2.8 | 0.11 | Satisfactory | |
| 4.6 | 0.18 | | Satisfactory |
| 7.1 | 0.28 | Unsatisfactory | |
| 11.2 | 0.44 | | Unsatisfactory |
| 18.0 | 0.71 | | |
| 28.0 | 1.10 | Unacceptable | |
| 46.0 | 1.80 | | |
| 71.0 | 2.80 | | Unacceptable |

**Tabella 1.2** Giudizio sulla qualità del livello di vibrazione di un macchinario

## 1.3 Tecniche di analisi

La scelta dei punti di misura delle vibrazioni è un elemento essenziale al pari della scelta degli strumenti di misura, del criterio per la valutazione del livello di vibrazione del tipo di condizionamento a cui il segnale viene sottoposto.

I segnali che vengono registrati durante il funzionamento dei motori, in prossimità dei cuscinetti oppure sulla scatola dei riduttori contengono una notevole quantità di informazioni.

Se si danneggia un pistone, un cuscinetto od un albero, è evidente che le vibrazioni generate durante il funzionamento subiscono dei mutamenti. Tuttavia, dall'analisi nel dominio del tempo dei segnali di spostamento, velocità ed accelerazione, spesso tali mutamenti sono mascherati dalla presenza di molte altre componenti. La semplice misura del livello di vibrazione globale può quindi trarre in inganno.

Le tecniche dell'analisi in frequenza sono state sviluppate per evidenziare in maniera selettiva quella parte del segnale che si ritiene possa essere sintomo inequivocabile di un malfunzionamento. Tali tecniche non permettono solo di verificare delle ipotesi sulla genesi dei fenomeni vibratori ma, in molti casi, sono in grado di indirizzare la ricerca di danni di cui non si conosce a priori l'origine.

Una volta individuate le posizioni di misura e, di conseguenza, il tipo dei sensori da utilizzare si procede con la misura (e lamemorizzazzione) dei segnali come funzione del tempo. I segnali, in uscita degli strumenti di misura, sono spesso assai poco indicativi dello stato di funzionamento del macchinario e necessitano di essere processati in maniera opportuna. Una delle prime operazioni a cui il segnale deve essere sottoposto è lafiltratura(weighting). Naturalmente tale operazione è successiva allafiltratura anti-aliasingche va effettuato sul segnale analogico. Con le operazioni difiltraturasi intende eliminare alcune parti del segnale di vibrazione che non sono direttamente in relazione con il fenomeno di interesse.Se, ad esempio, si intende analizzare le vibrazioni causate dallo sbilanciamento di un rotore rigido, i contributi ad alta frequenza dovuti al rumore o ad altri fenomeni di disturbo sono indesiderati. Poiché quindi si conosce la frequenza delle

vibrazioni direttamente dipendenti dallo sbilanciamento (sono sincrone con il rotore) è quindi assai utile utilizzare un filtro passa-bandacentrato sulla frequenza di rotazione del rotore stesso.I filtri possono essere di tipo passa alto, passa basso o passa banda a seconda delle esigenze.

La filtratura del segnale temporale non è l'unica operazione che può essere eseguita su segnali di vibrazione. Per quanto riguarda le macchine rotanti, i riduttori e, in generale, tutte quelle macchine operanti a regime periodico (rotanti ed alternative) è assai utile eseguire delle medie temporali sui segnali di vibrazione registrati in ogni periodo. La possibilità di eseguire correttamente l'operazione di media è legata alla possibilità diselezionare con estrema precisione all'interno della finestra temporale di misura quei dati che si riferiscono ad un singolo periodo. E' quindi necessario un tachimetro in grado di individuare con esattezza la posizione del rotore (macchine rotanti) o dell'albero a gomiti (macchine alternative).I risultati della semplice operazione di media possono essere sorprendenti.

Spesso le tecniche di media vengono applicate nel dominio della frequenza come nel dominio del tempo. Tuttavia le due tecniche sono molto diverse dal punto di vista teorico e pratico e non vanno confuse. Le tecniche di media nel tempo sono adottabili solo permacchine rotanti o alternative e le misure necessitano di essere sincronizzate tramite un segnale ditrigger per isolare un singolo ciclo o una determinata parte di esso.Le medie nel dominio delle frequenze sono utilizzate anche nelle macchine a flusso continuo e per l'analisi modale, quando si tenta di identificare le frequenze tramite test con eccitazione ad impulso.La presenza di disturbi di tipo casuale, o dovuti ad una non corretta applicazione dell'eccitazione impulsiva possono determinare dei picchi nelle singole frequenze che non sono dovuti ad alcuna risonanza.

# CAPITOLO 2

# Sezione ottica

## 2.1 La triangolazione

La triangolazione è una tecnica che permette di calcolare distanze fra punti sfruttando le proprietà dei triangoli. Questa tecnica molto antica è stata usata in topografia, in astronomia e perfino i navigatori si orientavano con strumenti e tecniche trigonometriche.
In principio era l'osservatore che attraverso la propria vista doveva puntare l'oggetto da misurare ed effettuare la misurazione a mano. Oggi, grazie all'ausilio delle svariate proprietà dei materiali semiconduttori e dei laser è possibile sfruttare la tecnica di triangolazione in modo del tutto indipendente dall'uomo.

### 2.1.1 Il triangolatore ottico passivo

Il triangolatore ottico passivo è l'esempio di partenza per dimostrare come sia possibile misurare una distanza utilizzando strumenti ottici, è costituito da lenti e da due specchi, come si può vedere nella **Figura 2.1:**

La distanza  dell'oggetto è ricavata, o meglio, triangolata conoscendo la distanza tra due punti A e B, in cui sono posizionati due specchi, uno fisso chiamato, beamsplitter, l'altro libero di ruotare attorno a B chiamato M. Il lato AB è posto ortogonalmente alla direzione di osservazione e ruotando lo specchio M si fa coincidere l'immagine riflessa dell'oggetto con l'immagine diretta, quando esse coincidono sarà possibile leggere l'angolo , che sarà l'esatta riproduzione dell'angolo nel vertice.

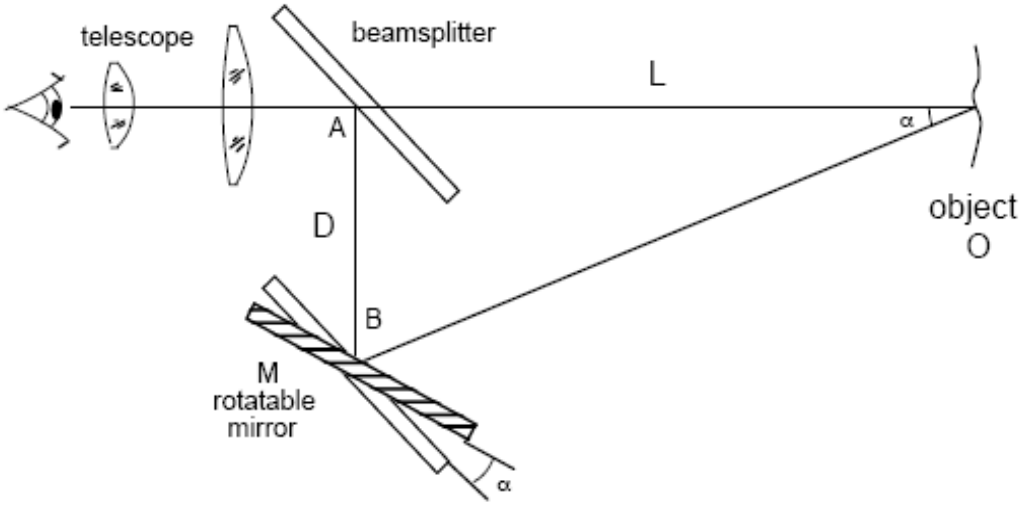



**Figura 2.1**: Il triangolatore ottico passivo

Una volta ottenuto l'angolo sarà possibile determinare la distanza, tramite la formula :

$$L = \frac{D}{tan\alpha} \qquad \textbf{(2.1}$$

Quando le distanze da misurare sono grandi, l'angolo diventa molto piccolo quindi, si può applicare una semplificazione alla formula attraverso l'approssimazione per piccoli angoli:

$$L = \frac{D}{tan\alpha} \simeq \frac{L}{\alpha} \qquad \textbf{(2.2}$$

Tuttavia questo tipo di misura si dimostra poco accurato se si misurano lunghe distanze, ovvero quando L diventa molto più grande di D perché, aumenta l'incertezza di misura sull'angolo che ormai è diventato molto piccolo.
L'errore di misura assoluto  e quello relativo , dovuti all'errore angolare , si sono espressi dalla relazione:

$$\Delta L = \frac{-D}{\alpha^2} \cdot \Delta\alpha = \frac{-L^2}{D} \cdot \Delta\alpha = k \cdot \Delta\alpha \qquad \textbf{(2.3}$$

$$\frac{\Delta L}{L} = \frac{-L}{D} \cdot \Delta\alpha = \frac{-\Delta\alpha}{\alpha} \qquad \textbf{(2.4}$$

dove *k* è definita sensibilità di *L* rispetto ad *α*.
L'accuratezza e la risoluzione di misura della distanza quindi, dipende dall'accuratezza e dalla risoluzione con cui si determina il valore dell'angolo α. Con una vite micrometrica si può ottenere un Δα = 3 mrad mentre, con l'utilizzo di un encoder angolare si può raggiungere un Δα = 0.1 mrad.

### 2.1.2 Il triangolatore ottico attivo

L'evoluzione del triangolatore ottico passivo, dove non è presente nessuna sorgente di luce e nessun rivelatore, è uno strumento dove è presente un emettitore di luce, possibilmente coerente, come ad esempio un laser, ed un dispositivo fotosensibile in grado di intercettare il fascio luminoso dell'emettitore con precisione. Lo strumento con i particolari appena descritti prende il nome di triangolatore ottico attivo e rispetto a quello passivo presenta il

vantaggio di non avere parti in movimento quindi, ci si può già aspettare che aumenterà la precisione, la rapidità e la ripetibilità delle misure.

La luce del fascio laser che colpisce il bersaglio viene in parte riflessa ed in parte diffusa, a seconda del tipo di superficie che incontra. Materiali metallici o a specchio rifletteranno la quasi totalità della luce, mentre materiali opachi aumenteranno il fenomeno di diffusione, anche il colore della superficie colpita influisce sulla quantità di luce che verrà diffusa. Per garantire la massima diffusione della luce il bersaglio dovrà essere di colore bianco opaco.

La luce diffusa dal bersaglio verrà raccolta da una lente che ha il compito di collimare e mettere a fuoco lo spot del laser sull'area del fotorivelatore.

Esistono varie tipologie di fotorivelatori, come ad esempio i CCD, 2Q e PSD, tutti in grado di determinare su quale porzione della loro superficie incide lo spot luminoso. Conoscendo la coordinata dello spot è possibile ricavare geometricamente l'angolo  tra il fascio di andata e quello di ritorno.

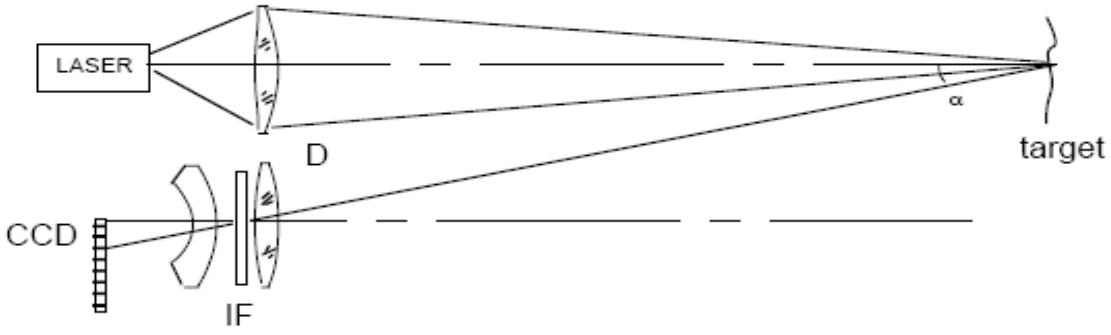



**Figura 2.2:** Schema funzionamento del triangolatore ottico attivo

La lunghezza d'onda del laser è stata scelta nel visibile per facilitare il puntamento del bersaglio e per ottenere buoni risultati sono sufficienti pochi mW di potenza emessa. La sezione del fascio ha forma ellittica di dimensione *1 x 3* μm in campo vicino. Una lente anamorfa rende la sezione del fascio circolare con dimensione $w_1$ di circa 5 μm e proietta lo spot sul bersaglio.

Sul bersaglio lo spot cambia di dimensione e diventa pari a:

$$w_1 \cdot \frac{L}{F_{ill}} \qquad \textbf{(2.5}$$

Parte della luce diffusa che giunge dal bersaglio viene raccolta dalla lente di lunghezza focale $F_{rec}$ e proiettata sul fotorivelatore posto nel piano focale della lente. Per una buona qualità della misura, è importante che sul fotorivelatore giunga possibilmente soltanto la luce del laser proveniente dal bersaglio e che sia in qualche modo schermato dalle altre componenti di luce ambientale.

Le dimensioni dello spot laser sul fotorivelatore sono calcolabili mediante la relazione:

$$w_{rec} = w_1 \cdot \frac{L}{F_{ill}} \cdot \frac{F_{rec}}{L} = w_1 \cdot \frac{F_{rec}}{F_{ill}} \qquad \textbf{(2.6}$$

Indicando con $x$ la distanza dello spot sul fotorivelatore dall'asse ottico della lente di ricezione, dalla figura si può notare come ad una variazione L + ΔL corrisponde una variazione $\alpha$ - $\Delta\alpha$ e $x$ - $\Delta x$.

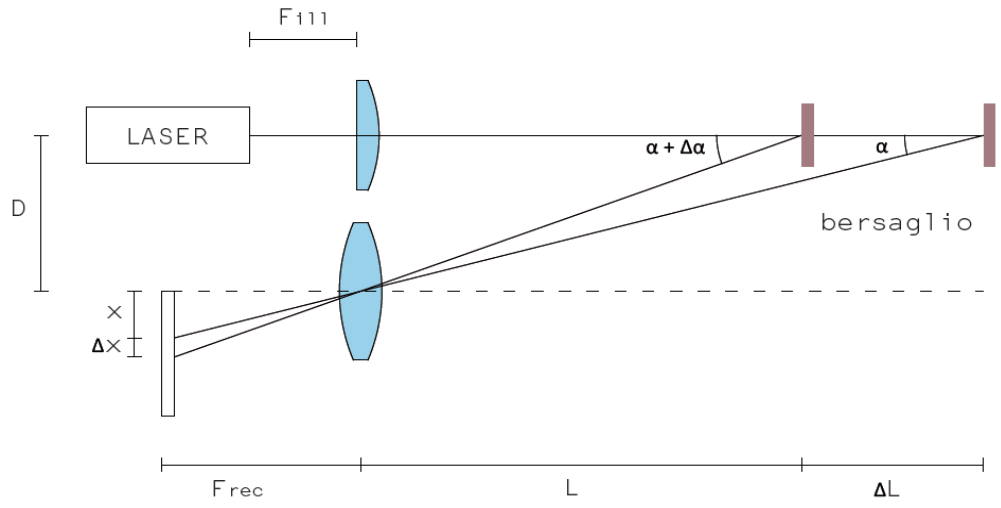



**Figura 2.3:** Rappresentazione di come avviene la conversione dell'angolo α in coordinata $x$, effettuata dalla lente di ricezione

La misura di L ricavata dalla posizione *x* sul fotorivelatore è data dalla relazione :

$$L = \frac{D}{x} \cdot F_{rec} \qquad \textbf{(2.7}$$

Differenziando in L e in *x* si ottiene l'errore di misura assoluto ΔL dovuto al minimo spostamento misurabile Δx sul rivelatore:

$$\Delta L = \frac{-D}{x^2} \cdot F_{rec} \cdot \Delta x \qquad \textbf{(2.8}$$

Ed infine, come il triangolatore ottico passivo si ottiene un errore di misura relativo ΔL/L pari a:

$$\frac{-\Delta L}{L} = \frac{-\Delta x}{x} = \frac{-\Delta \alpha}{\alpha} \qquad \textbf{(2.9}$$

### 2.1.3 Il triangolatore ottico attivo lineare

Come si può notare dalla relazione il triangolatore ottico attivo, con laser e fotorivelatore posti sullo stesso piano, presentano una caratteristica di ingresso-uscita di tipo non lineare e ciò, è dovuto al fatto che il bersaglio si sposta in una direzione ortogonale rispetto alla direzione dello spostamento *x* che compie lo spot laser ricostruito dalla lente sul fotorivelatore.

Nel caso di lenti convergenti, si può schematizzare il funzionamento della lente ricevente come un oggetto ottico in grado di focalizzare e ricostruire l'immagine dello spot laser sul bersaglio, sull'altro lato della lente ad una distanza $S_2$, secondo la legge:

$$\frac{1}{S_1}+\frac{1}{S_2}=\frac{1}{f} \qquad \textbf{(2.10}$$

dove $f$ è la lunghezza focale della lente.

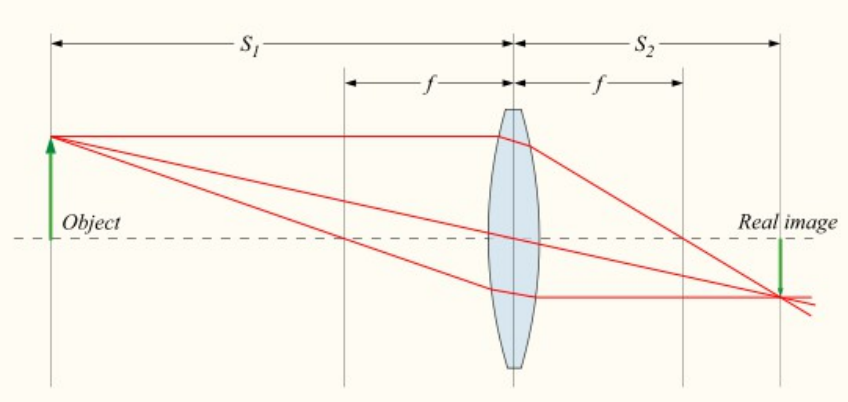



**Figura 2.4:** Formazione delle immagini in una lente positiva

Si può subito dedurre che, per quanto riguarda la nostra applicazione, risulta vantaggioso disporre la lente con un'inclinazione tale da far passare l'asse della lente per il centro dello spot laser sul bersaglio. In questo modo anche l'immagine ricostruita giacerà sull'asse della lente e ciò risulta comodo sia per posizionare il fotorilevatore esattamente al centro, sia per ottenere che le proiezioni delle immagini costruite siano più o meno ortogonali alla superficie del fotorilevatore e che quindi, non vengano distorte. Cosa che invece accade quando l'immagine dello spot laser ricostruito colpisce una superficie con una certa inclinazione.

Dalla considerazione di inclinare il fotorivelatore è nata l'idea di inclinare anche il laser. In questo modo, il raggio non colpisce il bersaglio sempre nello stesso punto ma, spostandosi su di esso in due direzioni e non più solo in una, modifica anche il relativo moto del puntino ricostruito sul fotorivelatore.

Di conseguenza si deduce subito che, quando l'inclinazione del fascio laser e l'inclinazione del fotorilevatore sono uguali il legame è perfettamente lineare,poiché le direzioni di spostamento del laser sul bersaglio e sul fotorivelatore sono parallele e ad ogni spostamento del bersaglio corrisponde un rispettivo spostamento sul fotorilevatore,

semplicemente ridotto per il rapporto di ingrandimento che sta applicando la lente. Per la misura di distanze piccole si può considerare che il rapporto di ingrandimento si mantenga costante e che quindi non introduca altre non linearità nel modello.

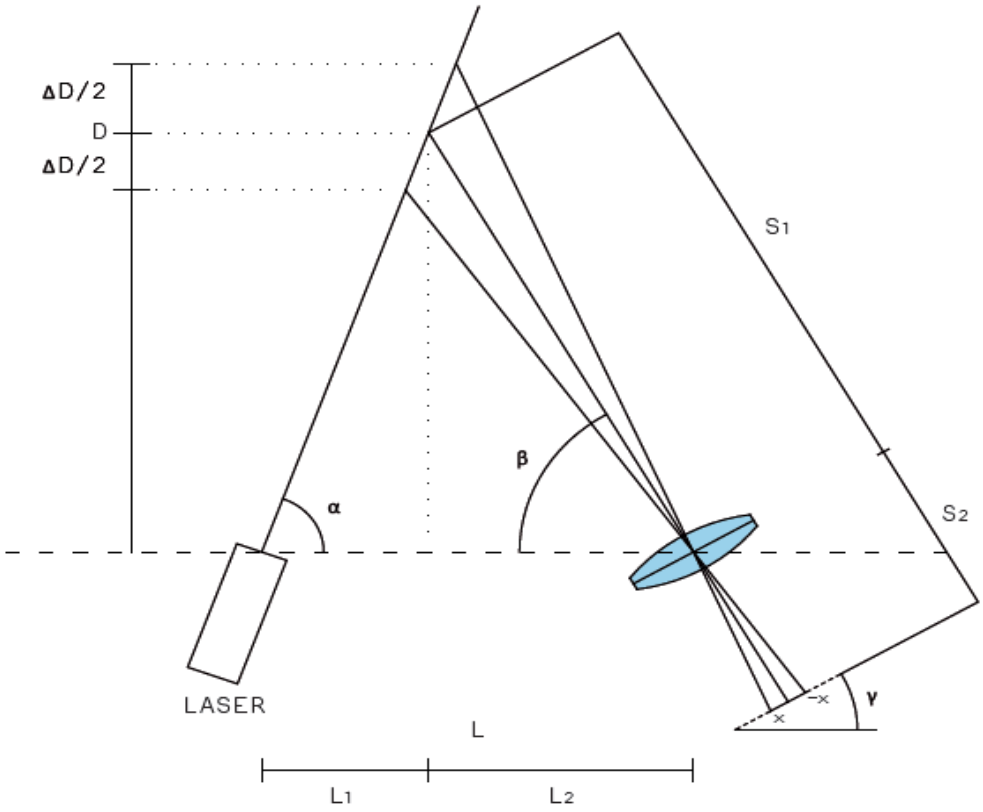



**Figura 2.5:** Rappresentazione geometrica del triangolatore ottico attivo che consente una risposta lineare

Per una generica costruzione geometrica mostrata nella figura si può descrivere il legame geometrico mostrato dai legami trigonometrici:

$$D = L_1 \tan(\alpha) \qquad \textbf{(2.11}$$

$$D = L_1 \tan(\beta) \qquad \textbf{(2.12}$$

da cui con semplici passaggi e considerando che $L = L_1 + L_2$ si può ricavare che:

$$D = L\frac{\tan(\alpha)\cdot\tan(\beta)}{\tan(\alpha)+\tan(\beta)} = L\frac{1}{\frac{1}{\tan(\alpha)}+\frac{1}{\tan(\beta)}} \quad \textbf{(2.13}$$

Quindi dalla relazione conoscendo l'ampiezza dei due angoli è possibile conoscere la distanza D del bersaglio. L'angolo α è l'angolo con cui è inclinato il laser quindi, lo si può considerare un parametro costruttivo che rimane costante una volta assegnato.
L'angolo β invece, varia a seconda dello spostamento del bersaglio e può essere ricavato conoscendo la posizione *x*, dello spot laser sulla superficie del fotorivelatore, sfruttando le proprietà dei triangoli rettangoli, attraverso la relazione:

$$\tan(\beta+\gamma) = \frac{S_2}{x} \quad \textbf{(2.14}$$

Applicando il teorema di addizione per la tangente è possibile scomporre la relazione e ricavare il valore dell'angolo β in funzione della posizione *x* e pervenire quindi alla:

$$\tan(\beta) = \frac{S_2 - x\tan(\gamma)}{x + S_2\tan(\gamma)} \quad \textbf{(2.15}$$

dove γ è l'angolo con cui è inclinato il fotorivelatore.
Sostituendo, si arriva ad esprimere la distanza D in funzione della posizione *x*, descritta dalla relazione:

$$D = L\frac{\tan(\alpha)(S_2 - x\tan(\gamma))}{x(\tan(\alpha)-\tan(\gamma)) + S_2\tan(\alpha)\tan(\gamma) + S_2} \quad \textbf{(2.16}$$

Da notare che nella relazione compare il termine *x* sia a numeratore che a denominatore quindi, per ottenere un legame con la distanza D il più lineare possibile bisogna fare in modo che il coefficiente (differenza tra tangenti), che moltiplica la *x* a denominatore tenda a zero. Quando laser e fotorivelatore sono inclinati con lo stesso angolo la caratteristica è completamente lineare.

Per mezzo di uno script in Matlab è possibile simulare e tracciare l'andamento della misura in funzione dei parametri costruttivi del triangolatore, quali:

- distanza a cui si vuol tenere centrato il bersaglio;
- lunghezza dell'area sensibile del fotorivelatore;
- lunghezza focale della lente utilizzata;
- inclinazione del laser (angolo $\alpha$);
- a scelta angolo $\beta$ nella condizione di bersaglio centrato oppure, la lunghezza $L_2$ a cui si trova la lente.

## 2.2 Il PSD

Come sensore da posizionare nel fuoco della lente si è scelto di utilizzare un PSD (Position Sensitive Detector) che, è un particolare tipo di fotodiodo che consente di distinguere la posizione di un punto luminoso che incide sulla sua area sensibile.
Il fotodiodo in questione è tecnicamente chiamato "*PSD ad effetto laterale*" e rappresenta l'evoluzione dei fotodiodi a quattro quadranti, anche chiamati in termini anglosassoni "*segmentedPSD's*".

### 2.2.1 Principio di funzionamento

I PSD ad effetto laterale hanno sostanzialmente la stessa struttura costruttiva di un diodo *pin* ed essendo sensibili alla radiazione luminosa sono in grado di generare delle correnti per effetto fotoelettrico, quando vengono colpiti dalla luce. La superficie fotosensibile è uno strato di silicio drogato, di tipo *P* ed è divisa dallo strato drogato tipo *N* da una zona di semiconduttore intrinseco. Per aumentare la resistenza serie delle regioni P ed N gli strati sono molto sottili e altamente drogati.
Nell'applicazione qui trattata siccome si intende misurare lo spostamento in una sola direzione verrà usato un PSD monodimensionale che, presenta in totale tre elettrodi, i primi due, sono degli anodi e si presentano come due depositi metallici posti sulle due estremità laterali del fotodiodo a contatto con la superficie fotosensibile mentre l'ultimo, il catodo, è uno strato metallico, depositato sull'intera superficie posteriore drogata tipo *N*.
Per schermare dalla luce l'area non attiva della parte superiore del sensore, questa viene ricoperta da uno strato di ossido di silicio.
Quando un raggio luminoso colpisce la superficie del PSD, la resistenza di tipo serie si ripartisce in due resistenze di valore proporzionale alla posizione in cui il raggio incide. Fisicamente, il raggio luminoso è in grado di far generare all'interno del semiconduttore una zona di svuotamento che, rende possibile il passaggio di elettroni dal catodo verso gli anodi quando, il fotodiodo è polarizzato inversamente, generando così delle foto-correnti,

anch'esse proporzionali alla posizione del fascio.

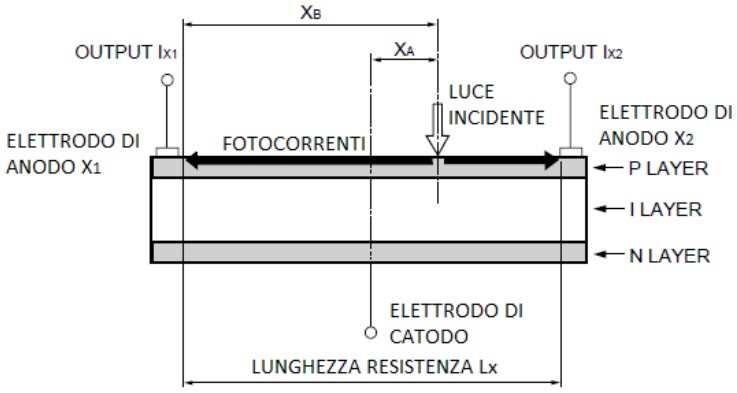



**Figura 2.6:** Sezione del PSD e schema di funzionamento

Tra i vantaggi offerti dal PSD sono da notare la linearità di risposta che è garantita sull'80% dell'area sensibile, con errori di linearità contenuti entro lo 0.5%, e la quasi indipendenza dalla dimensione dello spot luminoso, perché la zona di svuotamento si genera in prossimità del centroide del punto. Il tempo di risposta, che è limitato dalla capacità di giunzione, rende questi sensori ideali per applicazioni dove sono richieste elevate velocità di lettura, come nel nostro caso la misura di vibrazioni.

### 2.2.2 Caratteristiche elettriche

Un fotodiodo di silicio può essere rappresentato con un circuito equivalente, mostrato in **Figura 2.7** composto da un generatore di corrente in parallelo ad un diodo ideale. Il generatore di corrente rappresenta la corrente generata dalla radiazione incidente sulla superficie del PSD mentre, il diodo rappresenta la giunzione P-N. Si può aggiungere al modello anche una capacità di giunzione ($C_j$) e una resistenza di shunt ($R_{SH}$) sempre in parallelo alle altre componenti. Invece la resistenza serie ($R_S$) dello strato resistivo è posta

in serie a tutti i componenti del modello.

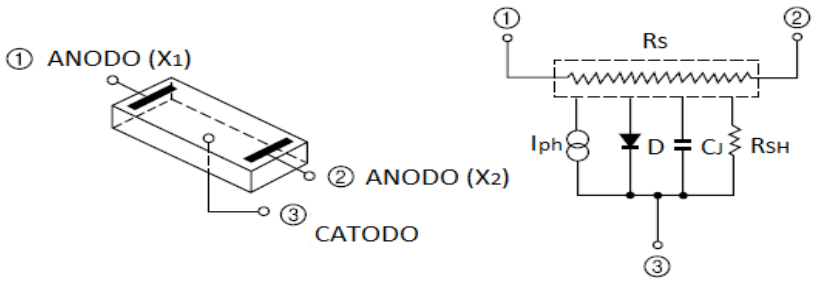



**Figura 2.7:** Circuito equivalente PSD

La resistenza di shunt rappresenta la pendenza nell'origine della curva caratteristica corrente-tensione del fotodiodo. Sebbene un fotodiodo dovrebbe avere una resistenza di shunt infinita, i valori attuali vanno da 10 a 1000 Mega Ohm. La resistenza di shunt viene usata per determinare la corrente di rumore quando al fotodiodo non viene applicata nessuna tensione inversa (modalità fotovoltaica).

Fotodiodi con alti valori di resistenza di shunt garantiscono quindi performances più elevate.

I confini della regione di deplezione si comportano come le superfici di un condensatore a facce parallele. La capacità di giunzione è direttamente proporzionale all'area diffusa ed inversamente proporzionale alla larghezza della regione di deplezione. Il valore della capacità di giunzione dipende principalmente dalla tensione inversa applicata agli elettrodi, come segue dalle seguenti relazioni:

$$C_j = \frac{\varepsilon_{Si}\varepsilon_0 A}{\sqrt{2\,\varepsilon_{Si}\varepsilon_0 \mu\,\rho\,(V_A + V_{bi})}} = A\sqrt{\frac{\varepsilon_{Si}\varepsilon_0}{2\,\mu\,\rho\,(V_A + V_{bi})}} = \frac{\varepsilon_{Si}\varepsilon_0 A}{W_d} \qquad \textbf{(2.17}$$

$$W_d = \sqrt{2\,\varepsilon_{Si}\varepsilon_0 \mu\,\rho\,(V_A + V_{bi})} \qquad \textbf{(2.18}$$

Dove $\varepsilon_0 = 8.854 \cdot 10^{-14}$ F/cm è la permittività elettrica del vuoto, $\varepsilon_{Si} = 11.9$ è la costante dielettrica del silicio, $\mu = 1400$ cm$^2$/Vs è la mobilità degli elettroni a 300 K, $\rho$ è la resistività del silicio, $V_{bi}$ è la tensione propria del silicio e $V_A$ è la tensione inversa applicata.

La figura mostra la dipendenza della capacità di giunzione dalla tensione inversa applicata. La capacità di giunzione viene usata per determinare la velocità di risposta del fotodiodo.

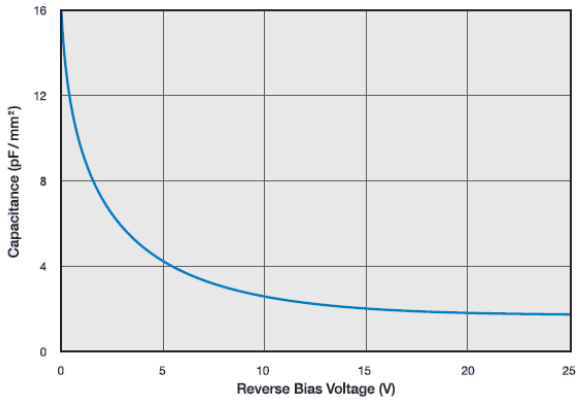



**Figura 2.8:** Capacità di giunzione versus Tensione inversa

La resistenza serie del fotodiodo si pone tra la resistenza dei contatti e la resistenza della zona di non deplezione del silicio. E data da:

$$R_s = \frac{(W_s - W_d)\,\rho}{A} + R_c \qquad \textbf{(2.19}$$

Dove $W_s$ è lo spessore del substrato, $W_d$ è la larghezza della zona di deplezione, A è l'area della superficie diffusa della giunzione, $\rho$ è la resistività del substrato e $R_c$ è la resistenza di contatto.

La resistenza serie viene usata per determinare la linearità del fotodiodo in modalità fotovoltaica (senza applicare tensioni inverse). Sebbene un fotodiodo ideale vorrebbe non avere resistenza di tipo serie, nella realtà i valori tipici si aggirato tra 10 e 1000 Ohm.

### 2.2.3 Caratteristiche ottiche

La responsività del fotodiodo è una misura della sensibilità della luce ed è definita come il rapporto tra la fotocorrente e la potenza della luce ricevuta, rispetto ad una determinata lunghezza d'onda:

$$R_\lambda = \frac{I_P}{P} \qquad \textbf{(2.20}$$

In altre parole è la misura di quanto è efficace la conversione della potenza luminosa in una corrente elettrica. La responsività varia con la lunghezza d'onda della luce incidente figura, così come con la tensione inversa applicata e la temperatura.

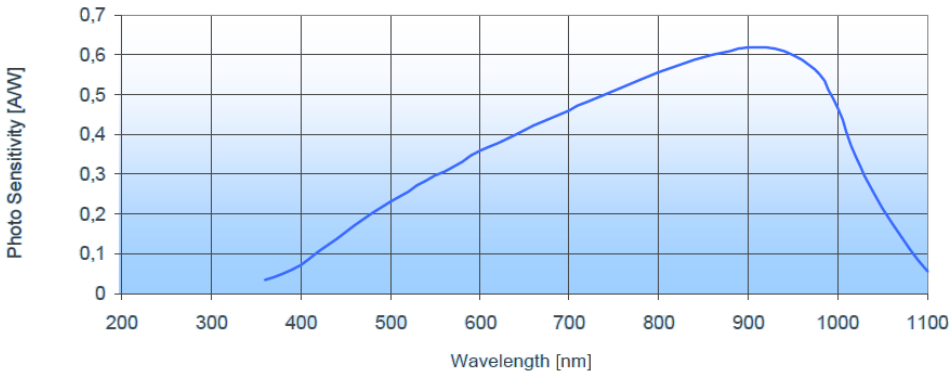



**Figura 2.9:** Responsività spettrale

### 2.2.4 Impiego nella misura

Quando al PSD viene applicata una tensione di polarizzazione inversa la fotocorrente che si genera $I_{ph}$, tenderà a scorrere dal catodo verso il terminale di anodo più vicino. Essa di ripartisce in base alla resistenza che incontra nei due cammini e ciascuna delle correnti raccolte dagli elettrodi risulta proporzionale ala distanza tra questi ultimi e il punto in cui si è generata la carica. La differenza tra le correnti raccolte sugli elettrodi $X_1$ e $X_2$ fornisce la coordinata x del centro dello spot laser.

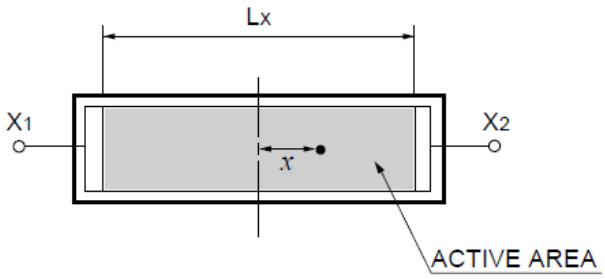



**Figura 2.10:** Superficie del PSD

Si possono scrivere le espressioni dei cammini percorsi dalla fotocorrente in funzione della posizione *x*:

$$R_{Xi} = x\rho \qquad \textbf{(2.21}$$

$$R_{X2} = (L_x - x)\rho \qquad \textbf{(2.22}$$

$$R_{X1} + R_{X2} = L\rho \qquad \textbf{(2.21}$$

dove con ρ si indica la resistenza per unità di lunghezza. Ne consegue che, le correnti $I_{X1}$ e $I_{X2}$ uscenti dagli elettrodi di anodo saranno:

$$I_{X1} = I_{ph} \frac{R_{X2}}{R_{X1} + R_{X2}} = I_{ph} \frac{L_x - x}{L_x} \quad \textbf{(2.22}$$

$$I_{X2} = I_{ph} \frac{R_{X1}}{R_{X1} + R_{X2}} = I_{ph} \frac{x}{L_x} \quad \textbf{(2.21}$$

Dalla differenza delle due correnti espresse dalle relazioni si ottiene un segnale proporzionale alla posizione *c*:

$$I_{diff} = I_{X2} - I_{X1} = \left(\frac{2x}{L} - 1\right) I_{ph} \quad \textbf{(2.22}$$

Il solo segnale differenza non è sufficiente per realizzare uno strumento perché, una variazione della potenza ottica determinerebbe una fotocorrente $I_{PH}$ diversa che noi interpreteremmo invece, come uno spostamento della coordinata *x*.
Per rendere il segnale che contiene il dato misurato insensibile dalla potenza ottica P ricevuta, si usa rapportare la differenza tra le due correnti degli elettrodi alla somma delle stesse due correnti che di fatto, costituiscono la fotocorrente $I_{PH}$.

$$I_{som} = I_{X2} + I_{X1} = I_{ph} \frac{L - x}{L} + I_{ph} \frac{x}{L} = I_{ph} \quad \textbf{(2.23}$$

$$\frac{I_{diff}}{I_{som}} = \frac{I_{X2} - I_{X1}}{I_{X2} + I_{X1}} = \left(\frac{2x}{L} - 1\right) \frac{I_{ph}}{I_{ph}} = \frac{2x}{L} - 1 \quad \textbf{(2.24}$$

Il segnale ottenuto è quindi indipendente anche dalla responsività del PSD.

## 2.3 Bozza di progetto

L'idea per realizzare questo strumento è di sfruttare un DSP per effettuare la differenza e la somma delle due correnti così ottenute. Il collegamento tra il DSP e il PSD non potrà essere diretto, ma necessiterà di un elettronica analogica di interfaccia che effettui un primo processamento del segnale per renderlo cosi acquisibile dal DSP.
Il DSP poi comanderà un DAC per dare una prima visualizzazione del segnale e inoltre come vedremo piloterà anche il LASER per un campionamento sincrono.

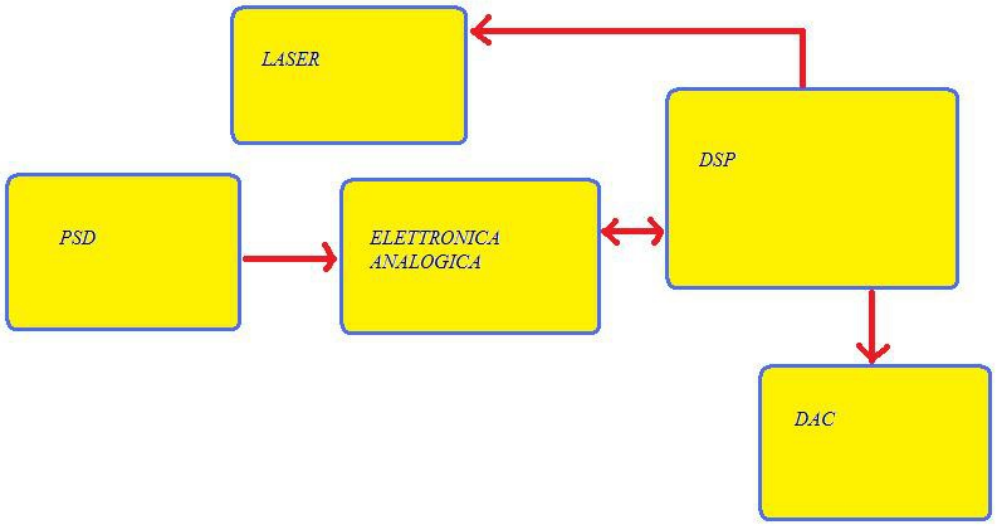



**Figura 2.11** Schema a blocchi del triangolatore

# CAPITOLO 3

# Sezione analogica

## 3.1 Introduzione

In questo capitolo verrà illustrata la realizzazione della prima parte del telemetro a triangolazione laser, ossia la progettazione dell'elettronica di interfaccia, la scelta dei componenti relativi e l' elettronica analogica di supporto che effettuerà un iniziale processamento del segnale.

Verranno proposte due possibili soluzioni, inizialmente una più immediata, con un funzionamento a tempo continuo del fascio laser con un conseguente lavoro tradizionale dei circuiti analogici che amplificheranno e filtreranno il segnale.

La seguente soluzione sarà impostata su un funzionamento impulsato del fascio laser con una conseguente integrazione di questi impulsi e un campionamento sincrono.

La soluzione impulsata mira a limitare il rumore integrato col segnale utile però ciò comporterà altri problemi quali i disturbi iniettati dai segnali a frequenza più alta ed un'elettronica necessaria al sostenimento di tali segnali.

### 3.1.1 Il LASER

Per effettuare la triangolazione ottica e la misura di vibrazione bisogna inanzi tutto disporre di un LASER. La scelta ricade necessariamente su un LASER a semiconduttore per motivi economici e pratici dato che i LASER a gas e a stato solido sono costosi, ingombrati e spesso e volentieri anche fragili. Inoltre questa tecnica di misura di distanza non necessita

di LASER di alta qualità, di specifiche restringenti sulla lunghezza d'onda o su potenze d'uscita particolarmente stabili, dato che la tecnica di misura in se è svincolata da tali parametri.Le motivazioni principali che hanno portato alla scelta del LASER sono la massimizzazione del rapporto segnale-rumore per avere misure piu solide rispettando però i limiti imposti dai livelli di sicurezza e una lunghezza d'onda di emissione nel visibile per permettere una maggiore comodità nella realizzazione, nonché nella misura e per una maggiore sicurezza dato che un fascio visibile nell'occhio comporta una reazione immediata del soggetto.

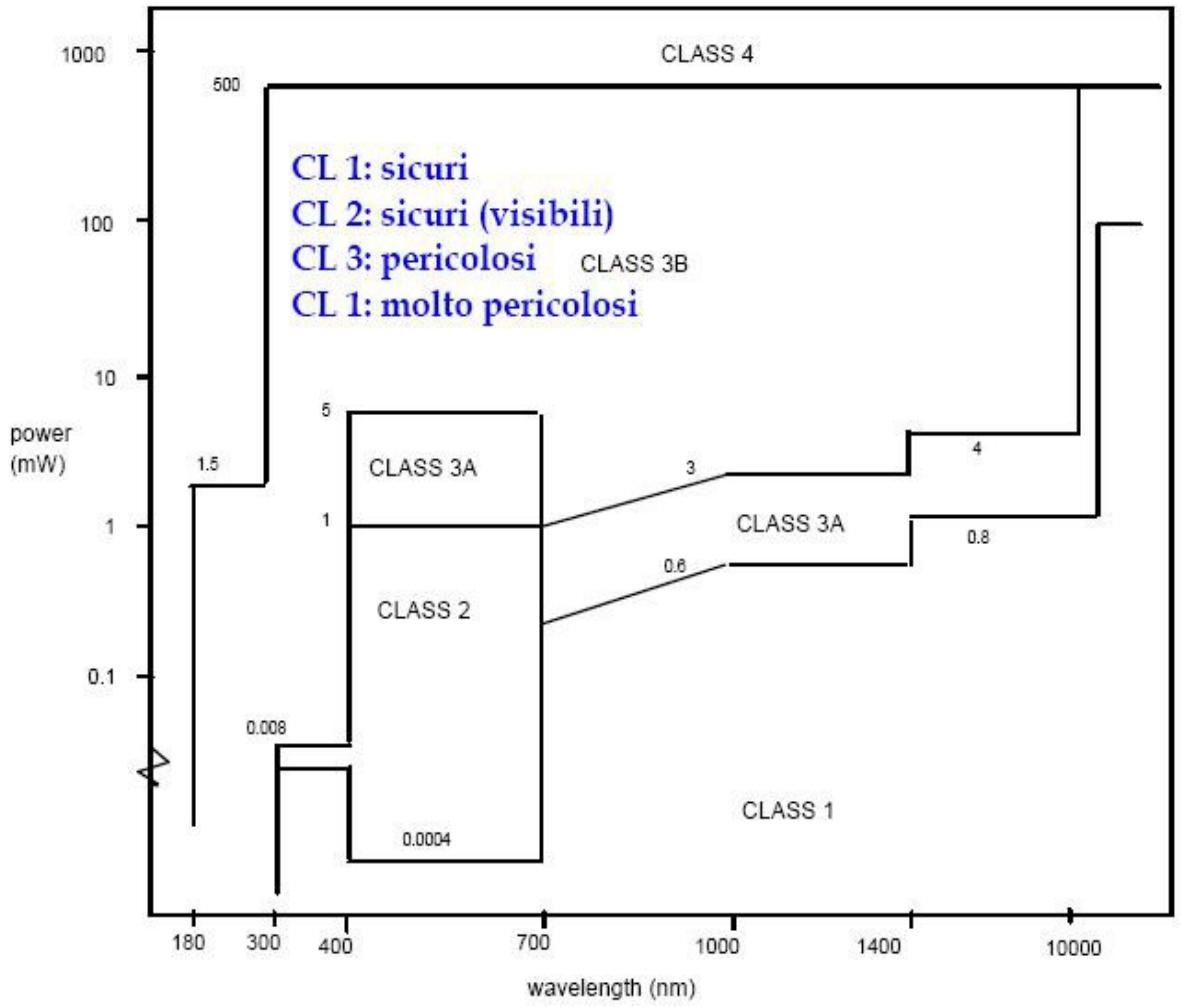



**Figura 3.1** Diagramma di laser safety

Si è scelto di utilizzare per questo sistema il diodo laser *APCD-650-02-C3* di *Arima Laser* in quanto offre diverse funzionalità in un unico oggetto di dimensioni molto ridotte (6.3 x 10.5 mm). Nello stesso package è contenuta anche la circuiteria necessaria per un'autoregolazione della potenza ottica in uscita.

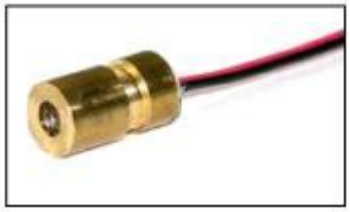


**Figura 3.2** Laser *APCD-650-02-C3*

La presenza di un fotodiodo interno, per la lettura della potenza ottica emessa dal LASER, e una retroazione negativa che agisce direttamente sulla corrente di poma del LASER, permettono a quest'ultimo di mantenere costante il suo punto di lavoro per valori della tensione di alimentazione tra 2.5V e 3.3V. Un ulteriore vantaggio di questo dispositivo è dato dalla presenza di una lente interna di materiale acrilico e di un sistema per la messa a fuoco manuale del fascio laser.

Il diodo LASER ha una potenza ottica di uscita di 2.5mW, sufficientemente bassa per garantire la sicurezza dell'utente, e una lunghezza d'onda $\lambda$ = 650 nm; la luce emessa nel visibile permette un facile puntamento.

| | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|
| **Optical** | | | | |
| Center Wavelength $\lambda_C$ | - | 655 | - | nm |
| Output Power | - | - | 3.0 | mW |
| Divergence angle | | 1.1 | | mrad |
| Output Aparture | | 2.4 | | mm |
| Beam Size at 10M | | 10 | | mm |
| **Electrical** | | | | |
| Current draw | - | - | 35 | mA |
| Supply voltage | 2.5 | - | 3.3 | V |
| **General** | | | | |
| Body | | Brass | | |
| Dimensions | | 6.2 x 11.0 | | mm |
| Lens | | Acryl | | |
| Mean time to failure (MTTF) | | >10000 | | h |

**Tabella 3.1** Caratteristiche Laser *APCD-650-02-C3*

### 3.1.2 PSD

Passiamo ora alla descrizione del ricevitore di posizione utilizzato nel progetto. Considerazioni relative ai costi e alla linearità della risposta, hanno condotto all'utilizzo di un rilevatore PSD a singolo asse.

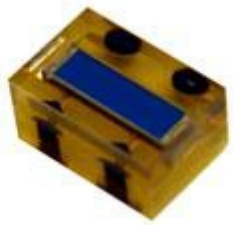


**Figura 3.3** PSD

Per lavorare in modo lineare conviene polarizzare il PSD affinchè lavori in modo fotoconduttivo piuttosto che fotovoltaico dato che quest'ultimo presenta una caratteristica logaritmica.

$$i_d = i_0\left(\exp\left(V_d / V_{th}\right) - 1\right) \qquad \textbf{(3.1}$$

Nel modo fotoconduttivo impongo la tensione ai capi del diodo in maniera che la corrente sia fissa. Preferibilmente si impone una tensione minore o uguale a 0 V così che la corrente imposta nel diodo sia un valore piccolo, non tanto per l'alteramento della misura, dato che implicherebbe semplicemente un offset, ma bensì per il rumore shot ad essa connessa. Nella realtà inoltre un rumore di tensione ai capi del diodo causerebbe forti variazioni di corrente dato che per $V_d > 0$ si ha una caratteristica del diodo molto ripida.
Per polarizzare il PSD affinché lavori in modo fotoconduttivo si collega il catodo all'alimentazione, mentre i due anodi verranno collegati ai morsetti negativi degli operazionali in configurazione a transimpedenza in maniera tale che le fotocorrenti

vengano così richiamate dalla retroazione e convertite in tensioni. Dato che la corrente è imposta nel PSD si ha che la corrente indotta dai fotoni catturati finisce tutta nella massa virtuale e il valore è

$$i_{ph} = \frac{\eta I_0 A e \lambda}{h c} \qquad \textbf{(3.2}$$

dove $\eta$ è il rendimento (ossia quanti fotoni incidenti vengono realmente converti in elettroni), $I_0$ l'intensità del fascio laser, $A$ l'area, $e$ la carica dell'elettrone, $h$ la costante di Planck e $c$ la velocità della luce.

| Parametro | Valore |
|---|---|
| Area attiva | 3.5 mm x 1 mm |
| Corrente di buio a 10V | 6.5 nA |
| Capacità di giunzione a 10V | 15 pF |
| Responsività a 633 nm | 0.4 A/W |
| Resistenza tra i due elettrodi di anodo | 50 ± 20 kΩ |
| Risoluzione dovuta al rumore | 0.05 µm |
| Errore sulla posizione rilevata a 632 nm | ± 0.2 % |
| Tensione di Breakdown | 35V |
| Temperatura operativa | -25 ÷ +85°C |

**Tabella 3.2** Parametri PSD

## 3.2 Transimpedenza

La transimpedenza è ideale per i segnali di corrente poiché presenta una bassa impedenza d'ingresso, fondamentale per questo tipo di segnale (inutile, anzi dannosa per i segnali di tensione i quali prediligono per una buona rilevazione un'impedenza d'ingresso più alta possibile), inoltre ne permette un'amplificazione e una conversione in tensione infatti il trasferimento è dato da meno l'impedenza vista nel ramo di reazione che è pari a $R_f$. Ciò perché la retroazione impone la massa virtuale al morsetto negativo dell'operazionale, quindi la fotocorrente del PSD non può alterare il potenziale e quindi l'unico percorso disponibile è tramite il ramo di reazione poiché se finisse nella $R_d$ dell'operazionale il potenziale varierebbe mentre nella $R_f$ no poiché l'uscita dell'operazionale può modificare il suo valore e lo varia proprio nella maniera tale da non alterare la massa virtuale.

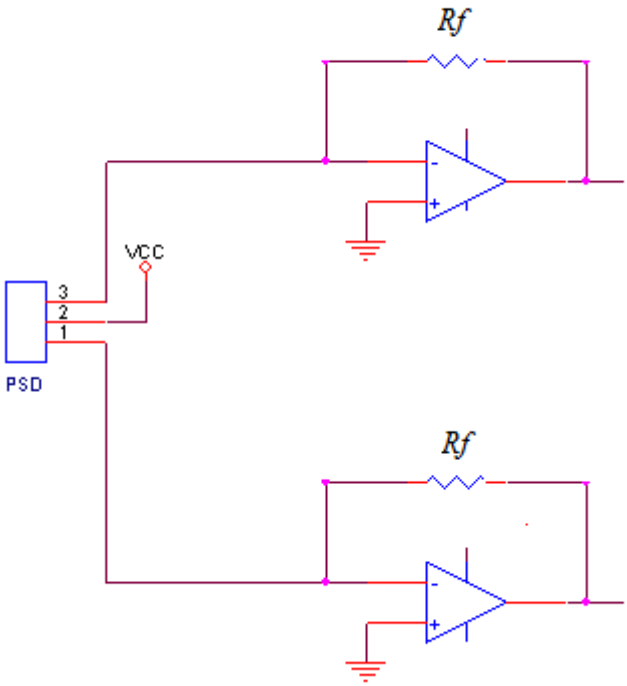



**Figura 3.4** Schema elettrico transimpedenza

Le tensioni risultano quindi essere

$$V_l = -I_l \cdot R_f \quad \textbf{(3.3}$$

$$V_r = -I_r \cdot R_f \quad \textbf{(3.4}$$

La scelta delle resistenza (100 kΩ) è dovuto all'entità delle correnti, che sperimentalmente sono risultate essere di circa 2 μA. Questo valore è il risultato di molteplici aspetti quali, la potenza ottica emessa dalla sorgente laser, la distanza del bersaglio di misura (4 cm), il coefficiente di diffusione della superficie colpita dal laser, l'angolo di vista della lente ed infine, la responsività del PSD (0.4 A/W). In questo modo, e in condizioni di fascio laser ricevuto nel centro del fotorivelatore, assumono entrambe il valore di circa 200mV.
La bassa impedenza d'ingresso è possibile grazie all'effetto della retroazione che abbassa l'impedenza vista di un fattore 1-$G_{loop}$ (guadagno d'anello).

In questo caso si è scelto come operazionale l'*MC33079* in cui le caratteristiche elettriche vengono elencate in seguito.

• Dual Supply Operation: 5.0 V to 18 V
• Low Voltage Noise: 4.5 nV/Hz
• Low Input Offset Voltage: 0.15 mV
• Low T.C. of Input Offset Voltage: 2.0 V/°C
• Low Total Harmonic Distortion: 0.002%
• High Gain Bandwidth Product: 16 MHz
• High Slew Rate: 7.0 V/s
• High Open Loop AC Gain: 800 @ 20 kHz
• Phase Margin $\phi_m$ = 55°

• Large Output Voltage Swing: +14.1 V/ −14.6 V
• Open Loop Output Impedance $|Z_O|$ = 37 Ω
• Differential Input Resistance $R_{in}$ = 175 k Ω
• Differential Input Capacitance $C_{in}$ = 12 pF
• Equivalent Input Noise Voltage $e_n$ = 4.5 nV/√Hz
• Equivalent Input Noise Current $i_n$ = 0.5 pA/√Hz

Questo è un buon operazionale per realizzare la nostra transimpedenza, possiede un basso rumore d'ingresso, un basso offset, già internamente compensato, quindi utilizzabile a buffer e con un banda di 16MHz che ampiamente copre i 50kHz di banda del nostro segnale, anzi sarà poi necessario filtrare così da eliminare il rumore presente nelle frequenze a noi non utili.

L'alimentazione dovrà essere duale poiché il segnale in uscita dalla transimpedenza sarà negativo visto che la configurazione può essere solamente invertente.

Calcoliamoci il valore della $R_{in}$. In ingresso alla nostra transimpedenza si vede la resistenza in retroazione del valore di 100 kΩ in parallelo alla $R_{in}$ differenziale dell'operazionale del valore di 175kΩ (relativamente bassa, cio poiché l'operazionale è stato realizzato in tecnologia bipolare).

$$Rin^0 = R_f \parallel R_d = 63\,k\Omega \qquad \textbf{(3.5}$$

In **Figura 3.5** viene mostrato lo schema circuitale ad anello aperto.

Il guadagno ad anello aperto equivale al guadagno d'andata, ossia il guadagno differenziale dell'operazionale pari a 100dB ( $10^5$ ) moltiplicato per il blocco in retroazione.

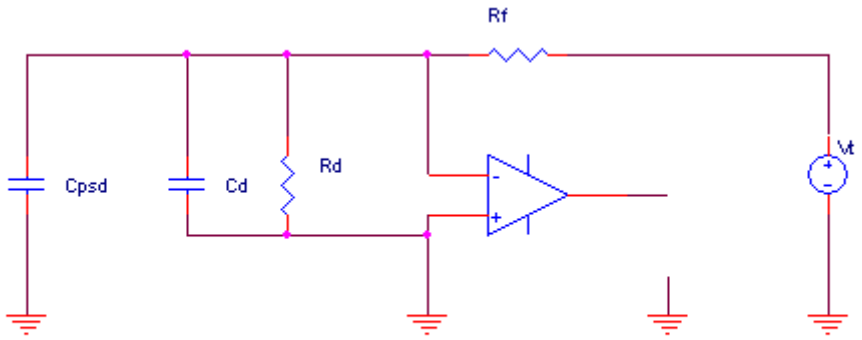



**Figura 3.5** Schema circuitale per il calcolo del guadagno d'anello della transimpedenza

In continua il beta del nostro sistema (ossia il blocco in retroazione) equivale alla partizione resistiva tra la resistenza in retroazione e la $R_d$ dell'operazione, quindi vale

$$\beta = \frac{R_d}{R_d + R_f} = 0.6363 \qquad \textbf{(3.6}$$

e da qui possiamo finalmente calcolare la $R_{in}$ ad anello chiuso come

$$Rin = \frac{Rin^0}{1 - G_{loop}} = 630\, m\Omega \qquad \textbf{(3.8}$$

che deriva dalla teoria dei sistemi reazionati.

Verifichiamo ora la stabilità della nostra transimpedenza tenendo in considerazione l'effetto della resistenza e la capacità parassite in ingresso all'operazionale.

Ad alta frequenza il termine capacitivo dominerà il parallelo con la $R_d$ quindi il beta

tenderà a 0. Ci resta da calcolare il polo di beta e quindi potremo costruire i grafici di Bode per determinare la stabilità del sistema.

Nel calcolo della stabilità non bisogna dimenticarsi degli effetti di carico del PSD che interverrà con la sua capacità parassita. La $\tau$ è $(C_d + C_{PSD}) \cdot R_f \| R_d = 1763\text{nsec}$ da cui ricaviamo una pulsazione di 580 krad e quindi una frequenza di taglio di 90 kHz.

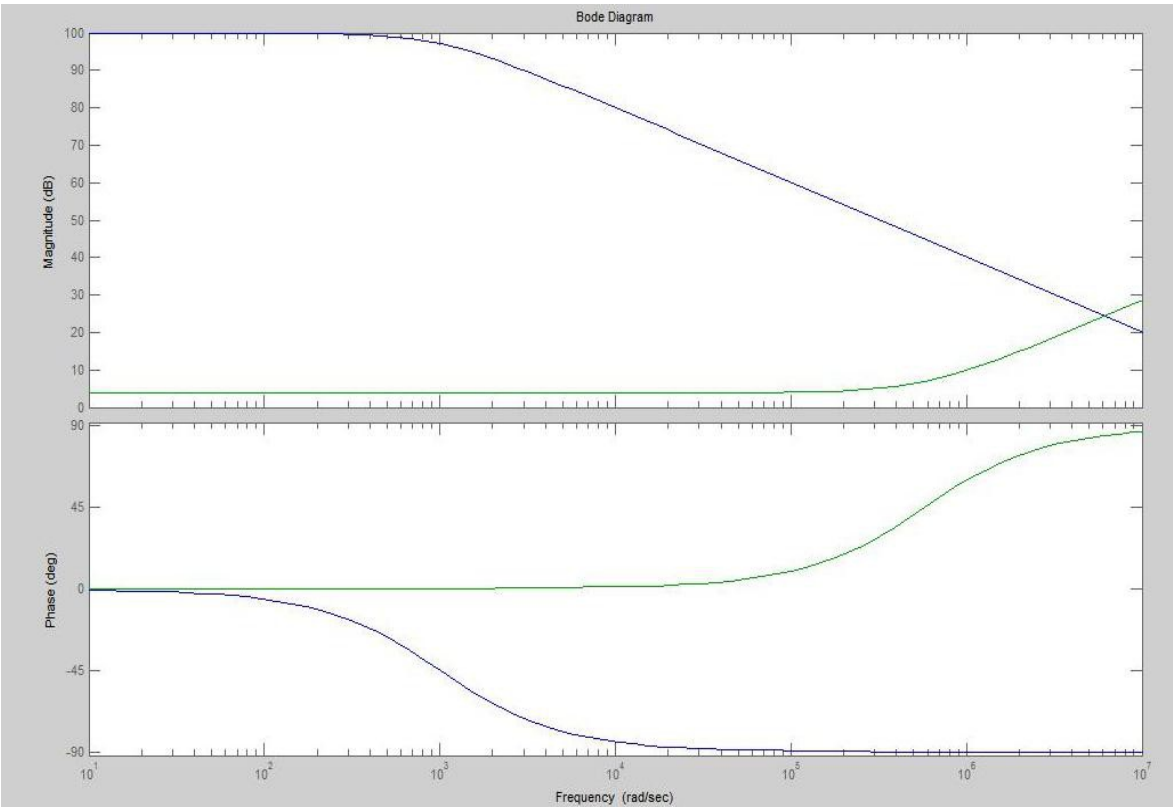



**Figura 3.6** Diagrammi di Bode dell'operazionale e del termine $1/\beta$

Come era intuibile il sistema risulta instabile poiché come si vede nella **Figura 3.6** il termine $1/\beta$ taglia il grafico del guadagno di andata con una pendenza di 40dB/decade poiché il termine $1/\beta$ sale a 20dB/decade mentre il guadagno d'andata scende con una pendenza di 20dB/decade che sommati danno appunto 40dB/decade.

Per risolvere tale problema possiamo mettere una capacità in parallelo alla resistenza di

retroazione così in un colpo stabilizziamo e filtriamo poiché variamo l'impedenza che determina il guadagno in modo da eliminare le frequenze in cui non abbiamo segnale utile ma solo disturbi e rumore.

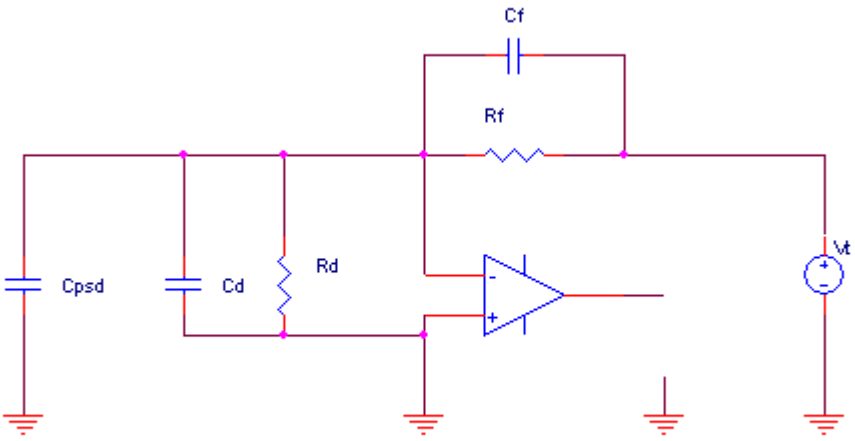



**Figura 3.7** Schema circuitale per il calcolo del guadagno d'anello della transimpedenza con la capacità di retroazione

500kHz può essere una buona frequenza in cui posizionare il polo del nostro filtro, con un calcolo inverso al precedente si ricava una capacità di 3.4pF a cui la capacità fisica che si avvicina di più è 3.3pF corrispondente a una frequenza di taglio di 480kHz. Il trasferimento risultante sarà

$$T_{id}(s)=\frac{R_f}{(1+sC_f R_F)} \qquad \textbf{(3.9}$$

Verifichiamo ora ancora la stabilità del nostro sistema.

In continua il trasferimento del blocco in retroazione è dato come prima dal partitore resistivo poiché le capacità sono circuiti aperti, quindi 0.63, mentre ad alta frequenza le impedenze capacitive domineranno i paralleli con le resistenze e quindi il trasferimento sarà dato stavolta dal partitore capacitivo.

$$\beta = \frac{C_f}{(C_f + C_d + C_{PSD})} = 0.1089 \qquad \textbf{(3.10}$$

Presumibilmente ci sarà quindi un polo seguito da uno zero che riporterà il guadagno assintoticamente ad una costante.

Il nuovo polo del $\beta$ è dato ora dalla somma delle capacità e dal parallelo delle resistenze, quindi

$$\tau = (C_f + C_d + C_{psd}) \cdot R_f \| R_d \qquad \textbf{(3.11}$$

$\tau$= 973 nsec che corrispondono a una frequenza di taglio pari a

$$f = \frac{1}{\tau \cdot 2 \cdot \pi} = 80\text{kHz} \qquad \textbf{(3.12}$$

Lo zero è probabilmente dato dal parallelo capacità-resistenza in retroazione, cioè abbiamo che il segnale non viene trasferito quando le correnti nella capacità e nella resistenza sono uguali in intensità ma opposte in verso. Ciò avviene quando la capacità presenta una reattanza uguale e contraria alla resistenza quindi

$$\frac{-1}{sC_f} = R_f \qquad \textbf{(3.13}$$

quindi

$$s = \frac{-1}{R_f C_f} \qquad \textbf{(3.14}$$

perciò abbiamo lo zero ai 480kHz, la frequenza di taglio del filtro passa-basso che abbiamo realizzato, infatti uno zero nell'anello di retroazione diventa un polo per il sistema completo.

Verifichiamo che lo zero combaci con il resto dei dati calcolati, infatti il prodotto guadagno-banda deve essere costante nel tratto in cui la funzione di trasferimento scende di 20dB/decade (è presumibile che scenda piuttosto che salga poiché il polo è a più bassa frequenza rispetto allo zero).

$$0.63 \cdot 160\text{kHz} = 0.1 \cdot Zero \qquad \textbf{(3.15}$$

$$Zero = 480\text{kHz} \qquad \textbf{(3.16}$$

Possiamo a questo punto costruire il grafico di Bode per verificare la stabilità del sistema riporato in **Figura 3.8**.

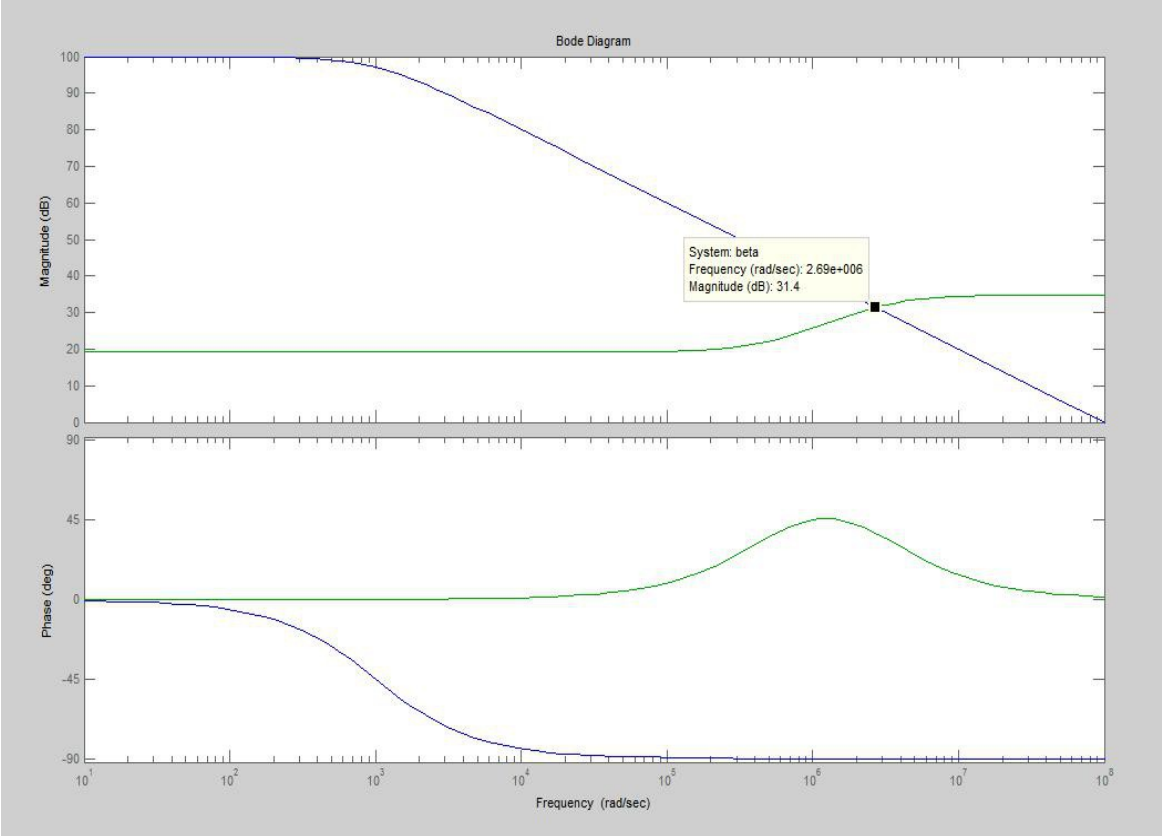



**Figura 3.8** Diagrammi di Bode dell'operazionale e del termine $1/\beta$ con la capacità di retroazione

Come si può vedere ora il termine $1/\beta$ taglia il guadagno d'andata nel punto in cui si trova il suo polo, quindi ora il nostro sistema si può considerare stabile. Verifichiamo il margine di fase

$$\phi_m = 180° - 90° - atan\left(\frac{f}{f_p}\right) + atan\left(\frac{f}{f_z}\right) = 52.3° \qquad \textbf{(3.17}$$

che ci garantisce una certa stabilità (i primi 90° sono determinati dal polo a bassa frequenza dell'operazionale).

## 3.3 Stadio di guadagno

In uscita da questo stadio appena descritto disponiamo di un segnale in tensione che varia da 0 a -400mV. Questo segnale non va ancora bene poiché poi si vorrà campionare il segnale con l'ADC del DSP che ha una dinamica di 3.3V. Dato che l'*MC33079* dispone di quattro operazionali si possono usare i restanti due per realizzare uno stadio di guadagno invertente per rendere positivo il segnale e sfruttare l'intera dinamica dell'ADC.

Si è scelto un guadagno pari a -5 che porterà il valore massimo di tensione a 2V per permettere una misura anche in bersagli più diffusivi che altrimente farebbero saturare lo strumento.

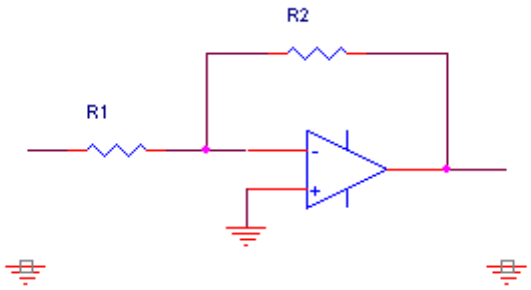



**Figura 3.9** Schema elettrico dello stadio di guadagno invertente

Il principio di funzionamento è uguale a quello dello stadio precedente, la retroazione impone la massa virtuale, quindi applicando in ingresso una tensione si ottiene una corrente pari a

$$I = \frac{V}{R_1} \qquad \textbf{(3.18}$$

che poi potrà andare solo sul ramo di retroazione, quindi attraverso $R_2$.

Il guadagno di questo stadio risulta quindi essere

$$G = \frac{-R_2}{R_1} = \frac{-47\,k\Omega}{10\,k\Omega} = -4.7 \qquad \textbf{(3.19}$$

che porterebbero ai nuovi trasferimenti

$$V_l = I_l \frac{R_f \cdot R_2}{R_1} = I_l \cdot 470\,k\Omega \qquad \textbf{(3.20}$$

$$V_r = I_r \frac{R_f \cdot R_2}{R_1} = I_r \cdot 470\,k\Omega \qquad \textbf{(3.21}$$

Anche in questo caso è necessario filtrare dato che il filtro dello stadio precedente non limiterebbe il rumore generato dai componenti degli stadi successivi, che benchè riportati in ingresso sarebbero diminuiti del fattore di transimpedenza del primo stadio, si ripercuoterebbero in uscita con una banda maggiore e che quindi ner amplificherebbe il peso.

Si è messa inizialmente una capacità di feedback pari a 15 pF che comporterebbe una frequenza di taglio pari a 225 kHz che, nonostante risulti più bassa rispetto a quella dello stadio precedente, si è deciso di lasciare poiché è comunque dello stesso ordine.

L'analisi della stabilità è analoga a prima come si può vedere dallo schema elettrico

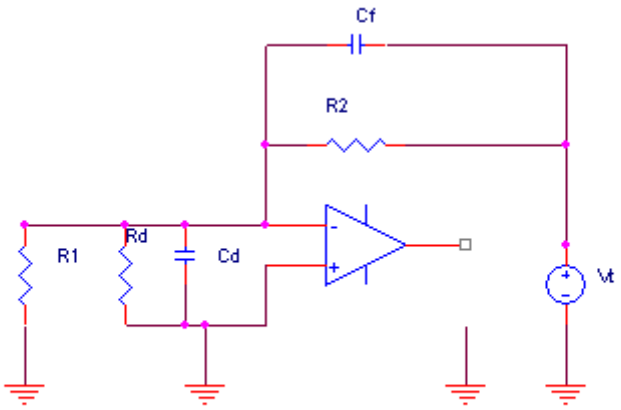



**Figura 3.10** Schema circuitale per il calcolo del guadagno d'anello dello stadio di guadagno invertente

Ci risulta un guadagno in continua

$$\beta = \frac{(R_d \| R_1)}{(R_d \| R_1 + R_2)} = 0.163 \qquad \textbf{(3.22}$$

e un polo

$$\tau = R_d \| R_1 \| R_2 \cdot (C_d + C_f) \qquad \textbf{(3.23}$$

$$f = \frac{1}{\tau \cdot 2 \cdot \pi} = 187\text{kHz} \qquad \textbf{(3.24}$$

e uno zero

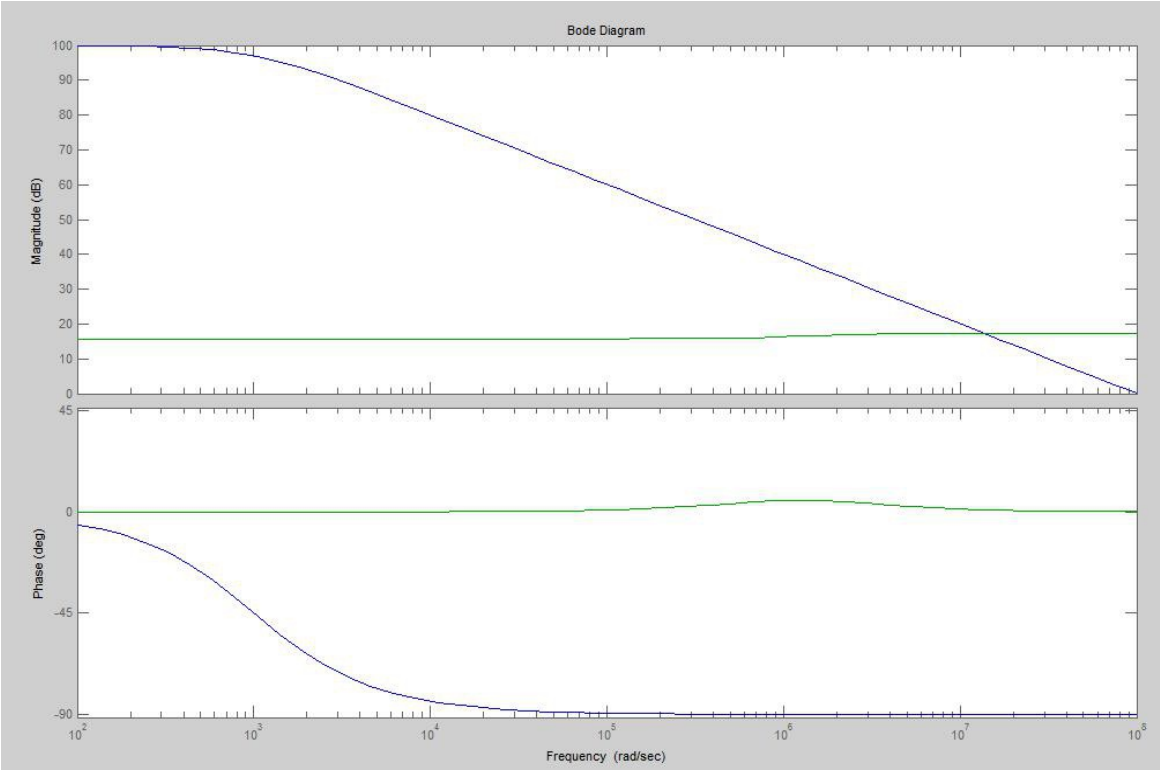
$$s = \frac{-1}{R_2 C_f} \quad \textbf{(3.25}$$

$$f = 187\text{kHz} \quad \textbf{(3.26}$$

che da un margine di fase di

$$\phi_m = 180° - 90° - atan\left(\frac{f}{f_p}\right) + atan\left(\frac{f}{f_z}\right) = 89° \quad \textbf{(3.27}$$

perfettamente stabile dato che il polo e lo zero sono talmente vicini da annullarsi quasi.




**Figuara 3.11** Diagrammi di bode dell'operazionale e del termine 1/β dello stadio invertente

## 3.4 Rumore e disturbi

Quando si esegue una misura elettronica o si effetua una trasmissione di segnale tramite un supporto fisico, bisogna tener presente che il risultato ottenuto è determinato dalla sovrapposizione del segnale utile con altri due elementi: il rumore, un contributo legato alla fisica dei portatori di carica nei dispositivi, descritto da leggi statistiche; i disturbi, dovuti ad accoppiamenti elettromagnetici o condotti consegnali provenienti da altri dispositivi.

Benchè sia il rumore che i disturbi siano elementi che limitano la ricezione del segnale o la sua misura, la loro differente natura porta ad atteggiamenti diversi nel tentativo di minimizzarli.

Il rumore in un circuito è un contributo dovuto alla natura fisica dei dispositivi, legato all'agitazione termica dei portatori a ad altri portatori stocastici. Data la sua origine non è possibile eliminare il rumore, perciò il progettista ha come unica possibilità quello di cercare di ridurlo attraverso apposite tecniche che vanno sotto il nome di filtraggio. Tale operazione deve essere svolta con grande accuratezza poiché filtrare significa ridurre il contributo di alcune frequenze spettrali. Per migliorare il rapporto segnale-rumore è necessario attenuare il rumore a quelle frequenze dove non c'è segnale, altrimenti l'operazione non porterebbe alcun beneficio.

Viceversa i disturbi apportano un contributo deterministico, conoscibile ed eliminabile non solo con tecniche di filtraggio selettivo, ma anche e sopratutto attraverso la schermatura dei cisrcuiti (sia di chi genera il disturbo, sia di chi lo subisce) o ricorrendo ad un migliore progetto delle connessioni, mirato proprio alla diminuzione degli accoppiamenti parassiti.

I disturbi più comuni sono quelli dovuti ad accoppiamento elettromagnetico, in particolare di tipo capacitivo. Quando esiste una capacità parassita tra due cavi si ha un accoppiamento del campo elettrico, la cui variazione induce una corrente nel circuito disturbato.

Le capacità parassite in un circuito, atte ad accoppiare disturbi, possono causare anche instabilità. Un esempio molto comune è dato dalla capacità parassita tra le linee di alimentazione e gli ingressi di un amplificatore operazionale, che può indurre una reazione positiva, specialmente per amplificatori a larga banda. Forti variazioni della corrente erogata dall'amplificatore possono indurre delle cadute di tensione sulle alimentazioni che, tramite l'accoppiamento capacitivo, si possono riflettere sull'ingresso dell'operazionale, causando una reazione positiva. Questo fenomeno è ancora più probabile quando si usano puà amplificatiri in cascata, in quanto i disturbi sulle alimentazioni dovuti agli stadi sucessivi si riflettono sul primo stadio, portando facilemte il guiadagno d'anello della reazione a valori superio a uno, inescando dunque oscillazione. La soluzione più semplice a questo incoveniente consiste nell'aggiungere delle capacità, tipicamente 100nF, tra le alimentazioni e massa il più vicino possibile agli integrati. In questo modo si diminuisce l'impedenza delle alimentazioni alle alte frequenze, riducendo i segnali dovuti alle variazioni di corrente assorbita dagli integrati e quindi abbassando notevolmente il guadagno d'anello della reazione positiva con gli ingressi. In questo caso non si possono utilizzare capacità elettrolitiche, in quanto non hanno un buon comportamento capacitivo ad alta frequenza, indispensabile invece per questa applicazione.Una ulteriore soluzione consiste nel distanziare i fili di ingresso del segnale dalle linee di alimentazione, se possibile, in modo da diminuire l'accoppiamento. A questo proposito, in caso di piste di un PCB, può aiutare un piano di massa sottostante, che fa diminuire notevolmente le capacità mutue tra le piste, oppure una pista connessa a massa interposta tra le due.

Nei circuiti costituiti da più stadi di guadagno in cascata il confronto trai contributi deve essere eseguito nello stesso punto del circuito. Se il primo stadio presenta un guadagno elevato risulta ragionevole considerare solo il suo contributo di rumore. I contributi di rumore all'uscita della transimpedenza si possono trovare moltiplicando le singole densità spettrali di rumore in ingresso per il quadrato del trasferimento a bassa frequenza e per la relativa banda equivalente di rumore.

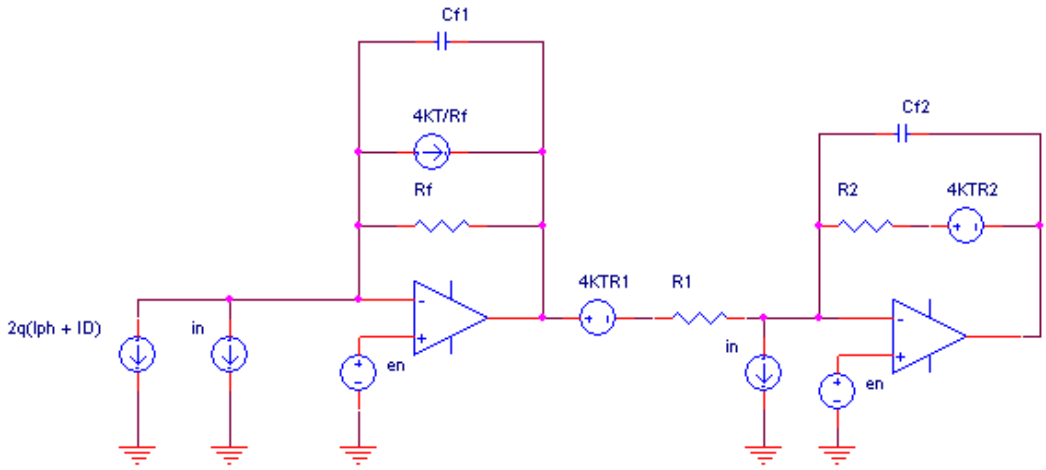



**Figura 3.12** Schema elettrico delle sorgenti di rumore

Elenchiamo tutti i contributi di rumore, gli riportiamo in ingresso e poi decideremo quali sarà più ragioneole trascusare.

I) Rumore shot della fotocorrente generata e della corrente di buio del PSD

$$2q\left(I_{ph}+I_{D}\right)=\left(1.13\ pA/\sqrt{Hz}\right)^{2}$$

che risulta già in ingresso.

II) Rumore termico della resistenza di feedback nallo stadio a transimpedenza

$$\frac{4KT}{R_{f}}=\left(0.4\ pA/\sqrt{Hz}\right)^{2}$$

che espresso in corrente anche questo risulta già in ingresso.

III) Rumore di corrente dell'operazionale del primo stadio

$$i_{n}=\left(0.5\ pA/\sqrt{Hz}\right)^{2}$$

che risulta già in ingresso.

IV) Rumore di tensione dell'operazionale del primo stadio

$$e_{n}=\left(4.5\ nV/\sqrt{Hz}\right)^{2}$$

che si trasfiresce in uscita del primo stadio tale e quale poiché nei suoi confronti il primo stadio si comporta da buffer, poi per riportarlo in ingresso bisognerà dividirlo per il trasferimento per il segnale al quadrato del primo stadio

$$\frac{e_n}{R_f^2}=(0.045\ pA/\sqrt{Hz})^2$$

V) Rumore termico di $R_1$

$$4\,K\,T\,R_1=(12.8\,nV/\sqrt{Hz})^2$$

che risulta in uscita al primo stadio e che quindi riportato in ingresso diventa

$$4\,K\,T\,R_1\frac{1}{R_f^2}=(0.13\ pA/\sqrt{Hz})^2$$

VI) Rumore termico di $R_2$

$$4\,K\,T\,R_2=(27.9\,nV/\sqrt{Hz})^2$$

che risulta in uscita al secondi stadio e che quindi riportato in ingresso diventa

$$4\,K\,T\,R_2(\frac{R_1}{R_2})^2\frac{1}{R_f^2}=(59\ fA/\sqrt{Hz})^2$$

VII) Rumore di tensione dell'operazionale del secondo stadio

$$e_n=(4.5\ nV/\sqrt{Hz})^2$$

che risulta in uscita al primo stadio. Riporato in ingresso diventa

$$\frac{e_n}{R_f^2}=(0.045\ pA/\sqrt{Hz})^2$$

VIII) In fine il rumore di corrente dell'operazionale del secondo stadio

$$i_n=(0.5\ pA/\sqrt{Hz})^2$$

che si riportato in ingresso diventa

$$i_n R_2^2 \left(\frac{R_1}{R_2}\right)^2 \frac{1}{R_f^2} = \left(0.05\ pA/\sqrt{Hz}\right)^2$$

L'entita del rumore stimato in ingresso è la somma quadratica delle varie densità spettrali di rumore, integrate per la banda del filtro del secondo stadio.

$$n_i = \sqrt{\left(2q\left(I_{ph} + I_D\right) + \frac{4KT}{R_f} + i_n + \frac{e_n}{R_f^2} + 4KTR_1\frac{1}{R_f^2} + 4KTR_2\left(\frac{R_1}{R_2}\right)^2\frac{1}{R_f^2} + \frac{e_n}{R_f^2} + i_n R_2^2\left(\frac{R_1}{R_2}\right)^2\frac{1}{R_f^2}\right)\frac{\pi}{2}B}$$

**(3.28**

$$n_i = 777\ pA$$

**(3.29**

Quindi l'ingresso dello stadio digitale si ritroverà un rumore di valore efficare pari a

$$n_{vout} = n_i * 470\,k\,\Omega = 365\,\mu V$$

**(3.30**

## 3.5 Gate integrator

Verificato il corretto funzionamento e le prestazioni (che verranno descritte in seguito) degli stadi precedenti ci si è chiesti se è possibile un miglioramento del sistema, o comunque un'alternativa a questo. Data l'intenzione di campionare il segnale per poi processarlo ci si accorge che non è necessario far lavorare il LASER in continua che quindi potrebbe essere impulsato. Ciò determinerà un campionamento sincrono e quindi un sincronismo nel sistema complessivo.

Il passo successivo è stato di integrare questo impulso LASER, che dopo il PSD si identificherà come impulso di corrente. L'integrazione servirà a massimizzare il segnale utile prendendo solo il rumore necessario in maniera tale da aumentare il rapporto segnale-rumore così che si migliorano le prestazioni del sistema complessivo.

Inoltre abbiamo un segnale negativo tra il primo e il secondo stadio, mentre in tutto il resto del sistema si lavora con segnali positivi, perciò sarebbe utile eliminare questo segnale negativo per passare da un alimentazione dual supply a una single supply e permettere anche l'eliminazione del secondo stadio che ha come unico scopo quello portare il segnale nella dinamica dell'ADC.

Nei capitoli seguenti verrà illustrato come sono stati realizzati tali circuiti, quali problemi realizzitivi hanno comporatato rispetto alla soluzione precedente e se effettivamente c'è stato un miglioramento nelle prestazioni.

### 3.5.1 Laser impulsato

Innanzi tutto per realizzare un LASER impulsato necessitiamo di un altro LASER poiché quello precedente ha un meccanismo di autoregolazione integrato della potenza che benchè renda più stabile il LASER, ne allunga le costanti di tempo di accensione e spegnimento dello stesso.

Il laser utilizzato per la realizzazione di questo sistema è l'*ADL-65052TL* le cui caratteristiche vengono elencate nella seguente tabella.

| Parameter | Symbol | Min. | Typ. | Max. | Unit | Conditions |
|---|---|---|---|---|---|---|
| Peak wavelength | $\lambda$ | 645 | 655 | 660 | nm | $P_o$=5mW |
| Threshold current | $I_{th}$ | - | 15 | 20 | mA | |
| Operating current | $I_{op}$ | - | 20 | 22 | mA | $P_o$=5mW |
| Operating voltage | $V_{op}$ | - | 2.2 | 2.5 | V | $P_o$=5mW |

**Tabella 3.3** Caratteristiche laser *ADL-65052TL*

Essenso questo diodo LASER, quindi a seminconduttore, sprovvisto del meccanismo di autoregolazione della potenza è piu conscio pilotarlo in corrente piuttosto che in tensione. Di seguito è mostrato lo schema elettrico di pilotaggio.

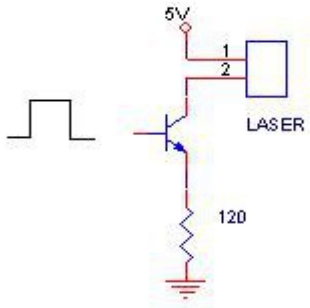



**Figura 3.13** Schema elettrico della rete di pilotaggio del laser

La base del transistore verrà comandata da un segnale PWM proveniente dal Piccolo avente ampiezza di 3.3V. Tale segnale sarà riportato sull'emettitore a meno della caduta di giunzione base-emettitore, quindi la resistenza si troverà ai suoi capi una tensione a onda rettangolare di ampiezza 0-2.6V. Questa resistenza determinerà la corrente che scorrerà nel LASER, quindi, con un facile calcolo, per ottenere la corrente adatta al LASER si è scelta una resistenza di 120 Ω.

$$I = \frac{V}{R} = \frac{2.6\text{V}}{120\,\Omega} = 21\,mA \qquad \textbf{(3.31}$$

Questo LASER a differenza del precedente è sprovvisto di una lente per la focalizzazione del fascio, quindi ne necissità di una esterna che ne aumenta leggermente l'ingombro.

### 3.5.2 Integratore

Per la teoria del filtraggio ottimo sappiamo che in presenza di rumore bianco il filtro ottimo è quello che ha la risposta all'impulso *w(t)* proporzionale all'ampiezza del segnale in ingresso *I(t)*.

$$w(t) \propto I(t) \qquad \textbf{(3.32}$$

Quindi dato che il nostro segnale in ingresso è un impulso rettangolare avremo come filtro ottimo quello che avrà come risposta all'impulso un rettangolo di larghezza pari a quella dell'impulso in ingresso (l'ampiezza non è importante poiché nella convoluzione

moltiplicherà in pari maniera segnale e rumore, l'unica cosa importante è che segua l'andamento del segnale utile). Il filtro che ha tale caratteristiche è l'integratore.

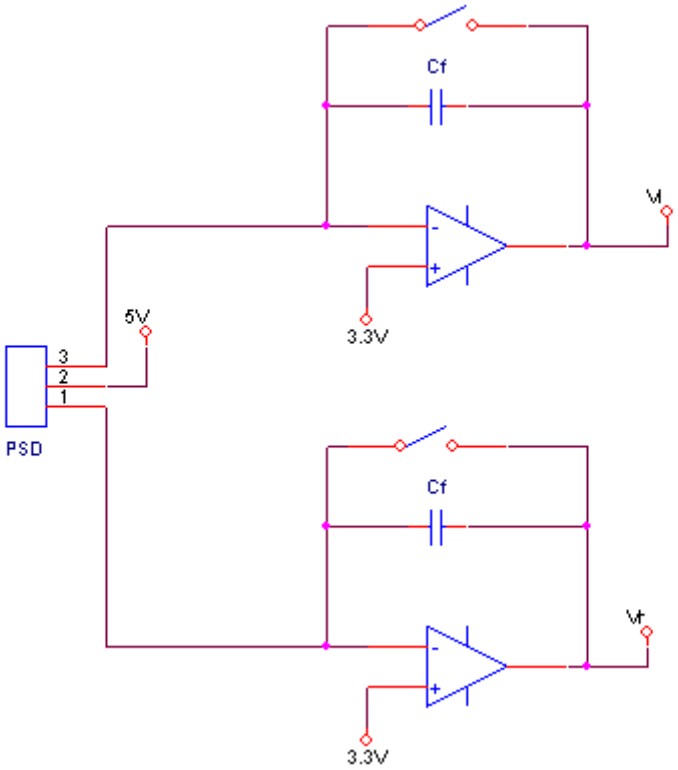



**Figura 3.14** Schema elettrico gate integrator

Il circuito sopra riportato ha due regioni temporali di funzionamento, una in cui l'interruttore è aperto e in questa fase la corrente di ingresso viene integrata nella capacità. L'altra in cui l'interruttore viene chiuso per cortocircuitare la capacità in modo da azzerare la carica ai suoi capi e prepararla quindi per integrare il prossimo impulso di corrente.

Per realizzare il filtro-ottimo, l'interruttore sarà pilotato da un segnale PWM, sempre proveniente dal Piccolo, però in controfase rispetto al segnale di pilotaggio del LASER.

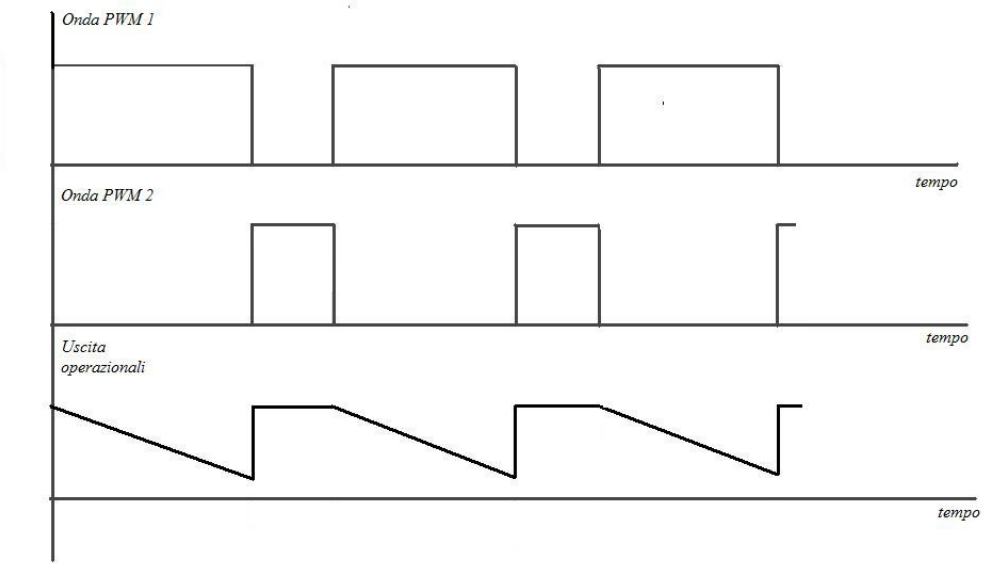



**Figura 3.15** Forme d'onda PWM e uscite degli operazionali

Quelle sopra riportate sono le forme d'onda teoriche che ci si aspetta dal funzionamento. Nella prima regione l'operazionale funziona da buffer e riporta in uscita la tensione fissata al morsetto positivo di 3.3V. Nella seconda regione l'operazionale funziona da integratore invertente, infatti si vede la rampa negativa in uscita. L'inversione e l'offset saranno poi aggiustati via software, l'importante è avere una caratteristica lineare nella zona che va dai 0 ai 3.3V, regione in cui effettuiamo il campionamento. La tensione con cui viene polarizzato il morsetto positivo è scelta appunto a questo scopo, inoltre permette al PSD di lavorare in modo fotoconduttivo, infatti sarà polarizzato tra 3.3V e 5V, ossia la tensione di alimentazione, dato che la retroazione porterà a 3.3V anche il morsetto negativo.

Teoricamente così abbiamo realizzato le nostre specifiche, ossia un' alimentazione single supply (0-5V) e una dinamica di uscita coincidente con quella dell'ADC con un singolo stadio.

Le tensioni risulteranno essere

$$V_l = 3.3\text{V} - \frac{1}{C_f}\int_0^T I_l\,dt \qquad \textbf{(3.33}$$

$$V_r = 3.3\text{V} - \frac{1}{C_f}\int_0^T I_r\,dt \qquad \textbf{(3.34}$$

Considerando una frequenza di campionamento di 1MHz e $T \approx \frac{1}{f_s}$ e una $I_{max}$= 4 $\mu A$ ricaviamo il valore di $C_f$ come

$$3.3\text{V} = \frac{I_{max}}{C_f \cdot f_s} \qquad \textbf{(3.35}$$

cosi che la massima variazione di tensione non sfori la dinamica ,quindi

$$C_f = \frac{4\mu\text{A}}{3.3\text{V} * 500\text{kHz}} = 2.42\ pF \qquad \textbf{(3.36}$$

da cui si prende 3.3pF per avere un po di margine.

### 3.5.3 Scelta dell'operazionale

L'operazionale si trova adesso a dover gestire segnali piu “veloci”, quindi quello scelto precedentemente non va più bene, ha un GBWP troppo basso. Bisogna trovarne un altro con un GBWP più alto e che sia internamente compensato dato che in una fase lavorerà da buffer.

La scelta è ricaduta sull'OPA 355 la cui banda a buffer arriva a 200MHz e le cui caratteristiche principali sono elencate di seguito

• WIDE BANDWIDTH: 200MHz GBW
• HIGH SLEW RATE: 360V/μs
• LOW NOISE: 5.8nV/√Hz
• EXCELLENT VIDEO PERFORMANCE:
• DIFF GAIN: 0.02%, DIFF PHASE: 0.05°
• 0.1dB GAIN FLATNESS: 75MHz
• INPUT RANGE INCLUDES GROUND
• RAIL-TO-RAIL OUTPUT (within 100mV)
• LOW INPUT BIAS CURRENT: 3pA
• LOW SHUTDOWN CURRENT: 3.4μA
• ENABLE/DISABLE TIME: 100ns/30ns
• THERMAL SHUTDOWN
• SINGLE-SUPPLY OPERATING RANGE: 2.5V to 5.5V
• MicroSIZE PACKAGES

Si può notare che anche il range d'alimentazione coincide con le nostre specifiche di progetto.
Essendo realizzato in tecnologia CMOS si può trascurare l'effetto della resistenza d'ingresso a differenza della capacità d'ingresso che va considerata ($C_{in}$=1.3pf,

confrontabile infatti con quella in retroazione!).

In questo caso non è necessario il calcolo della stabilità, dato che abbiamo solo impedenze capacitive e nessuna resistiva non avremo costanti di tempo e quindi non avremo poli e zeri. Ciò comporterà un trasferimento del $\beta$ piatto in frequenza e dato che l'operazionale è compensato internamente non si possono avere problemi di instabilità poiché si potrà tagliare il guadagno d'andata solo con una pendenza di 20dB/decade. Comunque calcoliamo il $G_{loop}$ per capire quanta banda abbiamo a disposizione nella fase integrativa

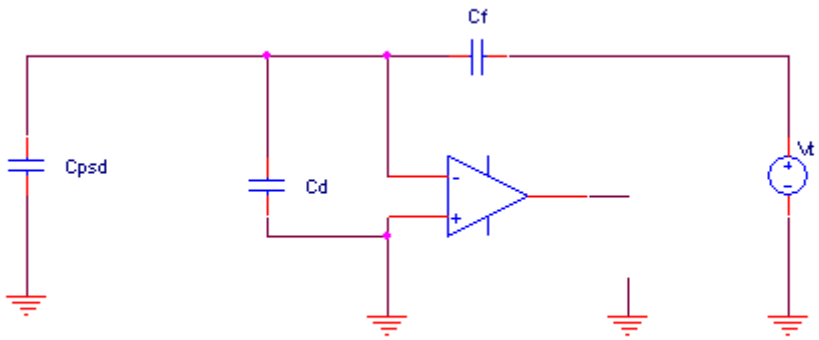



**Figura3.16** Schema elettrico per il guadagno d'anello

il β in questo caso è determinato dalla partizione capacitiva

$$\beta = \frac{C_f}{C_f + C_i + C_{PSD}} = \frac{3.3\text{pF}}{3.3\text{pF} + 15\text{pF} + 1.3\text{pF}} = 0.1684 \qquad \textbf{(3.37}$$

Come si vede risulta abbastanza basso a causa della capacità dominante del PSD che è abbastanza elevata per permettere un' area suficiente alla rivelazione dello spot laser.

La banda di lavoro diventa quindi $GBWP \cdot \beta = 33.67\text{MHz}$ sufficiente al nostro scopo.

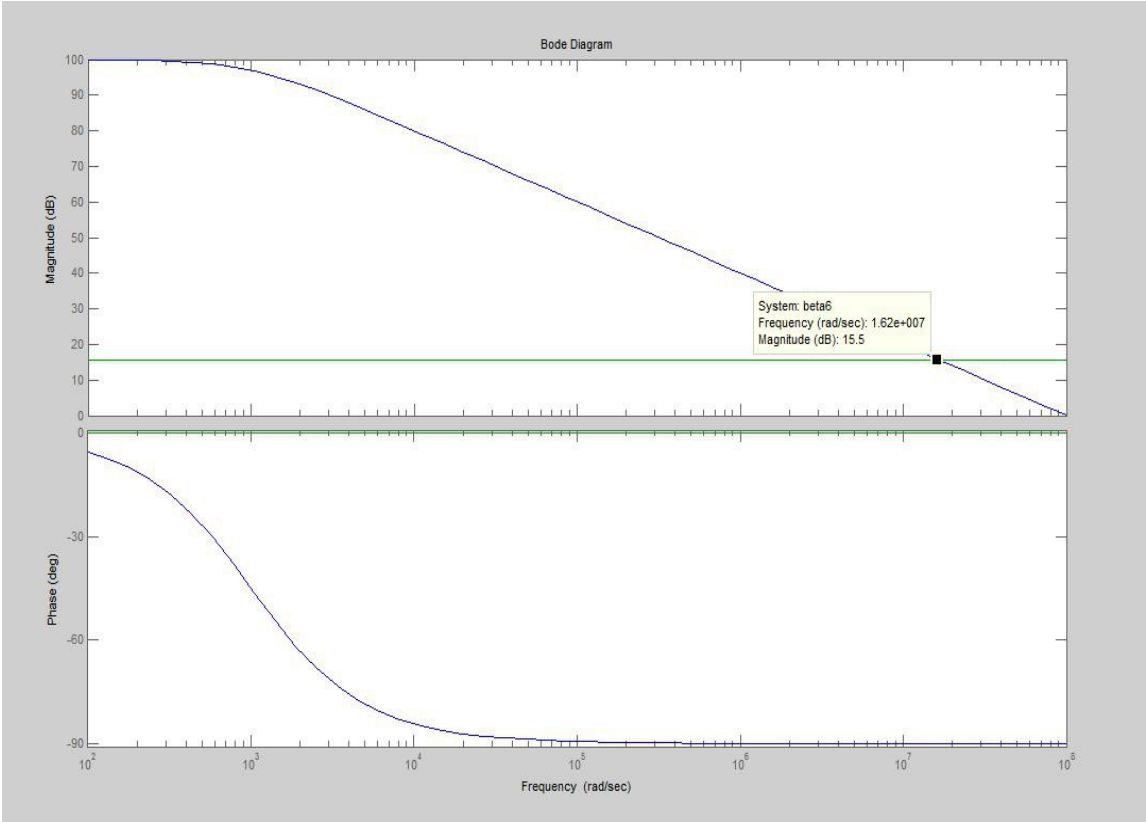



**Figura 3.17** Diagrammi di Bode della stabilità

### 3.5.4 Trasferimento

Il gate integrator non è un filtro tempo-invariante quindi non ha senso parlare di risposta all'impulso dato che nei vari istanti di tempo essa varia.
Assumiamo comunque come risposta all'impulso quella per gli impulsi che avvengono all'istante in cui l'interruttore si apre dato che in quell'istante parte anche l'integrazione del nostro segnale.

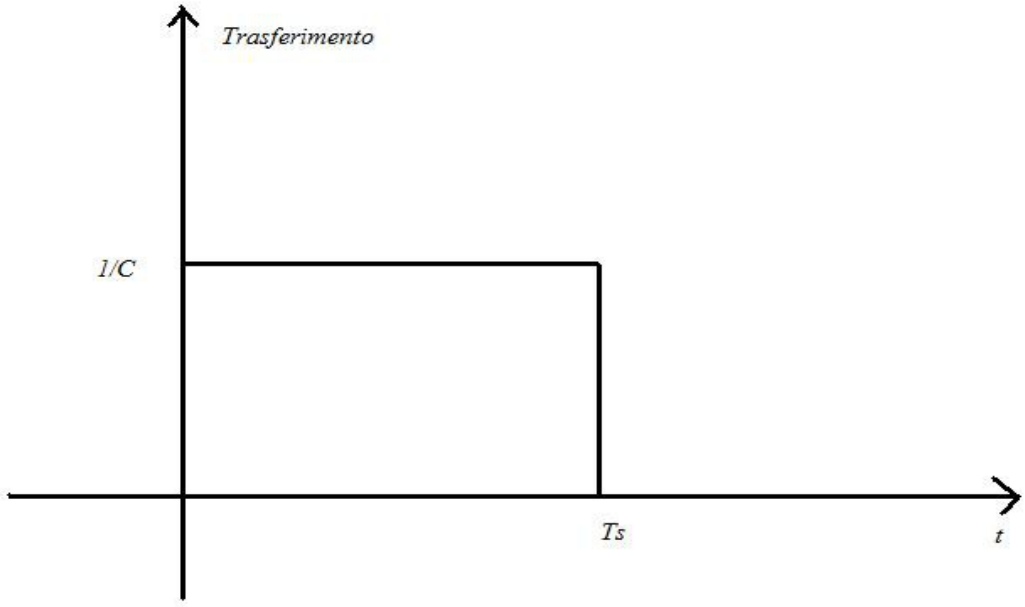
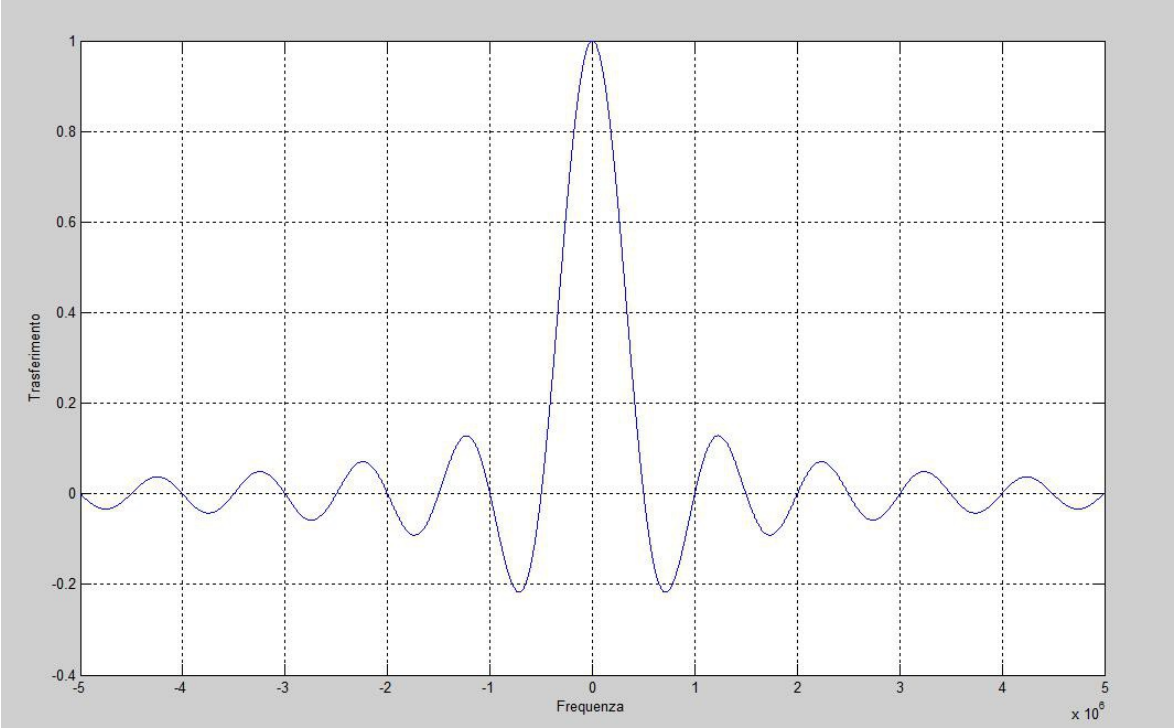



**Figura 3.18** Risposta all'impulso del gate integrator

Possiamo definire come funzione di trasferimento del nostro gate integrator la trasformata di questa particolare risposta all'impulso e come sappiamo la trasformata di un rettangolo è il seno cardinale




**Figuare 3.19** Risposta in frequenza del gate integrator

da cui è abbastanza complicata l'integrazione del rumore, risulta più semplice invece ragionare nel dominio del tempo.

Dato che il rumore è stazionario e bianco posso calcolarne il valore quadratico medio in uscita come prodotto della densità spettrale in ingresso per l'autocorelazione della risposta all'impulso del filtro valutata in 0.

$$n_{out}^2 = S_b \cdot k_{hh}(0) \qquad \textbf{(3.38}$$

Dato la densità spettrale di rumore è espressa in maniera unilatera dobbiamo tenere in considerazione un fattore 2.

$$n_{out}^2 = \frac{S_b \cdot k_{hh}(0)}{2} \qquad \textbf{(3.39}$$

L'autocorrelazione in 0 è l'integrale della risposta all'impulso al quadrato, quindi

$$k_{hh} = \int_0^T \frac{dt}{C^2} = \frac{T}{C^2} \quad \textbf{(3.40}$$

Possiamo perciò calcolare il rumore quadratico medio presente in uscita

$$n_{out}^2 = \frac{(50\text{fA})^2 \cdot k_{hh}(0)}{2} = (21\,\mu V)^2 \quad \textbf{(3.41}$$

che riportato in ingresso diventa

$$n_i^2 = \frac{n_{out}^2}{(T/C)^2} \quad \textbf{(3.42}$$

dove *T/C* è il trasferimento in continua che non è adimensionale ma ha le dimensioni di una resistenza

$$n_i^2 = (35\text{pA})^2 \quad \textbf{(3.43}$$

### 3.5.5 Trasmission gate

Ultimo componente che resta da scegliere è l'interruttore che dovrà cortocircuitare la capacità di retroazione.

Una possibile soluzione potrebbe essere un semplice pass-transistor come mostrato in **Figura 3.20**.

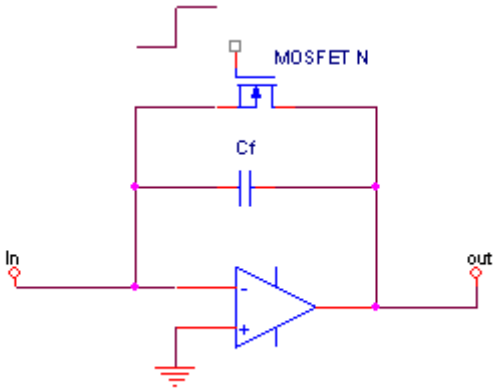
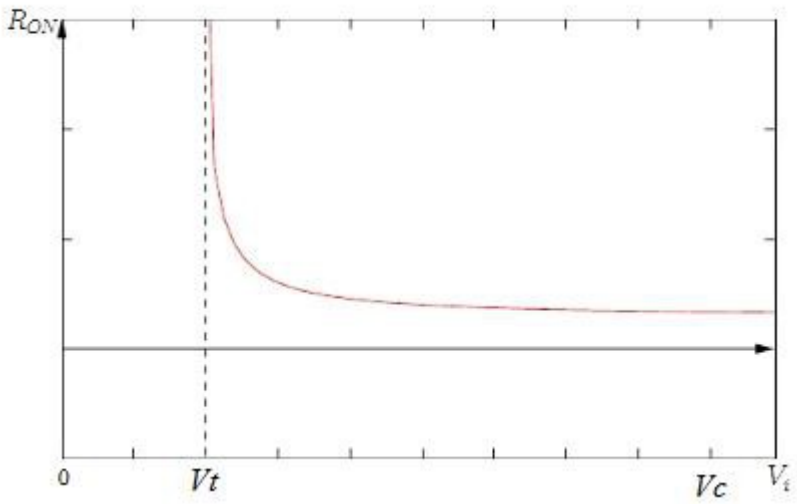
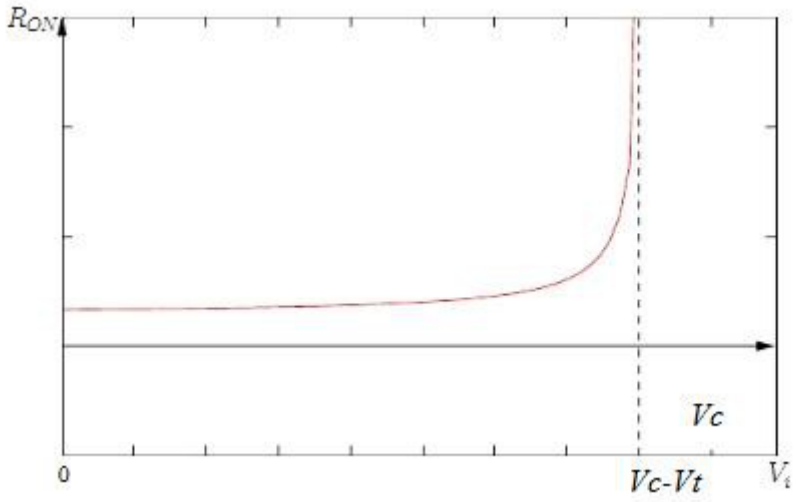



**Figura 3.20** Gate integrator con pass-transistor

che però presenterebbe una resistenza variabile al variare della tensione ai suoi capi (infatti la $R_{on}$ varia al variare della $V_{ds}$) e un trasferimento che arriverebbe verso l 'alto solo a $V_c - V_t$ nel caso di un nMOS e invece a $V_t$ verso il basso per un pMOS.



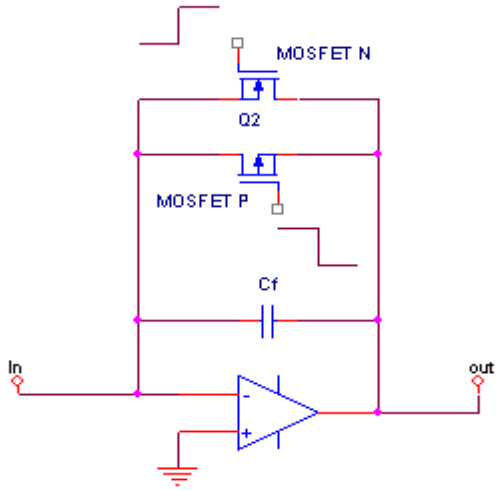

**Figuara 3.21** Anadamento della $R_{on}$ in funzione della tensione ai suoi capi in un nMOS




**Figuara 3.22** Anadamento della $R_{on}$ in funzione della tensione ai suoi capi in un pMOS

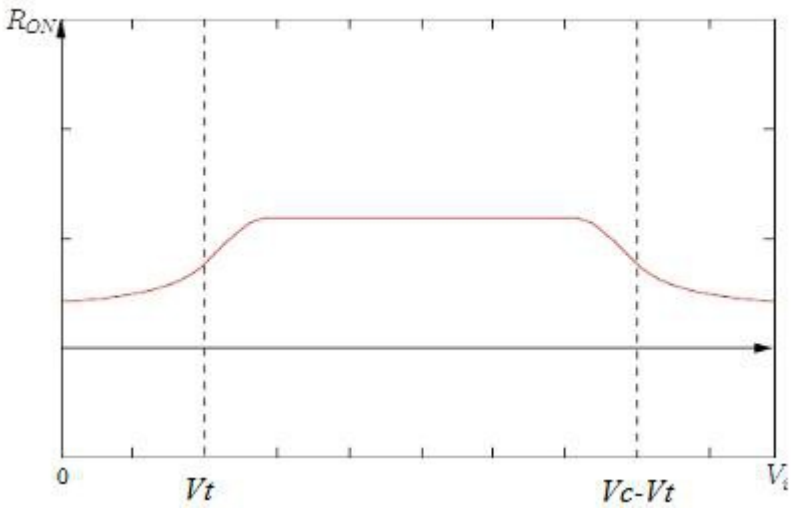
$V_c$ è la tensione di pilotaggio dell'interruttore che varia tra 0 e 3.3V determinata dal piccolo (nel caso invece fosse arrivata a 5V non sarebbe stato un problema dato che richiediamo il trasferimento di una tensione che va da 0 a 3.3V e quindi sarebbe stato sufficiente un nMOS).

Quindi si è deciso di sfruttare un trasmission gate




**Figuare 3.23** Gate integrator con trasmission gate

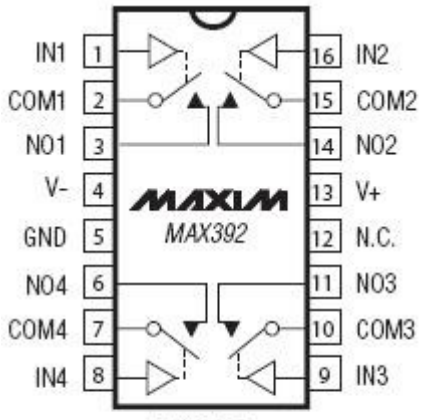
come si vede consiste in due transistor MOS, uno di tipo p e di tipo n, messi in parallelo per consentire una buona trasmissione di tutta la dinamica di tensione dato che dove lavora peggio uno lavora meglio l'altro. Infatti si può notare che anche la caratteristca della resistenza diventa piu costante.




**Figuara 3.24** Anadamento della $R_{on}$ in funzione della tensione ai suoi capi in un trasmission gate

A disposizione c'era il *MAX392* della Maxim il quale presenta una $R_{on} < 20\ \Omega$, in **Figura 3.25** è riportata la piedinatura.

Questo componente presenta internamente un circuito invertente che permette la realizzazione del segnale complementare per il MOS di tipo p, svincolandoci dal realizzarlo col DSP che ci comporterebbe un altro segnale con componenti ad alta frequenza che viaggerebbe nel nostro sistema comportando ulteriori disturbi.




| MAX392 | |
|---|---|
| LOGIC | SWITCH |
| 0 | OFF |
| 1 | ON |

**Figuare 3.25** Piedinatura MAX392

## 3.5.6 Campionamento

Dato che dovremo eserguire una conversione analogico-digitale, bisognerà tener conto del teorema del camionamento.

Il teorema del campionamento è uno dei teoremi base della teoria dei segnali e mette in relazione il contenuto di un segnale campionato con la frequenza di campionamento e le componenti minime e massime di frequenza del segnale analogico originale, definendo così la minima frequenza necessaria per campionare un segnale analogico senza perdere informazioni e per poter quindi ricostruire il segnale analogico tempo continuo originario**.**

In particolare il teorema afferma che, sotto opportune ipotesi, in una conversione analogico-digitale la minima frequenza di campionamento necessaria per evitare ambiguità e perdita di informazione nella ricostruzione del segnale analogico originario (ovvero nella riconversione analogico-digitale) con larghezza di banda finita e nota è pari al doppio della sua frequenza massima.

$$f_s \geqslant 2 f_{max} \qquad \textbf{(3.44}$$

Se non viene rispettato tale teorema, cioè si ha un sottocampionamento del segnale analogico nel dominio del tempo, nel dominio delle frequenze si ha la produzione di frequenze non proprie del segnale originario (*alias*) e viceversa dal dominio della frequenza al dominio del tempo producendo cioè una distorsione del segnale originario divenuto ora non più fedele.

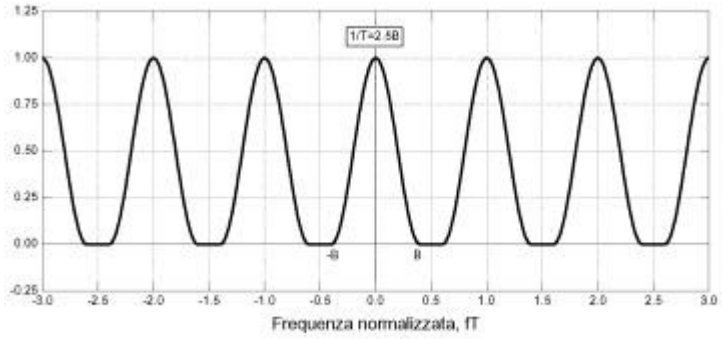



**Figura 3.26** Spettro segnale campionato rispettando il teorema di Shanon

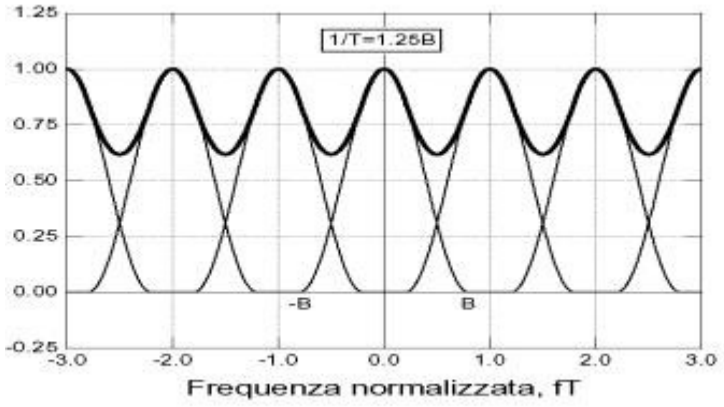



**Figura 3.27** Spettro segnale campionato non rispettando il teorema di Shanon

Per permettere al *Piccolo* di effettuare il processing del segnale campionato, non si può usare una frequenza di campionamento troppo elevata.
Nello stadio a transimpedenza si può considerare come frequenza massima quella del polo del filtro di primo ordine (220 kHz), quindi una buona frequenza di campionamento può essere 500kHz.
Nello stadio a gate integrato si ha che la stessa funzione di integrazione funge da filtro anti-alias, infatti assumendo che la durata dell'integrazione è circa pari al periodo di campionamento si vede un trasferimento in frequenza del seguente tipo nel grafico logaritmico.

(grafico)

Il seno cardinale ha la seguente formula

$$sinc(f T_s) = \frac{\sin(\pi f T_s)}{\pi f T_s} \quad \textbf{(3.45}$$

e dato che a numeratore il modulo del seno può essere al massimo pari ad uno lo si pone pari ad uno e si rischia al massimo di fare un aprossimazione per eccesso. Considerando come frequenza di taglio quella a -3 dB la si può ricavare cosi

$$\frac{1}{\pi f_{-3\mathrm{dB}} T_s} = \sqrt{2} \quad \textbf{(3.46}$$

$$f_{-3\mathrm{dB}} = 0.45 f_s < \frac{f_s}{2} \quad \textbf{(3.47}$$

# CAPITOLO 4

# Sezione digitale

## 4.1 Introduzione

In questo capitolo verrà descritta la parte digitale del triangolatore laser, ossia tutta la parte successiva al campionamento.
Si darà una prima descrizione del DSP coi relativi moduli utili al nostro scopo, dall'ADC alla perferica SPI, e in seguito verrà mostrata e descritta la programmazione di tali moduli.

## 4.2 Introduzione al microcontrollore

La famiglia di microcontrollori *F2806x Piccolo™* della *Texas Instruments* è provvista di:

I) un core *C28x™* e di un *Control Law Accelerator (CLA)* accoppiati con le periferiche di controllo altamente integrate a basso numero di pin.

II) regolatore interno di tensione single-rail.

III) Un ADC con due sample and hold che possono lavorare in parallelo, perfetti per campionare simultaneamente le nostre due tensioni provenienti dai precedenti stadi analogici, a 12 bit e con un range di conversione che va dai 0 ai 3.3 V ottimizzati per una bassa latenza.

IV) Otto unità ePWM sincronizzabili tra loro che sfrutteremo per sincronizzare il campionamento col resto del sistema oltre che per pilotare il LASER impulsato e il *trasmission gate* che cortocircuiterà la capacità di integrazione.

V) Una perifericazione di comunicazione SPI ad alta velocità che useremo per spedire i dati processati ad un DAC (o qualsiasi altra periferica esterna) per una visione real-time della misura.

VI) Un'unità di controllo PIE (Peripheral Interrupt Expansion) per la gestione degli interrupt.

Questi saranno i blocchi e le periferiche utili al triangolatore e di cui daremo una rapida descrizione per poter poi capire meglio certe scelte progettuali.

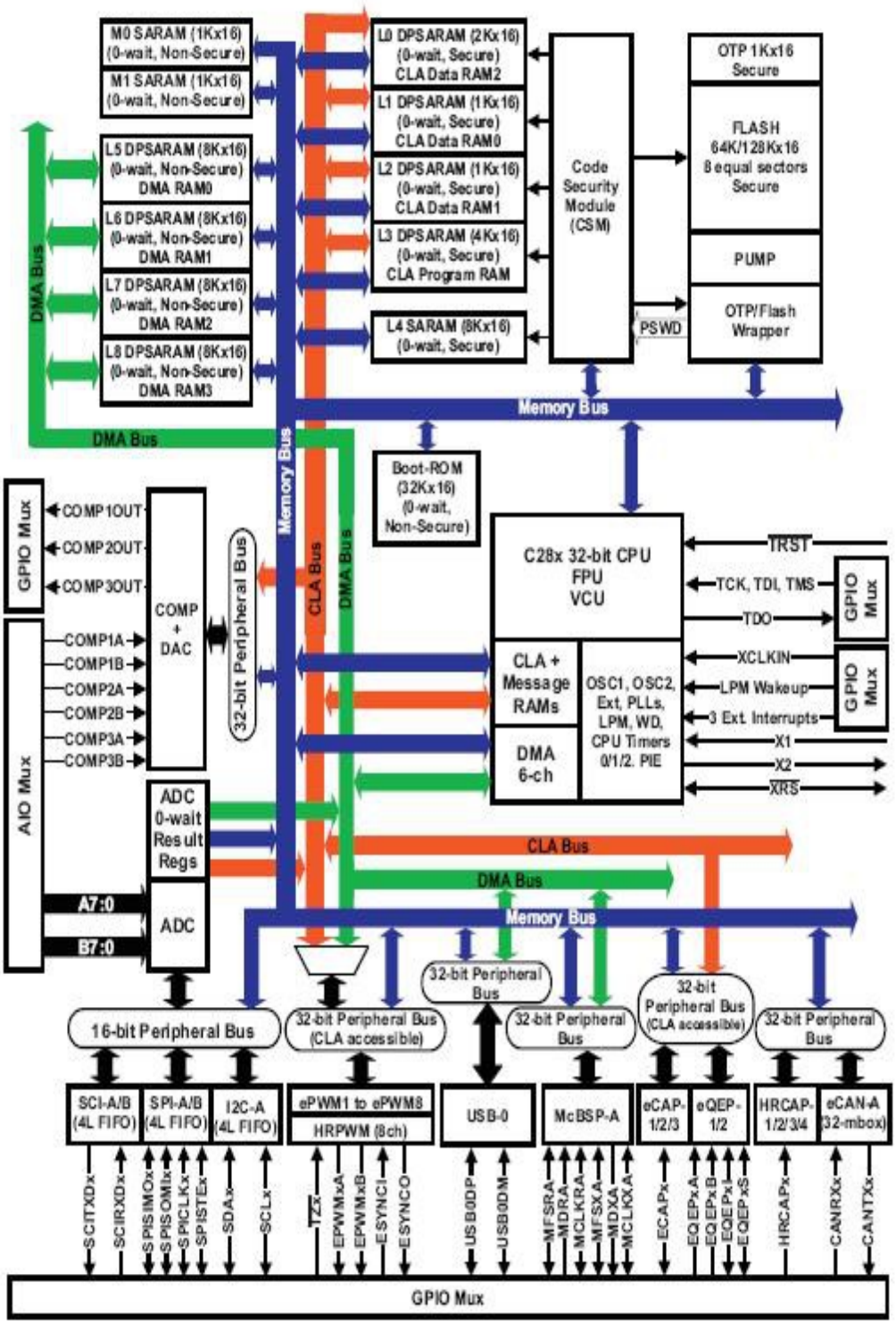



**Figura 4.1** Diagramma a blocchi del Piccolo

### 4.2.1 Analog to digital converter (ADC)

L'ADC del Piccolo è un 12 bit recyclic ADC di tipo SAR e pipeline. L'ADC SAR lavora per aprossimazioni successive, viene continuamente confrontato il valore digitale con quello analogico tramite un DAC ed un comparatore, partendo dal MSB fino ad arrivare al LSB. Inizialmente il bit posto in esame viene settato ad 1 e se l'uscita del comparatore è bassa tale bit viene lasciato alto, altrimenti viene posto uguale a 0.

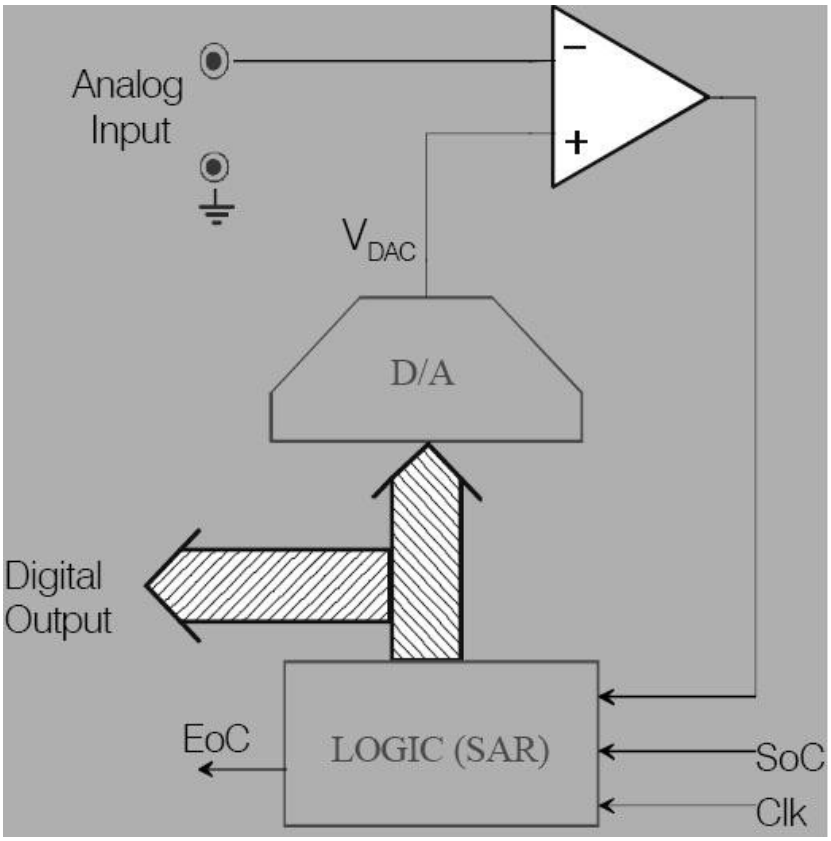



**Figura 4.2** Schema di funzionamento di un ADC SAR

Il tempo necessario per effettuare una conversione è

$$T_{conv} = \frac{n+1}{f_{clock}} \qquad \textbf{(4.1}$$

La circuiteria analogica di supporto è composta da un multiplexer analogico di front-end che gestisce 16 diversi canali d'ingresso, da due sample and hold che possono lavorare serialmente o in parallelo, l'unità di conversione e da un regolatore di tensione.
La circuiteria digitale invece è composta da sistema di conversioni programmabili, dai registri in cui vengono salvati i risultati della conversione, un interfaccia col mondo analogico, un interfaccia col BUS e un interfaccia con gli altri moduli on-chip.
La conversione può essere configurata per lavorare col riferimento di un bandgap interno oppure dato da una coppia di tensioni fornite dall'esterno (entro certi limiti).

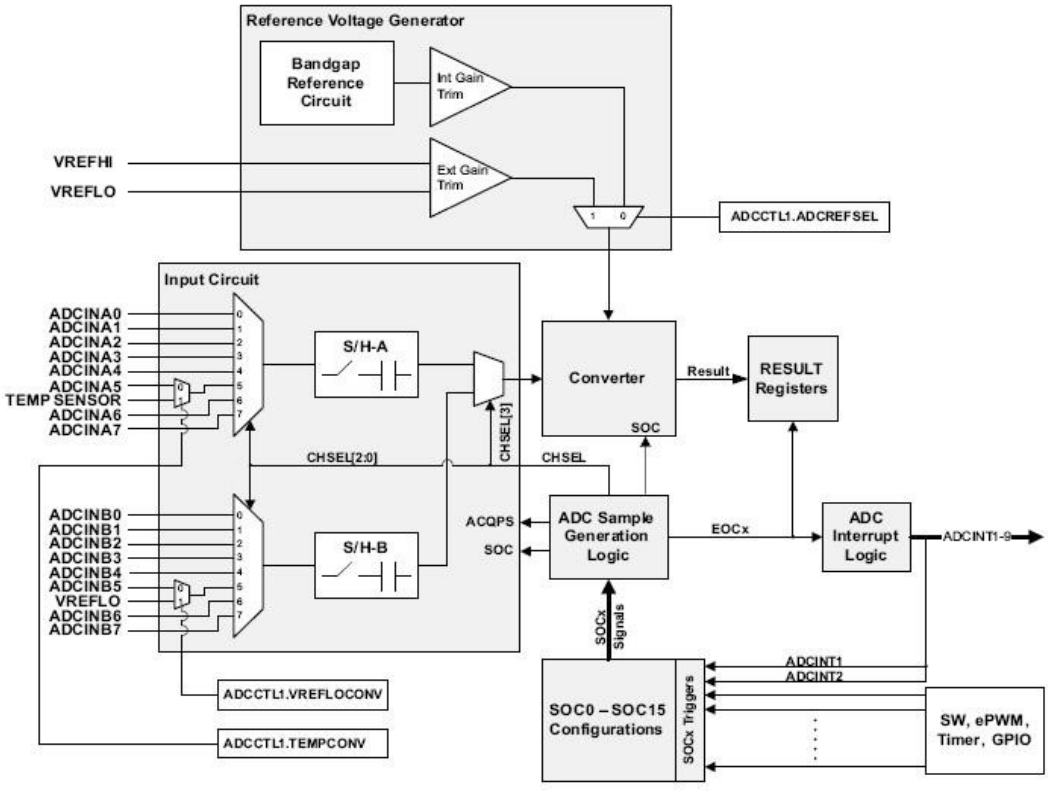



**Figura 4.3** Diagramma a blocchi dell'ADC

Le conversioni sono gestite dai SOC (*start of convertion*) che sono generati da un determinato evento di trigger. Questi SOC sono associati ad una singola conversione e ad un singolo canale quindi questi SOC sono caraterizzati dall'evento di trigger che lo innesca, dal canale associato e dalla finestra di conversione. I trigger che innescano il SOC possono arrivare via software, dal timer interno del Piccolo, da un particolare evento di interrupt o da un PWM. Quando il SOC è impostato per una conversione simultanea (nostro caso), il 4° bit che determina il canale viene ignorato e i due ingressi associati vengono accoppiati in maniera che il delay tra essi sia il minimo possibile. Sperimentalmente si è provato a far campionare ai 2 canali lo stesso segnale (una tensione variata a mano, quindi non con un elevata velocità) e si è visto che l'errore era al massimo di 2 LSB. Ogni SOC è indipendente dall'altro è può avere una qualsiasi configurazione di questi parametri.
A seconda del tipo di driver esterno si può variare la finestra del tempo di acquisizione per permettere al driver di trasferire efficientemente la carica all'ADC.

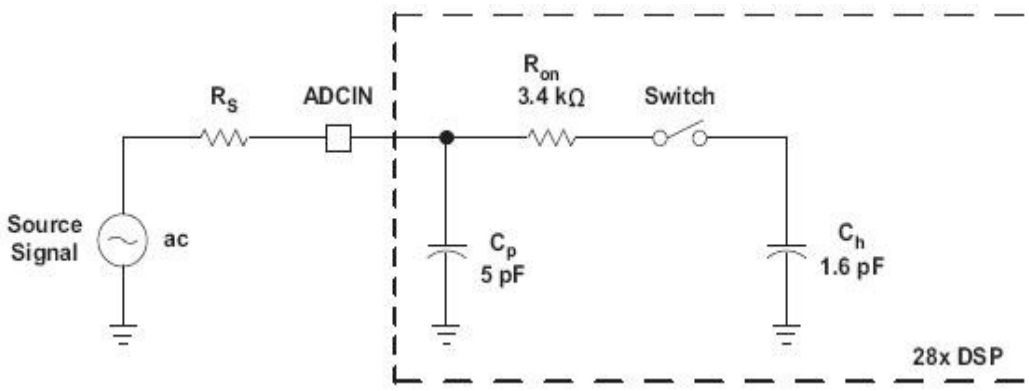



**Figura 4.4** Stadio d'ingresso dell'ADC

Esiste un particolare registro a 6 bit (ACQPS) in cui fissare il numero di cicli della finestra di conversione. Il numero scritto in questo registro è uno in meno del valore effettivo del numero di cicli che come limite minimo ha 7 (quindi nel registro ci sarà 6). Inoltre per effettuare la conversione l'ADC necessità di 13 colpi di clock.
Nel nostro caso abbiamo impostato la minima finestra di conversione (7 quindi) e la massima frequenza di clock (80 Mhz) ricaviamo perciò un tempo complessivo di

conversione pari a

$$T_{conv} = \frac{7 + 13\, cicli}{80\, MHz} = 250\, nsec \qquad \textbf{(4.2}$$

Ad ogni SOC è associato un EOC (*end of convertion)* che può generare un impulso sia all'inizio della conversione che alla fine. Ognuno di questi impulsi può essere impostato come trigger per scatenare un inerrupt all'interno dell'ADC. Questo interrupt poi può essere mandato al PIE per scatenare un interrupt generale oppure può sempliceme generare un segnale di flag per dire che la l'intterupt è avvenuto.
Per finire calcoliamoci il rumore dovuto alla quantizzazione del segnale analogica che risulterà essere

$$n_{ADC}^2 = \frac{LSB^2}{12} \qquad \textbf{(4.3}$$

Dove

$$LSB = \frac{3.3\, V}{2^{12}} = 805\, \mu V \qquad \textbf{(4.4}$$

quindi

$$n_{ADC}^2 = 232\, \mu V \qquad \textbf{(4.5}$$

### 4.2.2 Enhanced Pulse Width Modulator (ePWM)

La periferica ePWM è in grado di generare con un certo livello complessità onde rettangolari con un controllo o intervento minimo da parte della CPU.

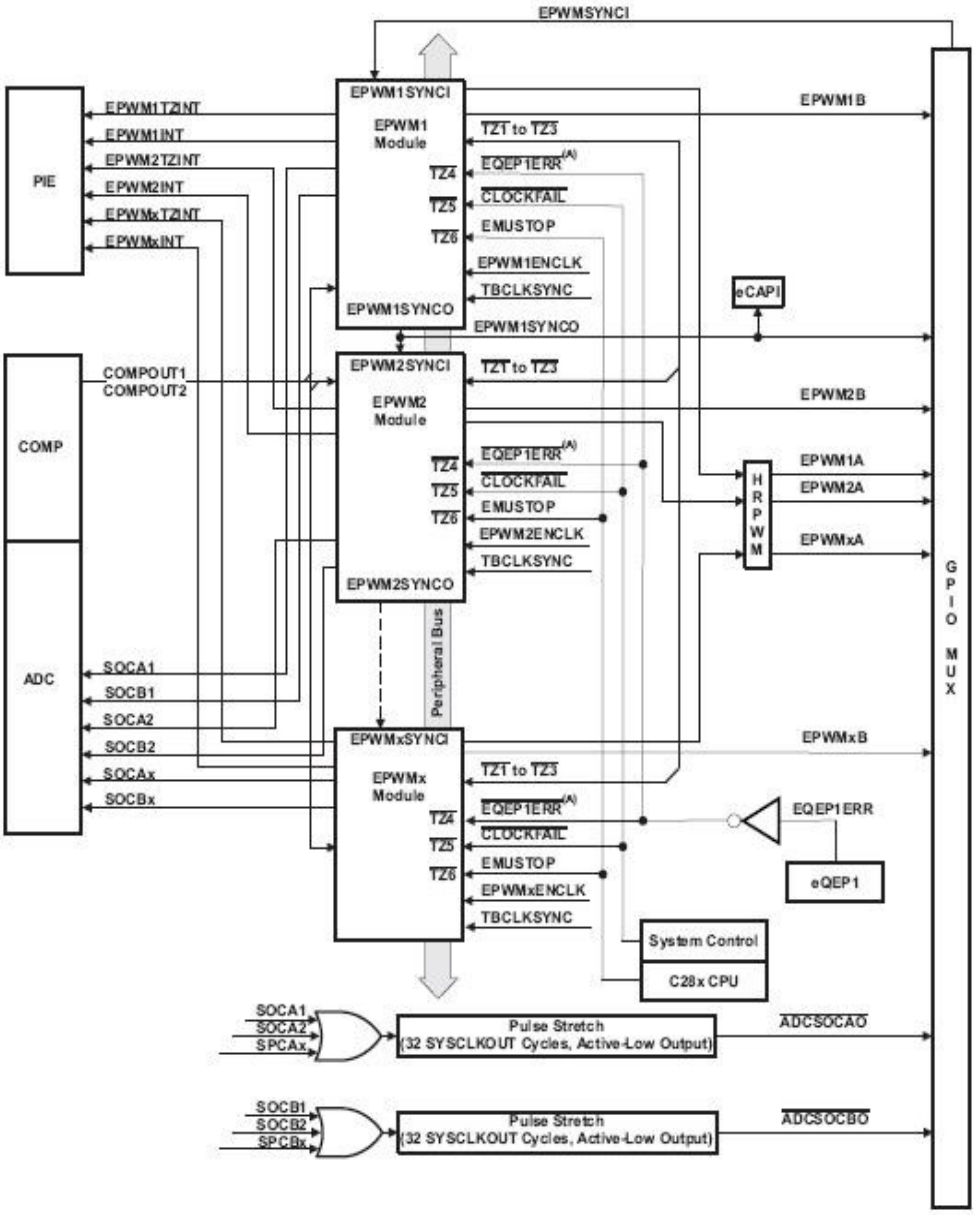



**Figura 4.5** Schema a blocchi ePWM

L'accoppiamento o la condivisione delle risorse non è concessa, invece il modulo ePWM è costruito da moduli più piccoli, ciascuno con le proprie risorse, in grado di cooperare insieme in modo da formare un sistema. Questo approccio modulare provvede a una migliore trasparenza della struttura aiutamdo l'utende ad una compressione più rapida.
Un submodulo ePWM è quindi un completo canale PWM con la disponibilità di due uscite EPWMxA e EPWMxB (x rappresenta il generico submodulo). Ciascun submodulo è identico all'altro, a meno di qualche eccezione che prevede un estensione hardware esterna per una maggior precisione dell'uscita PWM.
I submoduli PWM sono collegati tra di loro tramite uno schema di sincronizzazione del clock che permette un lavoro come sistema quando necessario.
Ognuno di questi submoduli è diviso ulteriormente in altri 8 submoduli, ognuno dei quali svolge compiti precisi: *Time-base*, *counter-compare*, *Action-qualifier*, *Dead-band*, *PWM-chopper*, *Trip-zone*, *Event-trigger*, *Digital-compare*. Daremo una rapida spiegazione solo delle funzioni necessarie ai submoduli utili per il triangolatore.
Il *Time-base* determina la frequenza di ciascun evento, gestisce il sincronismo con gli altri blocchi PWM, mantiene la relazione di fase con gli altri blocchi PWM, determina il tipo di

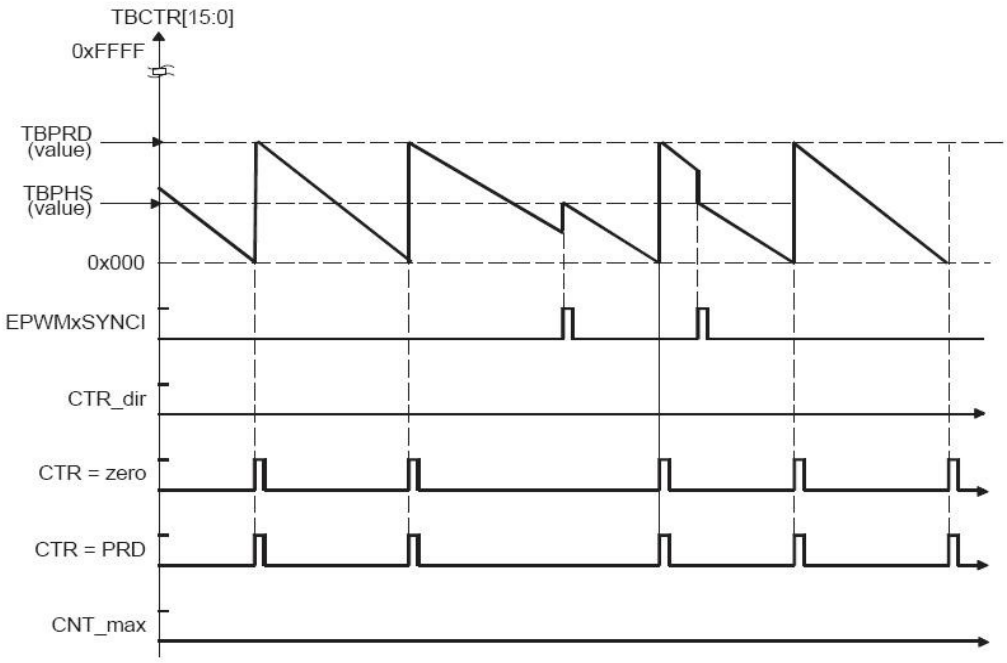



**Figura 4.6** Time-base down count mode

conteggio del *time-base counter* se *count-up, count-down*, o *count-up-and-down*, genera un evento se il *time-base counter* è uguale ad un specificato periodo oppure è uguale a zero e infine configura il rate del *Time-base clock* che sarà una frazione del *system clock*.
Il *Counter-compare* ha come ingresso il *time-base counter* che viene continuamente confrontato coi valori di *counter-compare A* e di *counter-compare B*. Quando viene verificata l'uguaglianza il *counter-compare* genererà un evento.

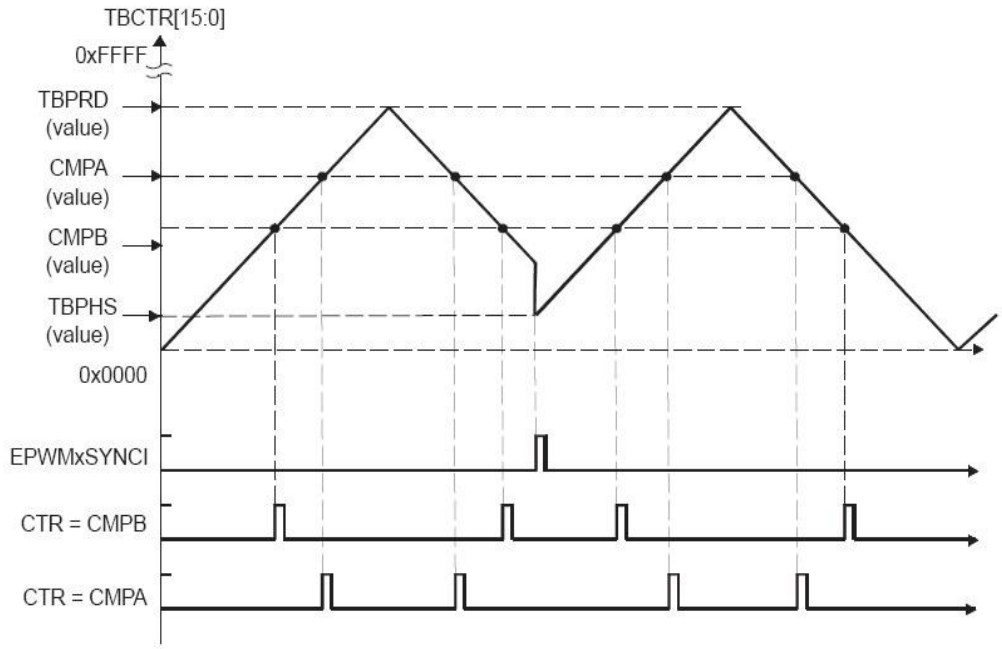



**Figura 4.7** Eventi di Counter-Compare in Up-Down-Count Mode

L'*action-qualifier* ha il ruolo piu importante nella costruzione delle forme d'onda PWM, decide quale degli eventi verrà convertito in quale azione per produrre le onde switchata alle uscite *EPWMxA* e *EPWMxB*. Le possibili azioni sono *set high*, *clear low*, , *toggle* e *do nothing*.

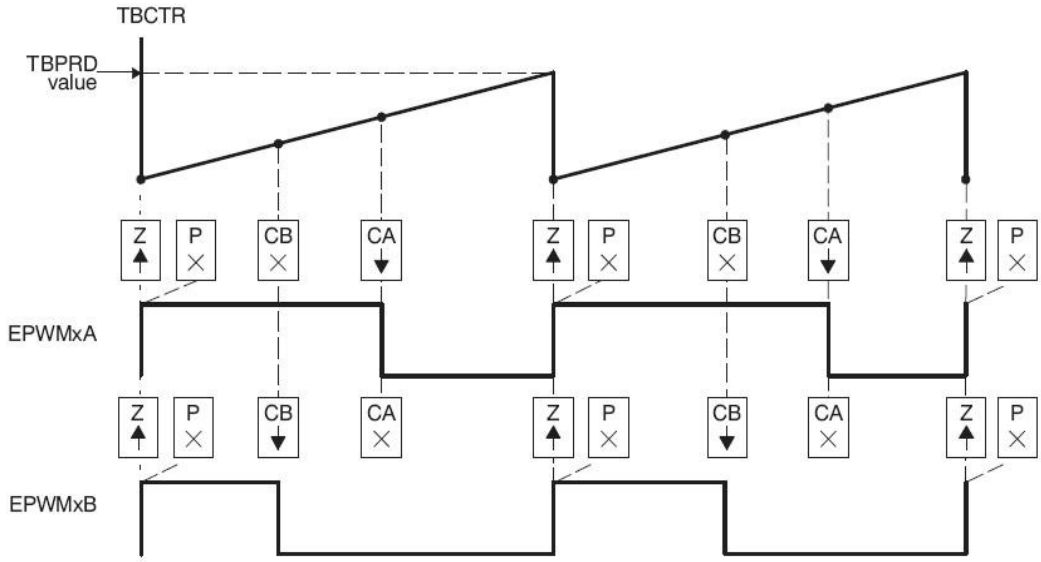



**Tabella 4.8** Forme d'onda indipendenti di EPWMxA e EPWMxB

Infine l'ultimo submodulo che ci interessa è l'*event-trigger* che prende in ingresso gli eventi generati nel *Time-base* o nel *Counter-compare* per generare segnali di flag, forzare l'*interrupt* o un SOC nell'ADC. Eventualmente può contare gli eventi tramite un *event counter* per determinare quanti eventi sono necessari per generare un trigger.

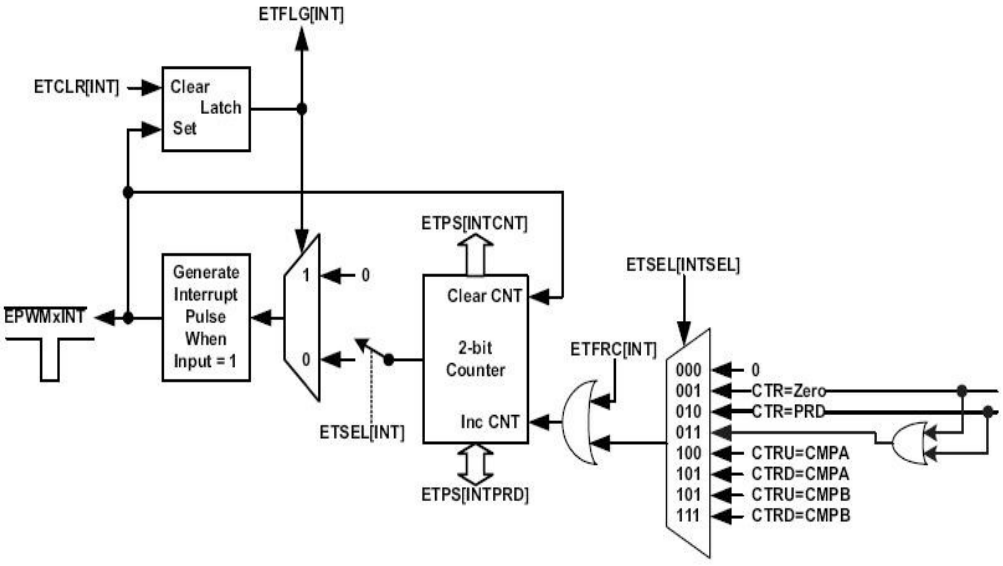



**Figura 4.9** Generatore di eventi di trigger per l'interrupt

### 4.2.3 Peripheral Interrupt Expansion (PIE)

La *Peripheral Interrupt Expansion* (PIE) multiplexa numerose fonti di *interrupt* in un insieme ridotto di ingressi. Il blocco è PIE in grado di supportare 96 interrupt individuali che sono raggruppati in blocchi di otto. Ogni gruppo alimenta12 linee principali di *interrupt* (INT1 a INT12). Ciascuno dei 96 interrupt è sostenuto dal suo vettore memorizzato in un blocco di RAM dedicato che è possibile modificare. La CPU impiega nove cicli di clock per prendere il vettore apropriato e salvare i registri critici, pertanto la CPU può rispondere rapidamente agli eventi di interrupt. La priorità degli interrupt è controllata via hardware e software. Ogni singolo interrupt può essere abilitato o disabilitato all'interno del blocco PIE.

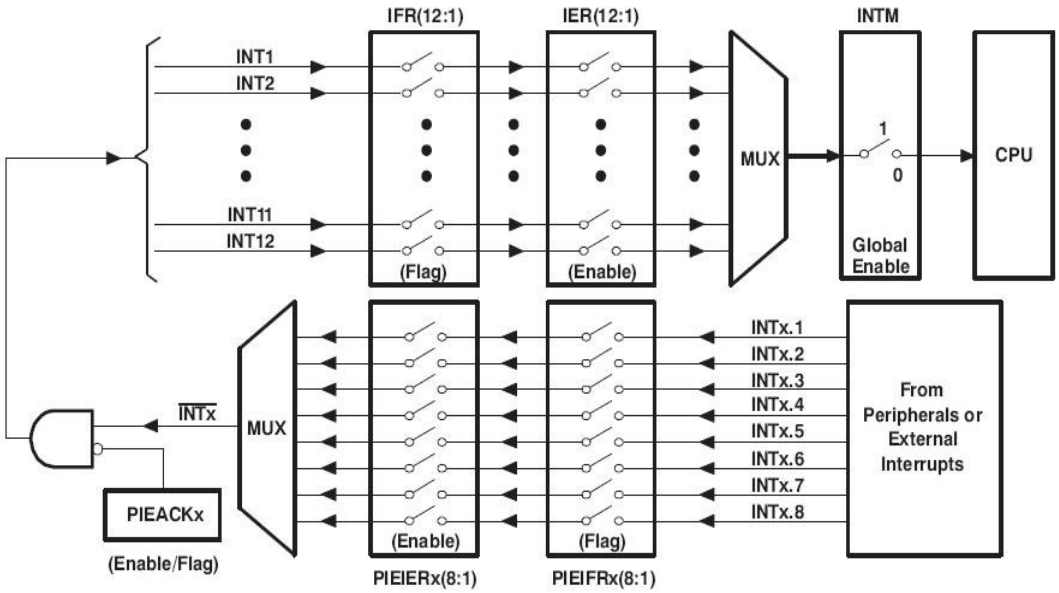



**Figura 4.10** Schema di multiplexing per gli interrupt

A livello periferico, quando avviene un *interrupt*, il corrispondente *interrupt flag* (IF) all'evento viene settato. Se il corrispondente *interrupt enable* (IE) è settato, la periferica genera in *interrupt request* al controllore PIE, altrimenti l'IF rimane settato finchè non verrà cancellato via software. Se l'*interrupt* venisse abilitato in un secondo momento e l'IF è ancora settato viene generata l'*interrupt request.*Gli interrupt poi delle periferiche (ad esempio l'ADC) dovranno essere cancellati manualmente.
Il blocco PIE poi multiplexa otto *interrupt* periferici in un solo *interrupt.* Questi *interrupt* sono poi divisi in 12 gruppi e l'interrupt di ciascun gruppo è poi multiplexato in un solo CPU *interrupt*.
Una volta che la richiesta viene mandata al blocco PIE il corrispondente bit di flag viene settato. Se anche l'enable fosse settato allora si controlla il corrispondente bit PIEACK per determinare se la CPU è pronta per ricevere l'*interrupt*, se non lo fosse la richiesta rimane in attesa.

### 4.2.4 Serial Peripheral Interface (SPI)

La *Serial Peripheral Interface* è un porta di comunicazione Input/Output ad alta velocità sincrona e seriale che permette ad un bit-stream di dati di una programmata lunghezza di essere ricevuto o mandato ad un rate programmato. Applicazioni tipiche includono la comunicazione con altri DSP oppure periferiche esterne di espansione come display, registri, ADC e DAC (nostro caso).

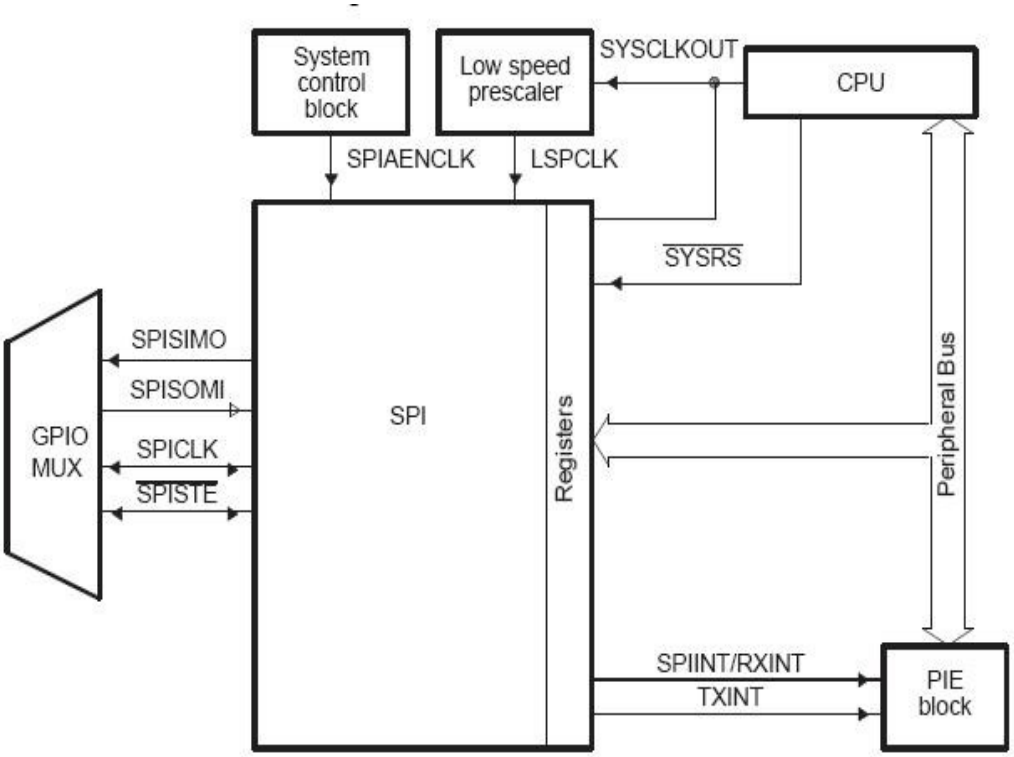



**Figura 4.11** Interfaccia SPI-CPU

L'SPI è composta da 4 pin:

• SPISOMI: SPI slave-output/master-input pin

• SPISIMO: SPI slave-input/master-output pin

• SPISTE: SPI slave transmit-enable pin

• SPICLK: SPI serial-clock pin

e può operare sia da *master* che da *slave*, nel primo caso ha un rate massimo di trasmissione di LSPCLK/4 mentre nell'altro di LSPCLK/8, dove LSPCLK è la frequenza di clock interna all'SPI.

Noi lo faremo operare da *master* dato che dovrà comandare il DAC che darà poi l'uscita analogica proporzionale allo spostamento.

Il master inizia il trasferimento dei dati, inviando il segnale SPICLK. Sia per lo slave che per il master, i dati vengono spostati fuori dai registri a scorrimento sul fronte del SPICLK

e bloccati nel registro a scorrimento sul fronte opposto. Se il bit PHASE CLOCK è alto, i dati vengono trasmessi e ricevuti un semiciclo prima della transizione del SPICLK. Di conseguenza, entrambi i controller invieranno e riceveranno i dati simultaneamente. Il software applicativo determina se i dati

dati sono significativi o fittizi.

Ci sono tre possibili metodi per la trasmissione dei dati:

• Master invia i dati; slave invia dati fittizi.

• Master invia i dati; slave invia i dati.

• Master invia i dati fitizi; slave invia i dati.

Il master può iniziare il trasferimento dei dati in qualsiasi momento, perché controlla il segnale SPICLK. Il software, tuttavia, determina in che modo il *master* capirà quando lo slave è pronto per la trasmissione dati.

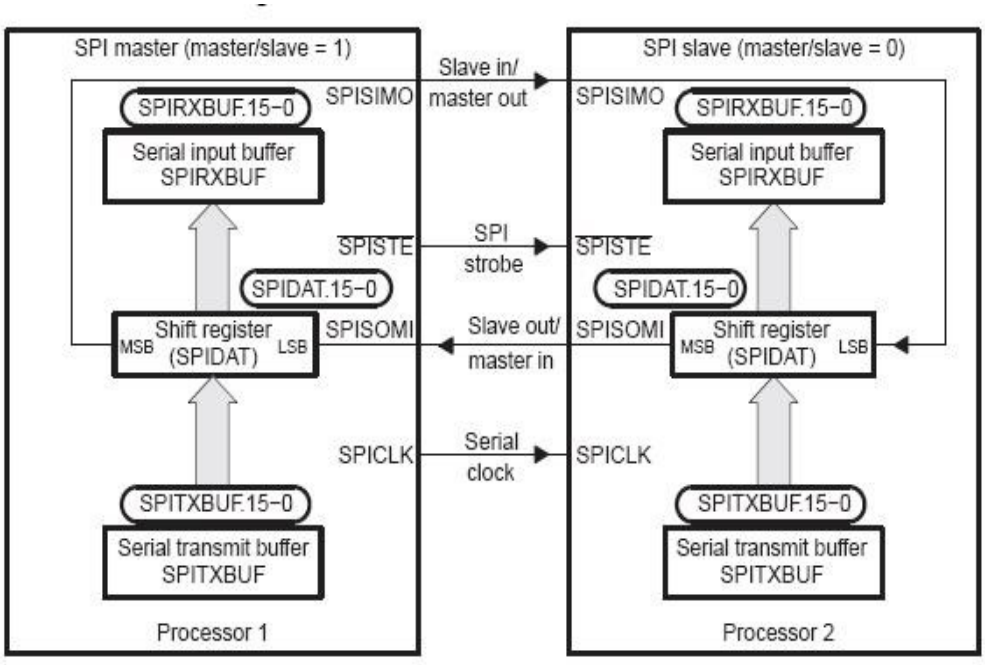



**Figura 4.12** Connessione Master-Slave

I dati scritti nel SPIDAT o nel SPITXBUF vengono trasmessi sul pin SPISIMO e per primo viene trasmesso il MSB (Most significant bit). Allo stesso tempo i dati, ricevuti vengono scritti tramite il SPISOMI nella LSB (Least significant bit) del SPIDAT. Quando il numero di bit selezionato è stato trasmesso, i dati ricevuti vengono
trasferiti al SPIRXBUF (buffer di ricezione) per essere letti dalla CPU.
Quando il numero specificato di bit di dati è stata shiftato attraverso lo SPIDAT, si verificano i seguenti eventi:

• I contenuti dello SPIDAT vengono trasferiti nel SPIRXBUF.

• SPI FLAG INT bit è impostato a 1.

• Se ci sono dati validi nel SPITXBUF (buffer di trasmissione), come indicato dal bit TXBUF FULL in SPISTS, questi dati vengono trasferiti nella SPIDAT e vengono trasmessi, altrimenti l'SPICLK si arresta dopo che tutti i bit sono stati shiftati fuori dallo SPIDAT.

• Se il bit INT SPI ENA è impostato a 1, viene generato un interrupt.

In una tipica applicazione, il pin SPISTE serve come un chip-enable pin per un dispositivo slave SPI. Questo pin viene abbassato dal master prima della trasmissione dei dati allo slave e viene rialzato dopo che la trasmissione è stata completata.
In modalità master, se il bit TRIWIRE è settato, consentendo a 3-wire SPI mode, SPISIMOx diventa il bi-direzionale SPIMOMIx (SPI out master, master in) pin, e SPISOMIx non viene più utilizzato dalla SPI. In modalità slave, se viene settato il bit TRIWIRE, SPISOMIx diventa il bi-direzionale SPISISOx (slave in SPI, slave
out) pin, e SPISIMOx non viene più utilizzato dalla SPI.

## 4.3 Progettazione del software

Terminata la descrizione delle periferiche del *Piccolo* utili al nostro progetto del telemetro vediamo effettivamente come sono state impiegate e l'evoluzione lungo le fasi di progetto.
Per una più semplice comprensione illustriamo prima una visione generale per poi andare nello specifico di ciascuna periferica.

### 4.3.1 Obiettivo

Al termine dei primi front-end analogici, abbiamo due segnali di tensione di ampiezza 0-3.3 V, sia che siano il risultato della *transimpedenza* che del *gate integrator*.
Queste tensioni andranno campionate simultaneamente per non avere errori e con una frequenza di campionamento ben fissata e la più stabile possibile per non creare distorsioni che genererebbero componenti spurie di ampiezza crescente all'aumentare della frequenza d'ingresso.
Una volta eseguito il campionamento si disporrà di due valori quantizzati che dovranno poi essere processati dal microcontrollore. Tale processamento consiste nell'eseguire la loro differenza, normalizzata rispetto alla somma dato che la potenza in ingresso non è costante e può variare in base a diversi fattori quali la potenza di emissione del fascio LASER che col tempo può subire variazioni e derive, il coefficiente di assorbimento dell'ambiente che può variare a seconda di dove viene effettuata la misusura e infine la stessa tecnica di triangolazione comporta una variazione della potenza incidente dato che a seconda della distanza si ha che il fascio lasere esegue percorsi piu o meno lunghi che quindi ne comportano un attenuazione più o meno rilevante.
Chiamando $x_1$ e $x_2$ i due valori risultanti dal campionamento delle tensioni abbiamo

$$y = \frac{x_1 - x_2}{x_1 + x_2} \qquad \textbf{(4.6}$$

dove *y* sarà il valore proporzionale allo spostamento.
Avere a disposizione questi segnali campionati e quantizzati da la disponibilità ad eseguire un ulteriore processing, come ad esempio un ulteriore filtraggio, del segnale a costo zero. Si deciderà perciò di realizzare un filtro di *Butterworth* per eliminare ulteriormente il rumore fuori banda. Lo si realizzerà di ordine più elevato possibile, cercando però di rispettare i tempi di campionamento dato che un filtro di ordine elevato necessità di un maggior tempo computazionale e per una visualizzazione della misura real time è necessario che il processing dei due campioni sia terminato prima che vengano acquisiti i due campioni successivi.
In fine, dopo avere ottenuto il segnale proporzionale allo spostamento e dopo averlo filtrato, compito finale sarà quello di mandarlo ad una periferica esterna di visualizzazione tramite l'SPI.

### 4.3.2 Impostazione dell'ADC

Primo step quindi abbiamo detto che è l'aquisizione tramite campionamento dei due segnali di tensione in uscita dai primi stadi analogici.
Per effettura il campionamento è necessario configurare l'ADC.

```
EALLOW;
AdcRegs.ADCCTL1.bit.ADCBGPWD  = 1;
AdcRegs.ADCCTL1.bit.ADCREFPWD = 1;
AdcRegs.ADCCTL1.bit.ADCPWDN   = 1;
AdcRegs.ADCCTL1.bit.ADCENABLE = 1;
AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;

EDIS;
```

La funzione EALOW serve a consentire l'accesso a registri che altrementi sarebbere

protetti per dare un maggior grado di sicurezza a registri che andrebbero modificati con più cura rispetto ad altri e permettere un corretto funzionamento del *Piccol*. EDIS termina la possibilità ad accedere a tali registri, è sempre buona norma usare queste funzioni dove necessario e non attivarli all'inizio e alla fine di tutto per preservare il *Piccolo* da errori.
In questa porzione di codice si alimenta la circuiteria del bandgap dell'ADC (ADCBGPWD), viene alimentata la circuiteria dei buffer per i riferimenti (ADCREFPWD), viene alimentata tutta la circuiteria analogica dell'ADC ad esclusione del bandgap e dei buffer (ADCPWDN), viene abilitato l'ADC, cosa consigliata di fare appena dopo aver alimentato tutta la circuiteria associata infatti (ADCENABLE) e infine viene scelto il bandgap interno come riferimento per l'ADC (ADCREFSEL), infatti all'ADC volendo si possomo mamdare dei riferimenti esterni.

```
    GpioCtrlRegs.AIOMUX1.bit.AIO2 = 2;
    GpioCtrlRegs.AIOMUX1.bit.AIO4 = 2;
    GpioCtrlRegs.AIOMUX1.bit.AIO6 = 2;
    GpioCtrlRegs.AIOMUX1.bit.AIO10 = 2;
    GpioCtrlRegs.AIOMUX1.bit.AIO12 = 2;
    GpioCtrlRegs.AIOMUX1.bit.AIO14 = 2;
```

In questa porzione di codice vengono impostati tutti i pin associati agli ADC solo come input che altrimenti resterebbero dei *General-Purpose Input/Output*, ossia dei pin generici. La maggior parte dei pin delle varie periferiche va impostata.

```
void AdcOffsetSelfCal()
{
    Uint16 AdcConvMean;
    EALLOW;
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;
    AdcRegs.ADCCTL1.bit.VREFLOCONV = 1;
    AdcChanSelect(13);
    AdcRegs.ADCOFFTRIM.bit.OFFTRIM = 80;
```

```
    AdcConvMean = AdcConversion();
    AdcRegs.ADCOFFTRIM.bit.OFFTRIM = 80 - AdcConvMean;
    AdcRegs.ADCCTL1.bit.VREFLOCONV = 0;
    EDIS;
}
```

Questa funzione ricalibra l'offset dell'ADC mediante la conversione di VRFLO tramite l'ADC e modificando il valore nel registro ADCOFFTRIM. VRFLO è campionato tramite un MUX interno che connette VRFLO ad A5 senza sacrificare un pin esterno. Questa funzione chiama altre due funzioni: AdcChanSelect (canale) che seleziona il canale da convertire e AdcConversion () che avvia diverse conversioni e ne restituisce la media.

```
void AdcChanSelect(Uint16 ch_no)
{
    AdcRegs.ADCSOC0CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC1CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC2CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC3CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC4CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC5CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC6CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC7CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC8CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC9CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC10CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC11CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC12CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC13CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC14CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC15CTL.bit.CHSEL= ch_no;
}



Uint16 AdcConversion(void)
{
    Uint16 index, SampleSize, Mean, ACQPS_Value;
    Uint32 Sum;

    index       = 0;
    SampleSize  = 256;
    Sum         = 0;
    Mean        = 999;
```

```
ACQPS_Value = 6;
AdcRegs.ADCSOC0CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC1CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC2CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC3CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC4CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC5CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC6CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC7CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC8CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC9CTL.bit.ACQPS  = ACQPS_Value;
AdcRegs.ADCSOC10CTL.bit.ACQPS = ACQPS_Value;
AdcRegs.ADCSOC11CTL.bit.ACQPS = ACQPS_Value;
AdcRegs.ADCSOC12CTL.bit.ACQPS = ACQPS_Value;
AdcRegs.ADCSOC13CTL.bit.ACQPS = ACQPS_Value;
AdcRegs.ADCSOC14CTL.bit.ACQPS = ACQPS_Value;
AdcRegs.ADCSOC15CTL.bit.ACQPS = ACQPS_Value;


AdcRegs.INTSEL1N2.bit.INT1E = 1;
AdcRegs.INTSEL1N2.bit.INT2E = 1;


AdcRegs.INTSEL1N2.bit.INT1CONT = 0;
AdcRegs.INTSEL1N2.bit.INT2CONT = 0;

AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;

AdcRegs.INTSEL1N2.bit.INT1SEL = 6;
AdcRegs.INTSEL1N2.bit.INT2SEL = 14;


AdcRegs.ADCINTSOCSEL1.bit.SOC0  = 2;
AdcRegs.ADCINTSOCSEL1.bit.SOC1  = 2;
AdcRegs.ADCINTSOCSEL1.bit.SOC2  = 2;
AdcRegs.ADCINTSOCSEL1.bit.SOC3  = 2;
AdcRegs.ADCINTSOCSEL1.bit.SOC4  = 2;
AdcRegs.ADCINTSOCSEL1.bit.SOC5  = 2;
AdcRegs.ADCINTSOCSEL1.bit.SOC6  = 2;
AdcRegs.ADCINTSOCSEL1.bit.SOC7  = 2;
AdcRegs.ADCINTSOCSEL2.bit.SOC8  = 1;
AdcRegs.ADCINTSOCSEL2.bit.SOC9  = 1;
AdcRegs.ADCINTSOCSEL2.bit.SOC10 = 1;
AdcRegs.ADCINTSOCSEL2.bit.SOC11 = 1;
AdcRegs.ADCINTSOCSEL2.bit.SOC12 = 1;
AdcRegs.ADCINTSOCSEL2.bit.SOC13 = 1;
AdcRegs.ADCINTSOCSEL2.bit.SOC14 = 1;
AdcRegs.ADCINTSOCSEL2.bit.SOC15 = 1;

DELAY_US(ADC_usDELAY);
```

```
    AdcRegs.ADCSOCFRC1.all = 0x00FF;

    while( index < SampleSize ){


        while (AdcRegs.ADCINTFLG.bit.ADCINT1 == 0){}
        AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
        Sum += AdcResult.ADCRESULT0;
        Sum += AdcResult.ADCRESULT1;
        Sum += AdcResult.ADCRESULT2;
        Sum += AdcResult.ADCRESULT3;
        Sum += AdcResult.ADCRESULT4;
        Sum += AdcResult.ADCRESULT5;
        Sum += AdcResult.ADCRESULT6;
        Sum += AdcResult.ADCRESULT7;


        while (AdcRegs.ADCINTFLG.bit.ADCINT2 == 0){}
        AdcRegs.ADCINTFLGCLR.bit.ADCINT2 = 1;
        Sum += AdcResult.ADCRESULT8;
        Sum += AdcResult.ADCRESULT9;
        Sum += AdcResult.ADCRESULT10;
        Sum += AdcResult.ADCRESULT11;
        Sum += AdcResult.ADCRESULT12;
        Sum += AdcResult.ADCRESULT13;
        Sum += AdcResult.ADCRESULT14;
        Sum += AdcResult.ADCRESULT15;

        index+=16;

    }


    AdcRegs.INTSEL1N2.bit.INT1E = 0;
    AdcRegs.INTSEL1N2.bit.INT2E = 0;

    Mean = Sum / SampleSize;

    return Mean;

}
```

Restano adesso da impostare adesso gli ultimi registri, nei quali ce ne sarà qualcuno meno di routine e che comporterà finalmente una certa scelta progettuale.

Bisogna inanzitutto pensare a che tipo di trigger usare, quale può essere più adatto al nostro scopo. A disposizione abbiamo i seguenti tipi di trigger:

• Software

• CPU Timers 0/1/2 interrupts

• XINT2

• ePWM1-7

Usare un trigger mandato via software non è preciso dato che l'esecuzione delle istruzioni non è costante, quindi può causare una variazione di tempo tra un campionamento e il successivo causando sistorsioni e la nascita di componenti armoniche spurie.

I Timers della CPU sono invece molto più stabili in frequenza e potrebbero essere una buona sorgente di trigger.

Un interrupt esterno lo escludiamo subito perchè ciò implicherebbe la progettazione di altri circuiti esterni di temporizzazione, cosa inutile dato che il Piccolo ne possiede già di suoi inutilizzati.

L'ePWM potrebbe essere un'altra buona sorgente da utilizzare come trigger, e se nella progettazione analogica ci fossimo fermati allo stadio a transimpedenza ne risulterebbe una scelta arbitraria tra ePWM e uno dei Timers della CPU.

Dato che però nello stadio a gate integrator necessitiamo di segnali PWM che pilotino l'accensione del LASER e del trasmission gate, sincroni tra loro ci fa ricadere la scelta sull'ePWM come sorgente di trigger. Infatti anche il campionamento dovrà essere sincrono e l'unità ePWM ci permette di avere forme d'onda PWM differenti tra loro ma con una relazione di fase programmabile.

```
AdcRegs.ADCCTL2.bit.ADCNONOVERLAP = 1;
AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;
AdcRegs.INTSEL1N2.bit.INT1E     = 1;
AdcRegs.INTSEL1N2.bit.INT1CONT  = 0;
AdcRegs.INTSEL1N2.bit.INT1SEL   = 1;
AdcRegs.ADCSAMPLEMODE.bit.SIMULEN0 = 1;
```

Con questo set di istruzioni disabilitiamo l'overlap, ossia la possibilità di poter effettuare un nuovo campionameto mentre una conversione è ancora in corso (ADCNONOVERLAP), decidiamo che l'impulso di interrupt avvenga alla fine della conversione, ossia quando l'ADC sta scrivendo il valore nel registro e non all'inizio del campionamento (INTPULSEPOS), abilitiamo l'interrupt interno dell'ADC (INT1E), facciamo in modo che gli impulsi di interrupt avvengano solo se prima viene cancellato il bit di flag via software

(INT1CONT), decidiamo di scatenare l'interrupt all'EOC1 (INT1SEL) e in fine accoppiamo il SOC0 e SOC1 per fargli lavorare in maniera simultanea (SIMULEN0).

```
AdcRegs.ADCSOC0CTL.bit.CHSEL    = 0;
AdcRegs.ADCSOC0CTL.bit.TRIGSEL  = 5;
AdcRegs.ADCSOC0CTL.bit.ACQPS    = 6;
```

Queste saranno le ultime impostazioni dell'ADC, poi sarà pronto, con CHSEL scegliamo ADCINA0 come canale analogico d'ingresso e di conseguenza, avendo attivato il modo simultaneo anche ADCINB0, selezioniamo come trigger per le SOC0 e SOC1 l'uscita EPWM1A (TRIGSEL) e per finire impostiamo la finestra di acquisizione al suo tempo minimo, ossia 7 cicli di clock (ACQPS).
Verifichiamo che quest'ultima impostazione sia compatibile con gli operazionali dati. 7 cicli di clock comporta che la $C_{hold}$ ha a disposizione

$$T_{sample} = \frac{7\,cicli}{80\,MHz} = 87.5\,nsec \qquad \textbf{(4.7}$$

In uscita dall'*OPA355* abbiamo a disposizione una corrente massima di 60 mA e una $R_{out}$ ad anello chiuso pari a 20 mΩ completamente trascurabile rispetto alla $R_{on}$ come si può vedere dallo schema elettrico. Possiamo considerare che la 5 pF si carichi istantaneamente, quindi la costante di tempo di carica del circuito risulta quindi essere

$$\tau = 3.4\,k\Omega \cdot 1.6\,pF = 5.28\,nsec \qquad \textbf{(4.8}$$

che porta un errore massimo pari a

$$\varepsilon = 3.3\,V \cdot e^{\left(\frac{-87.5}{5.28}\right)} = 200\,nV \qquad \textbf{(4.9}$$

che risulta molto minore di un LSB, quindi non influisce assolutamente sul valore campionato. Se fosse stato altrimenti non avrebbe avuto senso un tempo minimo di finestra di 7 cicli, ma sarebbero dovuti essere di più dato che l'impedenza di uscita dell'operazionale non influisce assolutamente sulla costante di tempo.

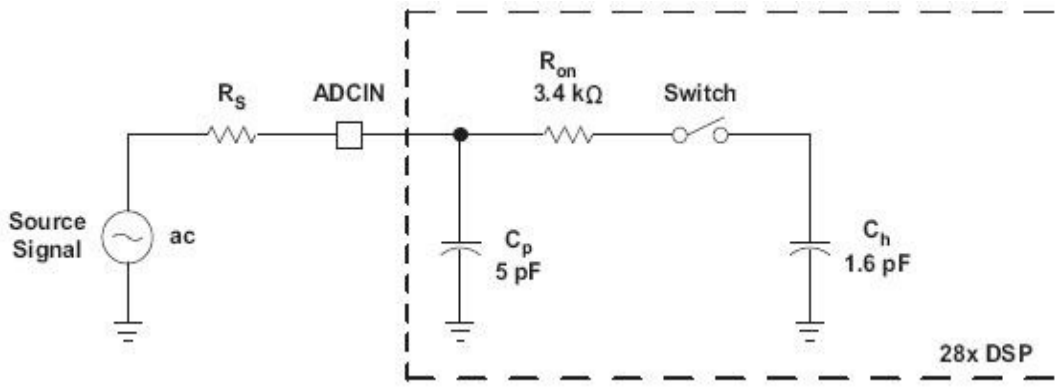



**Figura 4.13** Stadio d'ingresso dell'ADC

Verifichiamo che la capacità di corrente fornita dall'operazionale è in grado di fornire una derivata sufficiente a considerare la carica esponenziale.
Nella caratteristica esponenziale si ha che la derivata massima si ha nell'origine ed è pari a

$$\frac{dV}{dt}=\frac{V}{\tau}=\frac{3.3\,V}{5.28\,nsec}=625\,V/\mu sec \qquad \textbf{(4.10}$$

che risulta maggiore dello SR (360 V/µsec) dell'operazionale!
Vediamo quanto vale lo SR dovuto alla carica delle capacità aprossimando l'operazionale come un generatore di corrente di valore 60 mA,

$$\frac{dV}{dt}=\frac{I}{C_{tot}}=\frac{60\,mA}{6.6\,pF}=9\,kV/\mu sec \qquad \textbf{(4.11}$$

molto maggiore, quindi assolutamente ininfluente! Facendo una rozza aprossimazione per eccesso, si vede che con lo SR dell'operazionale, a raggiungere i 3.3 V ci si mette

$$\frac{3.3\,V}{SR} = \frac{3.3\,V}{360\,V/\mu sec} = 9\,nsec \qquad \textbf{(4.12}$$

e poi considerando che la carica ricominci da 0 in maniera esponenziale si ha un errore

$$\varepsilon = 3.3\,V \cdot e^{\left(\frac{-87.5-9}{5.28}\right)} = 1\,\mu V \qquad \textbf{(4.13}$$

ancora molto minore di un LSB. Essendo l'aprossimazione fatta per eccesso si ha che le cose nella realtà possono solo andare meglio.

### 4.3.3 Impostazione ePWM

Passiamo ora al modulo ePWM, il quale dovrà regolare il periodo di campionamento, l'accensione del LASER e la chiusura dello switch in parallelo alla capacità di integrazione. In seguito mostriamo le forme d'onda per dare maggiore chiarezza

(grafici)

Come si vede le onde PWM del LASER e dello switch risultano complementari, mentre il segnale di trigger dovrà essere dato poco prima (tericamente 87.5 nsec prima, cioè la durata della finestra di acquisizione) dello spegnimento del LASER.
Vediamo come configurare i vari registri per ottenere l'obbiettivo prefissato.

```
SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;     // ePWM1
SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;     // ePWM2
SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;     // ePWM3
SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;     // ePWM4
SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;     // ePWM5
SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;     // ePWM6
SysCtrlRegs.PCLKCR1.bit.EPWM7ENCLK = 1;     // ePWM7
SysCtrlRegs.PCLKCR1.bit.EPWM8ENCLK = 1;     // ePWM8
```

Inanzitutto si abilitano tutti i clock dei moduli PWM, operazione fondamentali per farli lavorare.

```
EPwm1Regs.ETSEL.bit.SOCAEN = 1;
EPwm1Regs.ETSEL.bit.SOCASEL      = 4;
EPwm1Regs.ETPS.bit.SOCAPRD       = 1;
EPwm1Regs.CMPA.half.CMPA   = periodopwm - 22;
EPwm1Regs.TBCTL.bit.CTRMODE      = 0;
EPwm1Regs.TBPRD = periodopwm;
```

In questo set di istruzioni abilitiamo l'impulso di SOC negli ADC del gruppo A (SOCAEN) e facciamo in modo che avvenga quando contatore risulti pari a CMPA (SOCASEL), impostiamo il contatore di eventi a uno cosi che l'ipulso avvenga ogni volta che la condizione si avvera (SOCAPRD), scriviamo nel registro CMPA il valore del periodo del PWM1 meno il valore 22 che corrisponde al numero di cicli che comprenderà il tempo di scarica della capacità in retroazione e la durata della finestra di campionamento. Infine impostiamo il PWM in modalità up-count (CTRMODE) e come numero di cicli di periodo del PWM lasciamo una variabile in modo tale da modificarla con facilità nel caso sia necessario (TBPRD).

```
EPwm1Regs.DBCTL.bit.OUT_MODE     = 1;
EPwm2Regs.DBCTL.bit.OUT_MODE     = 1;
EPwm4Regs.DBCTL.bit.OUT_MODE     = 1;
```

Settando i valori di questi registri salto il ritardo della *dead-band*. La *dead-band* è un submodulo presente all'interno di tutti i moduli PWM inutile per il nostro progetto, quindi lo ignoriamo sia in questa tesi che con le forme d'onda PWM bypassandolo.

La presenza degli altri 2 moduli PWM verrà spiegata in seguito.

```
EPwm1Regs.TBCTL.bit.PHSEN  =  1;
EPwm2Regs.TBCTL.bit.PHSEN  =  1;
EPwm3Regs.TBCTL.bit.PHSEN  =  1;
EPwm4Regs.TBCTL.bit.PHSEN  =  1;
EPwm5Regs.TBCTL.bit.PHSEN  =  1;
```

In questa maniera abilitiamo la fase in tutti i blocchi PWM, quest'operazione è necessaria per il sincronismo dei vari moduli. Quando un modulo PWM riceve un impulso di sincronismo nel contatore viene automaticamente caricato il valore scritto nel registro di fase.

```
EPwm5Regs.TBCTL.bit.SWFSYNC      =  1;
EPwm4Regs.TBCTL.bit.SWFSYNC      =  1;
EPwm3Regs.TBCTL.bit.SWFSYNC      =  1;
EPwm2Regs.TBCTL.bit.SWFSYNC      =  1;
```

Impostando a 1 questi registri invece imponiamo un iniziale segnale di sincronismo via software.

```
EPwm4Regs.TBCTL.bit.SYNCOSEL     =  0;
EPwm3Regs.TBCTL.bit.SYNCOSEL     =  0;
EPwm2Regs.TBCTL.bit.SYNCOSEL     =  0;
EPwm1Regs.TBCTL.bit.SYNCOSEL     =  1;
```

Il registro SYNCOSEL determina quale segnale di sincronismo manderanno in uscita i vari moduli PWM. Settando il SYNCOSEL del modulo EPWM1 mando in uscita da questo stesso modulo il segnale di sincronismo quando il contatore è uguale a zero, insomma ogni volta che si azzerra il conteggio. Negli altri moduli c'è scritto 0 nel registro SYNCOSEL per generare un impulso di sincronismo quando ne arriva un ingresso.
Ciò si è fatto poiché il segnale di sincronismo in uscita dal modulo EPWM1 è connesso al segnale di sincronismo in ingresso al modulo EPWM2, mentre il segnale di sincronismo in uscita dal modulo EPWM2 è connesso al segnale di sincronismo in ingresso al modulo

EPWM3 e cosi via. La **Figura 4.14** può rendere più chiaro il concetto.

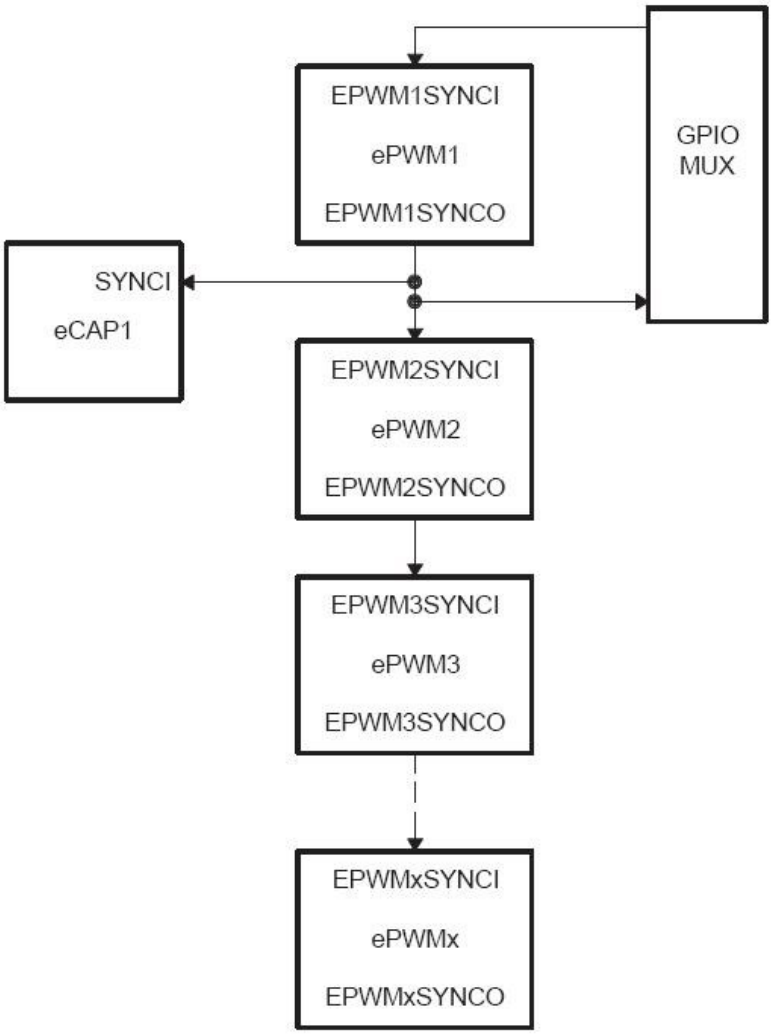



**Figura 4.14** Connessione del segnale di sincronismo dei submoduli PWM

Per rendere il tutto più efficiente è necessario sincronizzare i clock di tutti i moduli PWM con la seguente istruzione.

```
SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
```

Essa fa si che i clock dei moduli PWM abilitati partino col primo fronte di salita allineato.

Dopo aver sincronizzato tutti i moduli PWM, operazione di cui ne sarà più chiara l'utilità in seguito, possiamo iniziare a configurare le forme d'onda PWM che usciranno dal *Piccolo* per pilotare il LASER e lo switch.

```
EPwm1Regs.CMPB = periodopwm-15;

EPwm1Regs.AQCTLA.bit.ZRO = 1;
EPwm1Regs.AQCTLA.bit.CBU = 2;

EPwm1Regs.AQCTLB.bit.ZRO = 2;
EPwm1Regs.AQCTLB.bit.CBU = 1;
```

Impostiamo inanzitutto il valore di CMPB a peridopwm – 15, numero di cicli in cui vogliamo che il LASER sia acceso. Facciamo in modo che l'uscita A si alzi in zero e torni bassa quando il contatore raggiunge il valore di CMPB. Viceversa vogliamo che l'uscita B del modulo EPWM1 si abbassi in zero e si setti quando il contatore diventa pari a CMPB.
Non ci resta ora che impostare i pin d'uscita.

```
EALLOW;

GpioCtrlRegs.GPAMUX1.bit.GPIO0=1;
GpioCtrlRegs.GPADIR.bit.GPIO0=1;

GpioCtrlRegs.GPAMUX1.bit.GPIO1=1;
GpioCtrlRegs.GPADIR.bit.GPIO1=1;

EDIS;
```

Impostando GPAMUX1 pari ad uno facciamo in modo che il MUX colleghi i pin ai moduli PWM, mentre settando GPADIR facciamo si che vengano impostati come output.
Con questo abbiamo concluso il settaggio del modulo ePWM.

### 4.3.4 Impostazione SPI

Vediamo ora come verrà configurata la periferica di comunicazione seriale. Per tale operazione si decide di fare due piccole funzioni per rendere più chiaro il programma e un po' meno disordinato il programma.

```
void spi_init()
{
    SpiaRegs.SPICCR.all =0x000F;
    SpiaRegs.SPICTL.all =0x0006;
    SpiaRegs.SPIBRR =0x0000;
    SpiaRegs.SPICCR.all =0x009F;
    SpiaRegs.SPIPRI.bit.FREE = 1;
    SpiaRegs.SPIPRI.bit.TRIWIRE = 1;
}
```

Scrivendo nel registro SPICCR il valore esadecimale 000F impongono varie cose. Inanzitutto si mette la periferica SPI nello stato di reset, operazione obligatoria per eseguire certi settaggi. Si decide la polarità del clock in uscita, ossia si decide di inviare un dato sul fronte di salita, mentre di riceve sul fronte di discesa e ne si impone uno sfasamento nullo. Si disabilità il *loop back mode*, cioè una modalità di testa in cui si cortocircuitano internamente il SIMO e il SOMI (modalità che evidentemente vale solo nel *master mode*) per una funzione di controllo. In fine si imposta la lunghezza della parola che verrà shiftata in uscita, che viene impostata massima, cioè 16 bit.
Scrivendo invece nel registro SPICTL il valore esadecima 0006 si imposta la fase pari a 0, si imposta il *master mode* e si abilità la trasmissione, mentre l'interrupt viene disbailitato, ci calcoleremo infatti a priori il tempo necessario per eseguire una trasmissione dati.
Impostando invece SPIBBR a 0 impostiamo il valore del clock esterno che come già detto sarà il clock interno diviso per il valore scritto in questo registro che però assimila i valori minori o uguali a 3 come se fossero 4. Quindi scrivere 0, scrivere 1, scrivere 2, scrivere 3 o scrivere 4 è esattamente la stessa cosa, il clock esterno sarà LSPCLK/4, dove LSPCLK è il clock della periferica SPI.
Scrivendo ora 009F sul registro SPICCR lasciamo le stesse impostazioni di prima, meno

che si disabilità il reser e si abilita il *loop back mode*.
Infine la funzione termina con l'ipostazione di *free run*, quindi l'SPI continua a lavorare indipendentemente da tentati di sospensione esterni, e con l'impostazione di *3-wire*, quindi il pin SOMI è come se non esistesse più, il Piccolo può solo inviare dati e non riceverne. Infatti l'SPI verrà collegato ad un DAC il quale non da dati digitali in uscita, ma semplicemente converte in un segnale analogico quelli ricevuti in ingresso.

```
void spi_fifo_init()
{
    SpiaRegs.SPIFFTX.all=0xA040;
    SpiaRegs.SPIFFRX.all=0x005f;
    SpiaRegs.SPIFFCT.all=0x0;
}
```

Questi tre registri regolano parametri strettamente legati alla trasmissione, alla ricezione e al loro controllo. SPIFFTX come dice il nome, regola la trasmissione in modalità *First in First out*. Scrivendo quel particolare valore esadecimale si disabilità la modalità reset per permettere il funzionamento della periferica di trasmissione, si impone la stato vuoto del FIFO, si resetta l'interrupt e si azzera il bit di flag, e per finire si impone il valore 0 come numero di bit dentro il *shift register* che scatenerebbe l'intterupt.
Nel registro SPIFFRX che gestisce la ricezione nella modalità FIFO si azzerra il bit di overflow flag che si alza in automatico quando il registro di ricezione si riempe, si disabilità il reset, si impone lo stato vuoto, si resetta l'interrupt e se ne azzera il bit di flag, infine si impone a 16 il valore che determina il numero di bit nel *shift register* per cui si scatenerebbe l'intterupt.
SPIFFCT determina solo il ritardo nella trasmissione tra i dati presenti nel *shift register* e quelli presenti nel livello superiore di buffer, che poi finiranno comunque nel *shift register* per essere trasmessi.

### 4.3.5 Impostazioni PIE

Un interrupt è un segnale asincrono che indica un bisogno di attenzione da parte di una periferica finalizzata ad una particolare richiesta di servizio oppure un evento sincrono che consente l'interruzione di un processo qualora si verifichino determinate condizioni.
L'interrupt se possibile è meglio non usarlo dato che blocca ogni processo del *Piccolo* finchè non vengono eseguite le istruzioni scritte nel registro ad esso dedicato.
Quindi si decide di azzerrare ogni *interrupt enable* sia a livello CPU che a quello del PIE per poi attivarli man mano che risulterà necessario.

```
void InitPieCtrl(void)
{
    // Disable Interrupts at the CPU level:
    DINT;

    // Disable the PIE
    PieCtrlRegs.PIECTRL.bit.ENPIE = 0;

      // Clear all PIEIER registers:
      PieCtrlRegs.PIEIER1.all = 0;
      PieCtrlRegs.PIEIER2.all = 0;
      PieCtrlRegs.PIEIER3.all = 0;
      PieCtrlRegs.PIEIER4.all = 0;
      PieCtrlRegs.PIEIER5.all = 0;
      PieCtrlRegs.PIEIER6.all = 0;
      PieCtrlRegs.PIEIER7.all = 0;
      PieCtrlRegs.PIEIER8.all = 0;
      PieCtrlRegs.PIEIER9.all = 0;
      PieCtrlRegs.PIEIER10.all = 0;
      PieCtrlRegs.PIEIER11.all = 0;
      PieCtrlRegs.PIEIER12.all = 0;

      // Clear all PIEIFR registers:
      PieCtrlRegs.PIEIFR1.all = 0;
      PieCtrlRegs.PIEIFR2.all = 0;
      PieCtrlRegs.PIEIFR3.all = 0;
      PieCtrlRegs.PIEIFR4.all = 0;
      PieCtrlRegs.PIEIFR5.all = 0;
      PieCtrlRegs.PIEIFR6.all = 0;
      PieCtrlRegs.PIEIFR7.all = 0;
      PieCtrlRegs.PIEIFR8.all = 0;
      PieCtrlRegs.PIEIFR9.all = 0;
      PieCtrlRegs.PIEIFR10.all = 0;
      PieCtrlRegs.PIEIFR11.all = 0;
```

```
        PieCtrlRegs.PIEIFR12.all = 0;
}
```

## 4.4 Piccolo F28069 controlSTICK

Il *Piccolo* viene fornito su una board con una circuiteria di supporto per permetterne un facile *debug* e un caricamento dei programmi via USB che inizialmente verranno caricati nella memoria flash, poi quando se ne è certi del funzionamento e lo si vuole caricare definitivamente, viene salvato sulla memoria ROM e verrà caricato ogni volta che il Piccolo verrà collegato all'alimentazione. All'acquisto, nel *Piccolo* c'è caricato un semplice programma di accensione dei LED che nel caso non funzionasse implicherebbe un malfunzionamento del sistema.

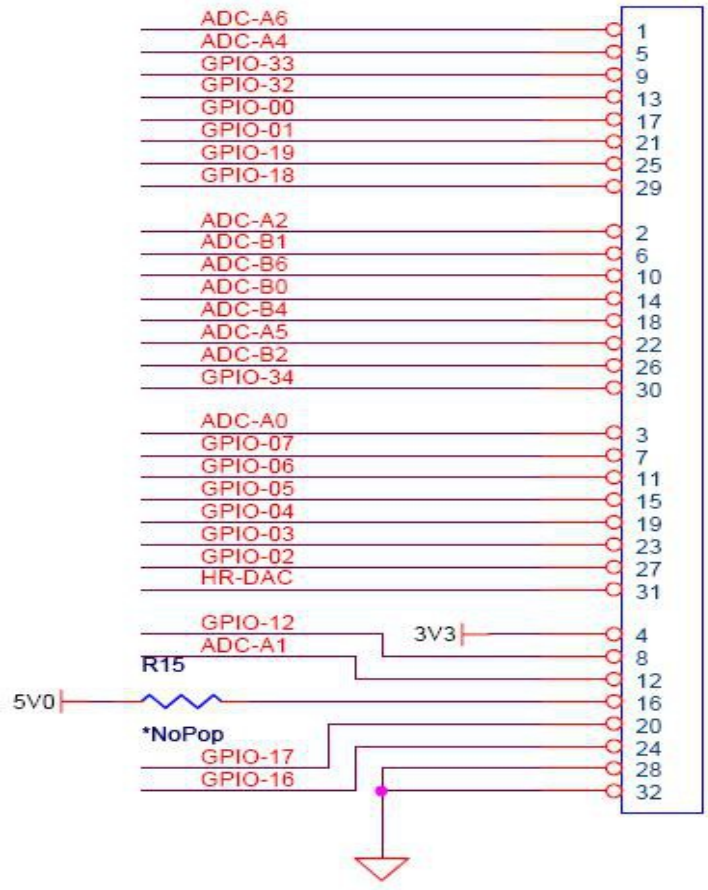



**Figura 4.15** Piedinatura del controlStick

# Capitolo 5

# Realizzazione dello strumento e misure

## 5.1 Introduzione

Realizzata l'ottica, l'elettronica analogica e la configurazione del PSD, non ci resta che mettere insieme l'intero sistema per realizzare il misuratore di vibrazioni a triangolazione laser.
In questo capitolo si discuterà di tutti quei problemi che hanno comportato la realizzazione dello strumonte con le relative modifiche e migliorie apportate per incrementarne la sensibilità.
In fine verranno riportate e commentate le prestazioni con grafici.

## 5.2 Prima configurazione

La prima configurazione viene realizzata con la transimpedenza di cui ne riportiamo in seguito lo schema elettrico completo

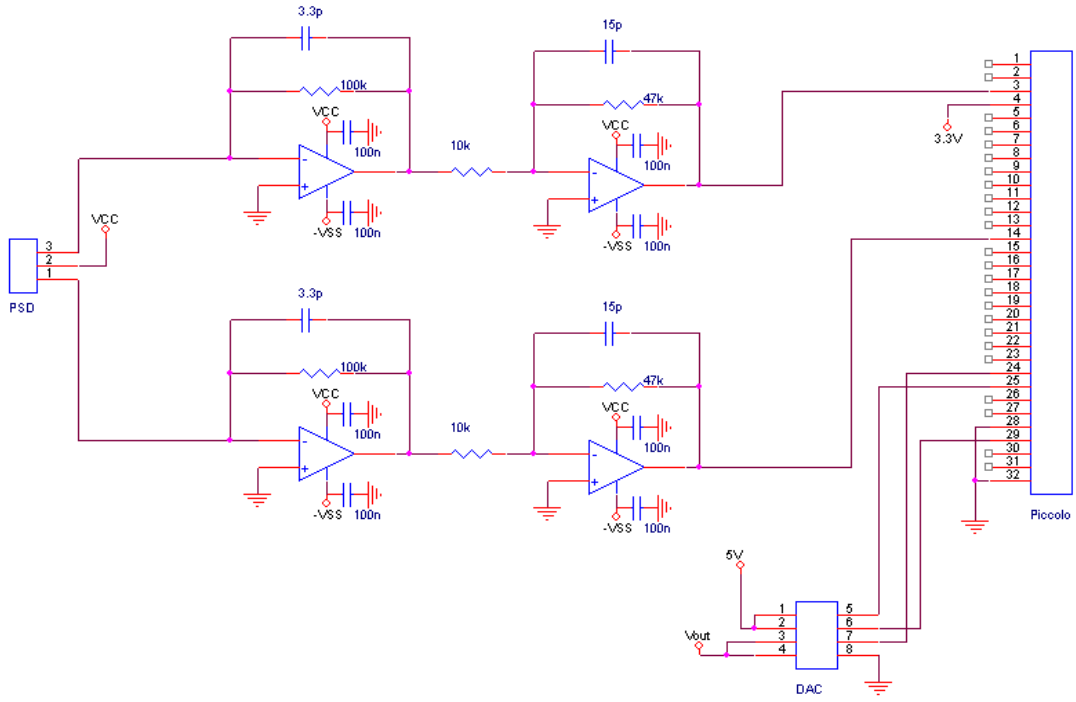



**Figura 5.1** Schema elettrico completo del triangolatore con transimpedenza

In questa configurazione le forme d'onda PWM in uscita dal *Piccolo* sono inutili dato che il LASER e tutto il resto del circuito analogico lavora in regime continuo, del modulo ePWM ci interesserà solo il segnale di trigger per l'ADC.

Le uscire degli operazionali vengono connesse ai pin 3 e 14 del *Piccolo F28069 controlSTICK*, che corrispondo agli ingresso ADC-A0 e ADC-B0, ossia gli ingressi della coppia di ADC che abbiamo impostato per il campionamento simultaneo.

Alla fine di ogni conversione si ha che il valore campionato da ADC-A0 viene scritto nel registro AdcResult.ADCRESULT0, mentre quello campionato da ADC-B0 viene scritto nel registro AdcResult.ADCRESULT1. Questi registri vengono associati ai SOC, infatti

sarebbe più corretto dire che AdcResult.ADCRESULT0 è associato a SOC0 mentre AdcResult.ADCRESULT1 è associato a SOC1. (INTERRUPT)
I valori di questi registri sono degli int (a 16 bit quindi) e noi li convertiamo il float32 (a 32 bit) dato che poi dovranno essere processati e filtrati, operazioni che necessariamente vanno fatte con variabili float per poterne garantire la precisione e la stabilità.

```
ADCRESULTA0 = AdcResult.ADCRESULT0;
ADCRESULTB0 = AdcResult.ADCRESULT1;
```

La conversione viene fatta semplicemente uguagliano il valore dei registri AdcResult.ADCRESULT0 e AdcResult.ADCRESULT1 a delle variabili float32 ADCRESULTA0 e ADCRESULTB0.
Fatto ciò possiamo procedere con la prima operazione per ottenere ul valore proporzionale alla distanza

```
diffA0B0=(ADCRESULTA0-ADCRESULTB0)/(ADCRESULTA0+ADCRESULTB0);
```

che equivale infatti alle formula

$$y=\frac{x_1-x_2}{x_1+x_2} \qquad \textbf{(5.1}$$

Disponiamo quindi adesso di un segnale proporzionale allo spostamento, normalizzato e che può assumere sia valori positivi e che quelli negativi.
Dato che poi il valore andrà mandato alla periferica SPI, per essere riconvertito in un segnale analogico tramite il DAC, che tratta solo dati unsigned int è necessario che questo segnale diventi positivo.
Per farlo diventare positivo ci sommiamo il valore 32767 che lo sposta esattamente a metà della dinamica delle variabili unsigned int.

```
diffA0B0=diffA0B0*i2;
diffA0B0=diffA0B0 + 32767;
```

Come si può vedere non ci si è limitati a a sommare il valore 32767, ma si è effettuata anche la moltiplicazione per un coefficiente i2 lasciato indefinito il cui scopo sarà amplificare e massimizzare la dinamica del segnale.

### 5.2.1 Filtro di Butterworth

Abbiamo adesso un valore proporzionale allo spostamento, centrato a metà della dinamica positiva e amplificato, l'ultima cosa di cui ha bisogno è un filtraggio per eliminare il rumore fuori banda. Realizziamo quindi un filtro di Butterworth.

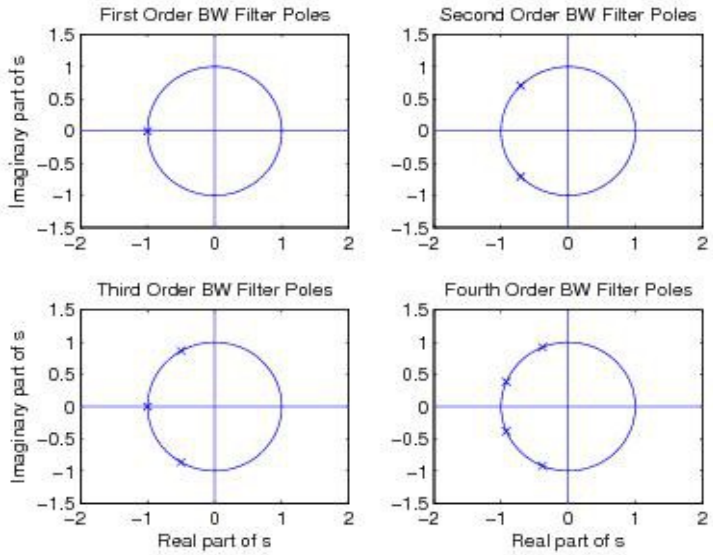



**Figura 5.2** Posizionamento dei poli sull'asse immaginario dei primi 4 polinomi di Butterworth

Il filtro di Butterrorth è un filtro che posiziona i poli nel semipiano complesso sinistro sulla circonferenza di raggio $\omega_p$, dove il termine $\omega_p$ rappresenta la pulsazione di taglio desiderata.

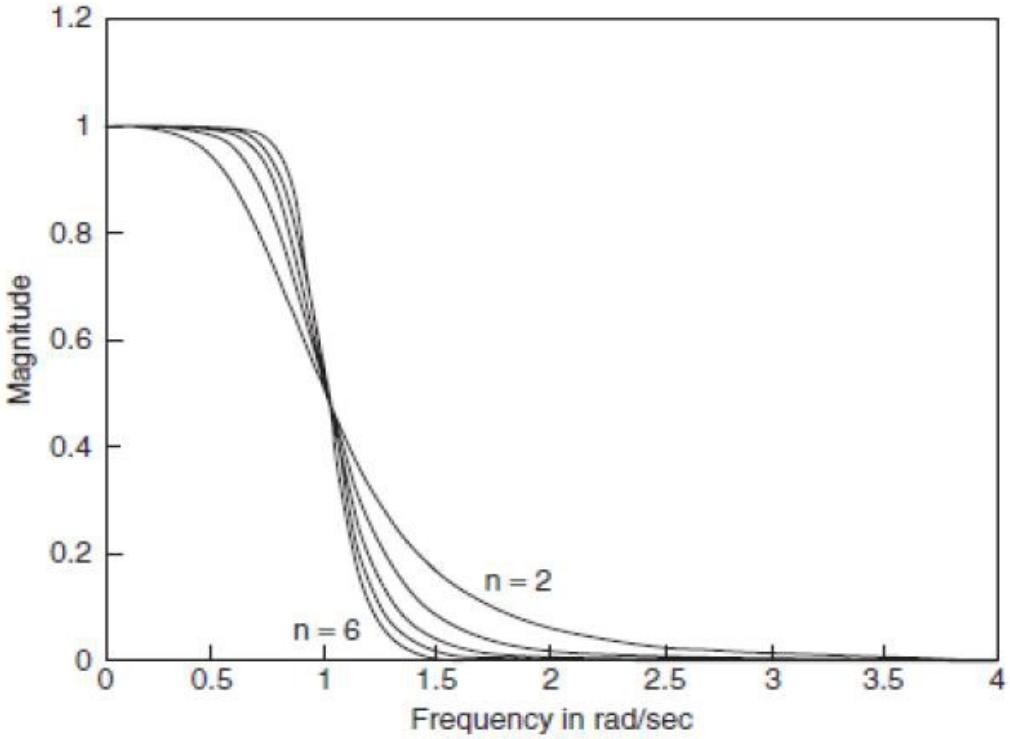



**Figura 5.3** Risposta in frequenza di filtri di Butterworth con ordini diversi

Il filtro di Butterrorth è il filtro che presenta la maggior piattezza del modulo in banda passante.

Sperimentalmente si verificherà che il filtro di ordine maggiore il cui tempo di computazione non genera uno sforamento del periodo di campionamento è il terzo, sufficiente al nostro compito.

Per realizzare la funzione di trasferimento del filtro di Butterworth del terzo ordine è necessario il polinomio di Butterworth dello stesso ordine. In seguito sono riportati i polinomi di Butterworth di vari ordini.

| $n$ | Butterworth Polynomial $D(p)$ in Polynomial and Factored Form |
|---|---|
| 1 | $p+1$ |
| 2 | $p^2+\sqrt{2}p+1$ |
| 3 | $p^3+2p^2+2p+1=(p+1)(p^2+p+1)$ |
| 4 | $p^4+2.61326p^3+3.41421p^2+2.61326p+1$ $=(p^2+0.76537p+1)(p^2+1.84776p+1)$ |
| 5 | $p^5+3.23607p^4+5.23607p^3+5.23607p^2+3.23607p+1$ $=(p+1)(p^2+0.618034p+1)(p^2+1.931804p+1)$ |
| 6 | $p^6+3.8637p^5+7.4641p^4+9.1416p^3+7.4641p^2+3.8637p+1$ $=(p^2+0.5176p+1)(p^2+1.4142p+1)(p^2+1.9318p+1)$ |

**Figura 5.4** Polinomi di Butterworth

Al polinomio di Butterworth del terzo ordine bisognerà sostituire all'incognita il termine $s/\omega_p$ per ottenere un passabasso, dove $s$ è la variabile complessa della trasforma di Laplace. La funzione di trasferimento sarà quindi data da 1 fratto il polinomio di Butterworth col termine incognito sostituito da $s/\omega_p$.

$$H(s)=\frac{1}{\left(\frac{s}{\omega_p}\right)^3+2\left(\frac{s}{\omega_p}\right)^2+2\left(\frac{s}{\omega_p}\right)+1} \quad \textbf{(5.2}$$

Questa funzione di trasferimento però ancora non è realizzabile nel *Piccolo* dato che è a tempo continuo. Dobbiamo passare al dominio tempo discreto, ossia dal dominio analogico di $s$ a quello digitale di $z$. Per fare ciò abbiamo a disposizione due possibili metodi: l'invarianza della risposta all'impulso e la trasformazione bilineare.
L'invarianza della risposta all'impulso consiste nel campionare la risposta all'impulso del filtro analogico e poi effettuarne la trasformata z.

$$h(n)=h_a(nT_s) \quad \textbf{(5.3}$$

$$H(z)=Z[h_a(nT_s)]=\sum_0^\infty h_a(nT_s)\cdot z^{-n} \qquad \textbf{(5.4}$$

Tecnica che ci risulta scomoda da applicare e che inoltre soffre del problema dell'aliasing.
La trasformazione bilineare consiste nell'esguire la seguente sostituzione nella funzione di trasefrimento

$$s=\frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \qquad \textbf{(5.5}$$

Questa tecnica non presenta l'inconveniente dell'aliasing dato che riesce a mappare tutta la frequenza analogica nella circonferenza unitaria del dominio *z*. Ovviamente ciò non può avvenire linearmente, questo risulta il suo inconveniente!
La funzione di mappatura è la seguente

$$\Omega=2\,atan\left(\frac{\omega}{2\mathrm{f}_s}\right) \qquad \textbf{(5.6}$$

dove $\Omega$ rappresenta la pulsazione digitale che può andare solo da $\pi$ a $-\pi$.

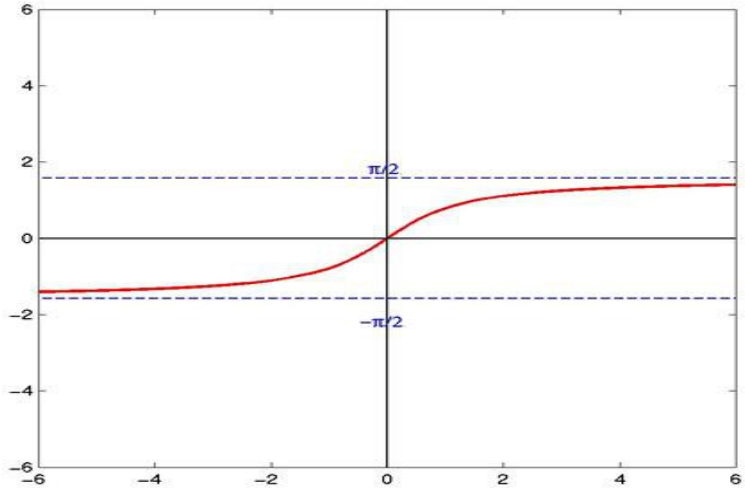



**Figura 5.5** Arcotangente

Come si può vedere la funzione arcotangente è fortemente non lineare, tuttavia in un intorno dell'origine la si può considerare lineare, infatti per $x$ tendente a 0 si ha che l'arcotangente di $x$ è assintotica a $x$.

$$atan(x) \sim x \; per \; x \to 0 \qquad \textbf{(5.7}$$

Quindi se $\omega << 2f_s$ possiamo considerare la mappatura lineare, è importante perciò che la pulsazione analogica del nostro polo, ossia la banda del nostro segnale, sia molto minore di due volte la frequenza di campionamento. O viceversa la frequenza di campionamento dovrà essere molto maggiore di metà volte la massima pulsazione del segnale.
Nel nostro caso abbiamo $f_p$ = 80 kHz che corrisponde a una pulsazione di 502 krad/sec e una $f_s$ = 500kHz.
Verifichiamo l'errore massimo di mappatura.

$$atan\left(\frac{\omega}{2f_s}\right) = 0.4636 \qquad \textbf{(5.8}$$

$$\frac{\omega}{2f_s} = 0.5 \qquad \textbf{(5.9}$$

Da cui ricaviamo che lo scostamento dalla linearità pari a

$$(0.5 - 0.4636)/0.5 = 0.07 \qquad \textbf{(5.10}$$

Uno scostamento del 7% è accetabile. Applichiamo quindi il metodo della trasformazione bilineare per ottenere il filtro di Butterworth digitale.
Per ottenere i coefficienti del filtro digitale usiamo una comoda funzione di Matlab: bilinear. Questa funzione riceve in ingresso la frequenza di campionamento, i coefficienti

del numeratore e del denominatore della funzione di trasferimento di cui si vuole l'equivalente digitale con la trasformazione bilineare e da in uscita i coefficienti della funzione di trasferimento nel dominio discreto. Riportiamo in seguito le istruzioni usate per ottenere i coefficienti.

```
>> format long
>> fs=500

fs =

   500

>> polo=80*2*pi

polo =

    5.026548245743669e+002

>> [numd,dend]=bilinear(1,[(1/polo),1],fs);
>> polo2=polo*polo

polo2 =

    2.526618726678876e+005

>> [numd2,dend2]=bilinear(1,[(1/polo2),(1/polo),1],fs);
```

La prima istruzione, format long, serve per ottenere fino a 14 cifre dopo la virgola al momento della visualizzazione della variabile. Ciò perchè il troncamento dei coefficienti causa uno spostamento dei poli nel piano complesso *z*, rischiando di farli uscire dal cerchio di raggio unitario che delimita la zona di stabilità. Questo scostamento è proporzionale all'errore di troncamento, quindi più cifre si hanno a disposizione dopo la virgola e meno rilevante sarà lo spostamento dei poli.
Per velocizzare i calcoli si è deciso di scomporre il filtro del terzo ordine in una cascata di due filtri, uno del primo ordine e uno del secondo ordine, infatti il polinomio di Butterworth si può riscrivere come

$$(p+1)(p^2+p+1) \quad \textbf{(5.11}$$

e determina le due funzioni di trasferimento

$$H_1(s)=\frac{1}{(\frac{s}{\omega_p})+1} \quad \textbf{(5.12}$$

$$H_2(s)=\frac{1}{(\frac{s}{\omega_p})^2+(\frac{s}{\omega_p})+1} \quad \textbf{(5.13}$$

Questo spiega l'uso di due volte la funzione bilinear.
Ricaviamo quindi le funzioni di trasferimento nel dominio digitale

$$H_{d1}(z)=\frac{0.334511170731938+0.334511170731938z^{-1}}{1-0.330977658536123z^{-1}} \quad \textbf{(5.14}$$

$$H_{d2}(z)=\frac{0.143940904262370+0.287881808524741z^{-1}+0.143940904262370z^{-2}}{1-0.851513722288680z^{-1}+0.427277339338162z^{-2}} \quad \textbf{(5.15}$$

Ricordandosi poi che

$$H_d(z)=\frac{Y(z)}{X(z)} \quad \textbf{(5.16}$$

si ricavano le due funzioni

$$y_1[n]=0.334511170731938\mathrm{x}[n]+0.334511170731938\mathrm{x}[n-1]+0.330977658536123\mathrm{y}[n-1]$$

**(5.17**

$$y_2[n]=0.143940904262370\mathrm{x}[n]+0.287881808524741\mathrm{x}[n-1]+0.143940904262370\mathrm{x}[n-2]+...$$
$$...+0.851513722288680\mathrm{y}[n-1]-0.427277339338162\mathrm{y}[n-2]$$

**(5.18**

Possiamo quindi realizzare finalmente il filtro digitale di Butterworth del terzo ordine.

```
// coefficienti filtro del primo ordine

float32 a1=0.334511170731938;
float32 a2=0.334511170731938;
float32 a3=0.330977658536123;

// coefficienti filtro del secondo ordine

float32 b1=0.143940904262370;
float32 b2=0.287881808524741;
float32 b3=0.143940904262370;
float32 b4=0.851513722288680;
float32 b5=-0.427277339338162;
```

Abbiamo qua sopra dichiarato i coefficienti necessari al filtro.

```
y1=(a1*diffA0B0) + (a2*x2) + (a3*y2);
x2=diffA0B0;
y2=y1;


w1=b1*y1 + b2*t2 + b3*t3 + b4*w2 + b5*w3;
w3=w2;
w2=w1;
t3=t2;
t2=y1;
```

Ed ecco realizzato il filtro di Butterworth del terzo ordine come cascata di due filtri e che

in ingresso riceve diffA0B0 e in uscita restituisce w1.

Disponiamo ora del nostro segnale proporzionale allo spostamento, a metà della dinamica positiva, amplificato e filtrato. Non resta ora che spedirlo tramite la periferica di comunicazione SPI al DAC per verificare il funzionamento del triangolatore laser e le sue prestazioni.

### 5.2.2 DAC

Diamo prima una rapida occhiata al *DAC8531* che sarà il DAC utilizzato per la conversione digitale-analogica. Esso è un DAC voltage scaling a 16 bit che butta fuori la tensione desiderata tramite una serie di interruttori connessi nei vai punti di un partitore resistivo. Verrà alimentato con una tensione di 5 V, tensione che rappresenterà anche il valore massimo d'uscita dato che è rail to rail e adatta alle sue tensioni di lavoro comprese tra 2.7 V e 5 V.

In base al codice caricato nel DAC si determina quale nodo della stringa resistiva verrà connessa poi all'amplificatore tramite la chiusura di uno degli interruttori.

Ogni resistenza ha ai suoi capi una tensione pari a 1 LSB ed essendo la tensione prelevata da una stringa resistiva si ha che la caratteristica è sicuramente monotona.

In seguito ne viene mostrata l'architettura.

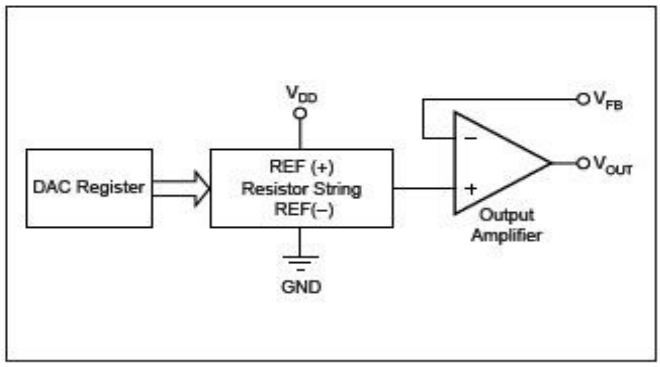



**Figura 5.6** Architettura DAC

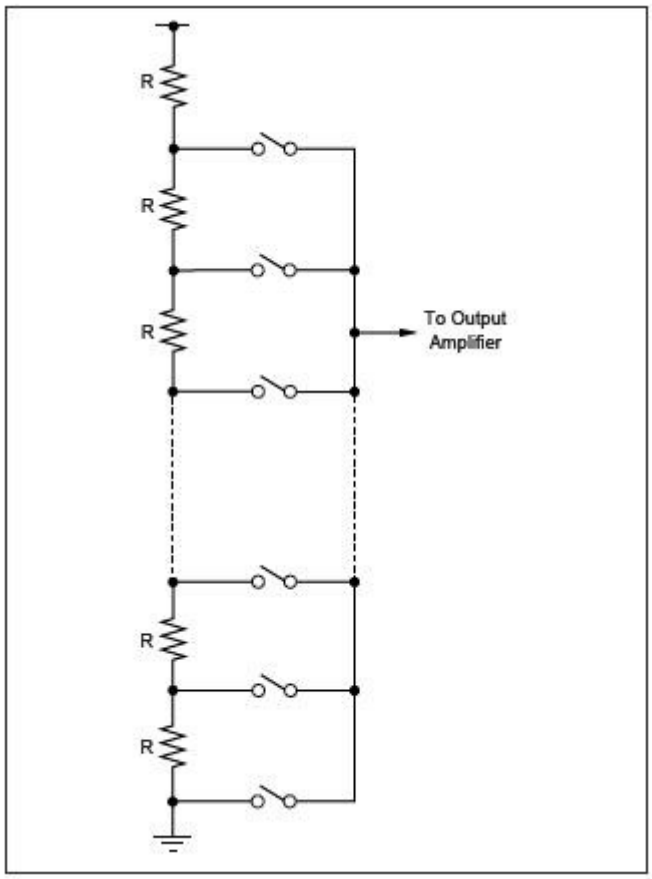



**Figura 5.7** Fascia di resistenza

Il codice di conversione è binario ed è descritto dalla legge

$$V_{out} = V_{ref} \frac{D}{65536} \qquad \textbf{(5.19}$$

dove *D* rappresenta il numero binario mandato nel registro del DAC.

Il *DAC8531* ha un interfaccia seriale composta da 3 linee (SYNC, SCLK, $D_{IN}$). La comunicazione inizia quando la linea di SYNC viene abbassata. I dati in $D_{IN}$ vengono

shiftati all'interno di un shift register a 24 bit sul fronte di discesa del clock in SCLK che può arrivare fino a 30 MHz maggiore dei 20 MHz massimi che il *Piccolo* può trasmettere, quindi possiamo far andare il Piccolo al massimo delle sue possibilità. Per iniziare una nuova trasmissione bisogna alzare il SYNC per un minimo di 33 nsec. Se il SYNC viene alazato prima della fine della comunicazione il shift register viene resettato e la comunicazione viene considerata invalida.
Dei 24 bit del shift register i primi 6 non contano niente, i bit 6 e 7 identificano la modalità di funzionamento (noi manderemo sempre 00 per lavorare in *normal mode*) mentre gli ultimi 16 bit rappresentano il dato.
Mostriamo in seguito il codice per avviare la comunicazione.

```
if(i==2)
       {

    SpiaRegs.SPIDAT = sdata>>8;
    SpiaRegs.SPITXBUF = sdata<<8;
    sdata=0;

    i=0;
      }
      i++;
```

Come si vede, dato che prima dei bit dove risiede l'informazione sulla posizione del bersaglio bisogna mandare 8 bit pari a zero, nel shift register SPIDAT viene scritto il dato shiftato verso destra di 8 bit, dato che la trasmissione parte dal bit più significativo. Il registro SPIDAT però è composto solo da 16 bit, quindi l'informazione risulta incompleta, quindi si scrivono i bit mancanti nel registro SPITXBUF, cioè il buffer di trasmissione FIFO che copia il dato destinato all'uscita nel shift register SPIDAT non appena termina la trasmissione, cosi che la trasmissione dati prosegue senza interruzioni. Il registro di buffer SPITXBUF è composto anche lui solo da 16 bit, quindi effettuando uno shift verso sinistra di 8 bit fa si che vengono scritti solo gli 8 bit meno significativi del dato negli 8 bit pi significativi del registro, cosi che saranno i primi ad essere trasmessi quando verranno

copiati nel shift register SPIDAT, mentre gli 8 restanti resteranno 0, ma comunque non sarebbero stati presi in considerazione dal DAC.
Altro particolare da notare è che la trasmissione avviene una volta ogni 4 cicli, questo per permettere all'ADC di lavorare a frequenza più alta.
Con LSPCLK = 40 MHz otteniamo un SPICLK = 10 Mhz e quindi un tempo necessario alla comunicazione pari a

$$T_{SPI} = \frac{32}{SPICLK} = 3.2\,\mu sec \qquad \textbf{(5.20}$$

troppo grande rispetto al periodo di campionamento pari a 2 μsec.
Il rate di comunicazione risulta quindi fissato da 2 cicli di campionamento $f_s/2$ = 250 ksample/sec.

### 5.2.3 Prestazioni

Ci resta solo da valutare il coefficiente che determina il guadagno i2 e poi il sistema risulterà interamente realizzato. In seguito vengono riportati varie caratteristiche statiche con diversi coefficienti di moltiplicazione i2 scelti con ragionevolezza. Se ne studierà la linearità e in base al compromesso tra sensibilità e dinamica di misura si sceglierà quello a noi più adatto.
La caratteristica di ampiezza si valuterà con un bersaglio con vite micrometrica in grado di traslare con risoluzione di 5 μm e se ne preleverà il segnale dall'uscita del DAC.
Nella Grafico viene riportata la caratteristica statica nel caso venga scelto una coefficiente i2 = 41700. Si nota che la caratteristica è abbastanza lineare e satura ai valori limite di tensione del DAC che in questo caso sono 0 e 3.3 V.

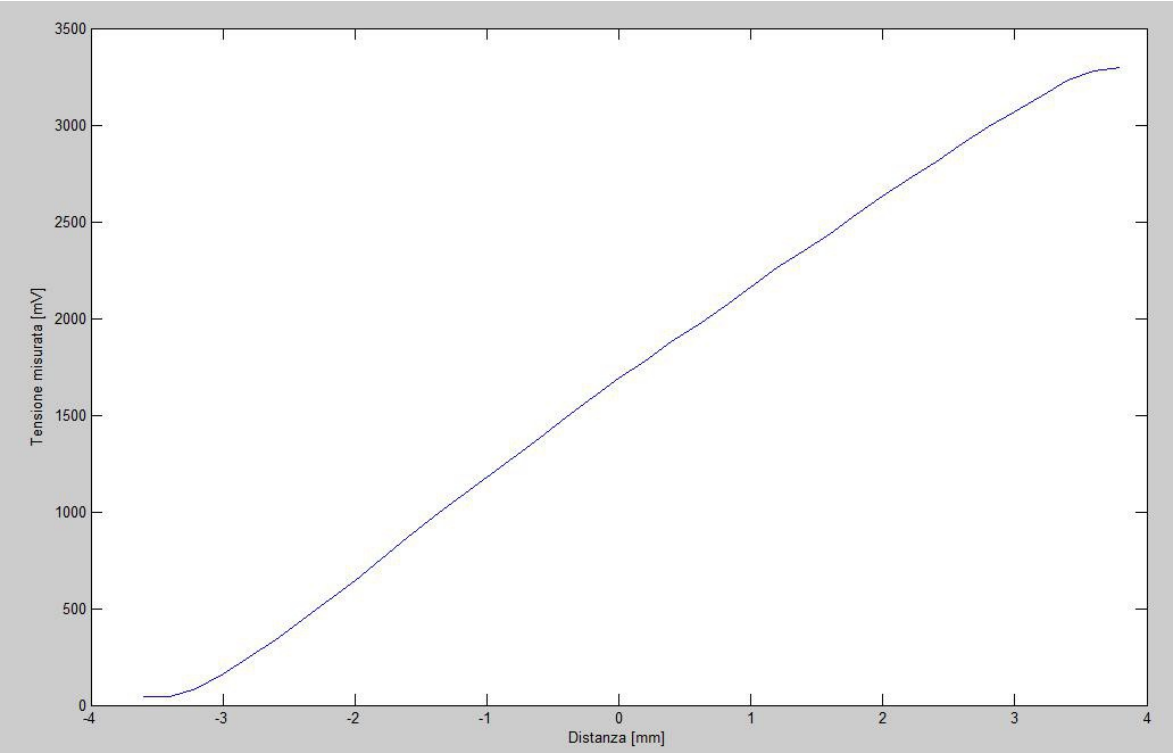



**Figura 5.8** Caratteristica statica triangolatore con transimpedenza con coefficiente i2 = 41700

La caratteristica è stata ricava con uno step pari a 0.2 mm. Si nota una variazione tra uno step e l'altro di 100 mV da cui ricaviamo una sensibilità pari a

$$S = \frac{\Delta V}{\Delta L} = \frac{100\,mV}{0.2\,mm} = 0.5\,V/mm \qquad \textbf{(5.21}$$

Nei grafici vengono riportati i valori di deviazione quadratica media calcolati dall'oscilloscopio in tensione e poi, tramite la divisione per la sensibilità, riportati nell'equivalente rumore in μm.

La deviazione si vede incrementare spostandosi verso i bordi dato che mentre una delle fotocorrenti aumenta, diventando più ampia nei confronti del rumore, l'altra diminuisce diminuendo il rapporto segnale-rumore e determinando un rapporto segnale-rumore complessivamente peggiore. Si vede anche che in un bordo la deviazione aumenta con una pendenza maggiore dato che il segnale è più debole poiché il bersaglio è più distante.

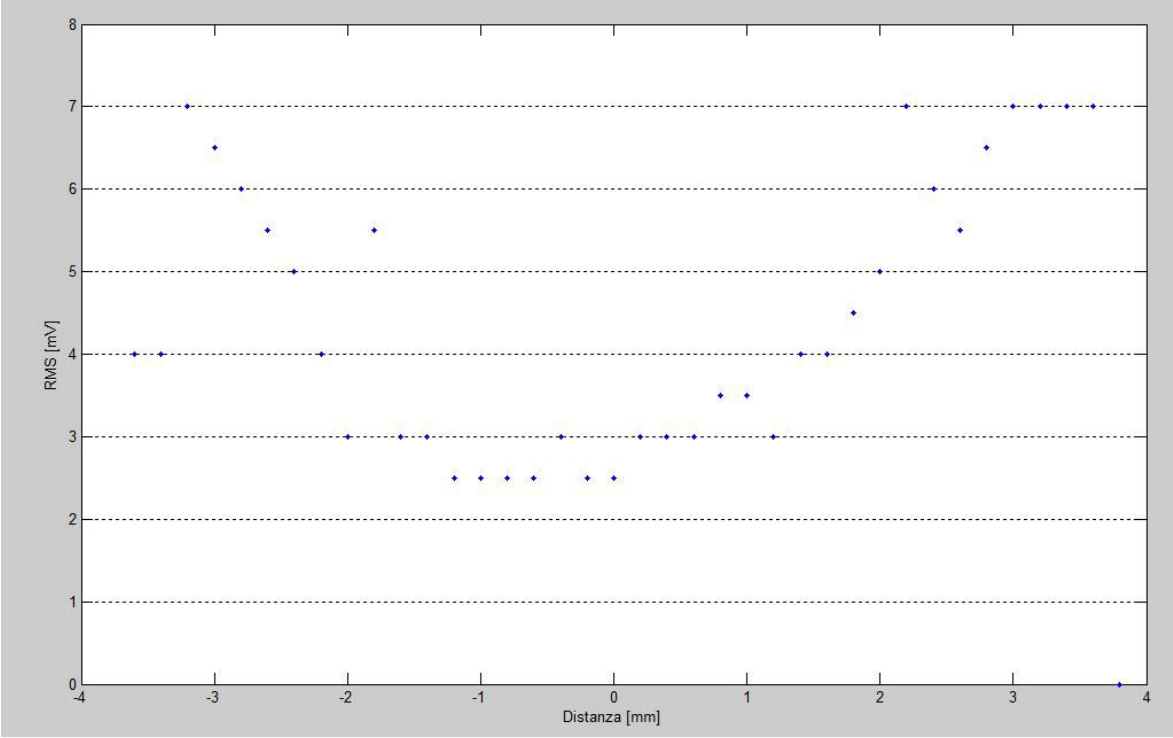



**Figura 5.9** Deviazione standard in mV

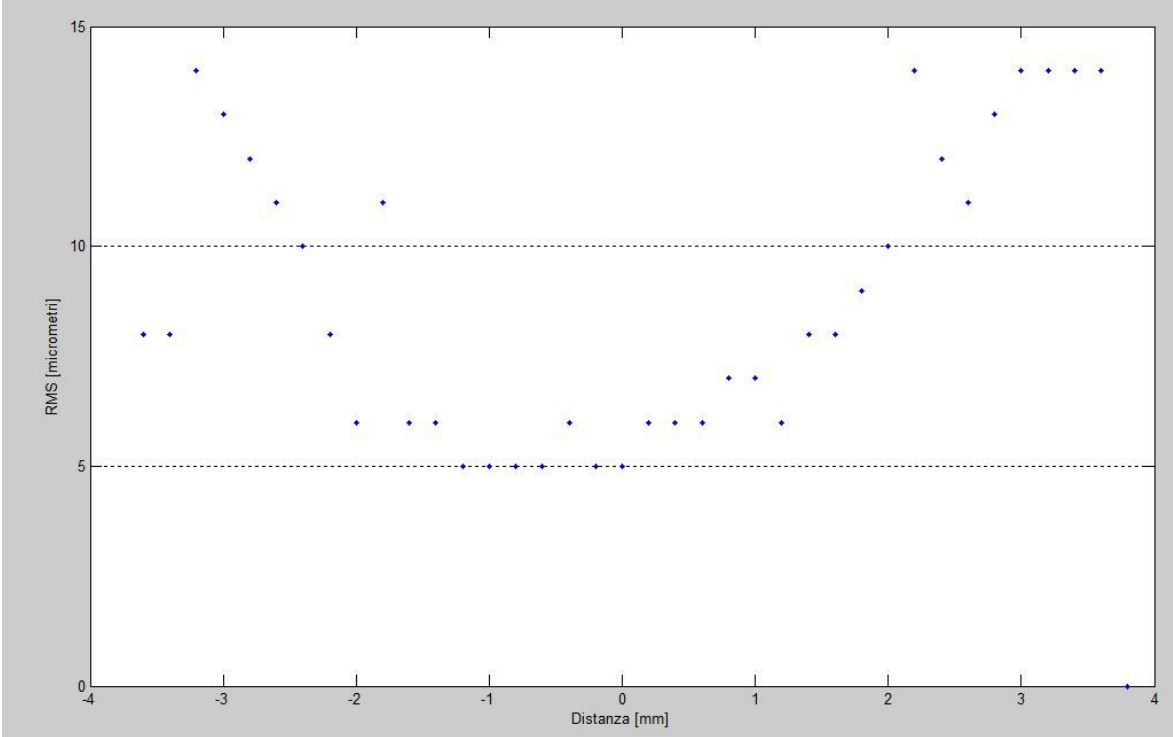



**Figura 5.9** Deviazione standard in µm

Nel Grafico viene riportata la caratteristica statica con un coefficiente i2 quadruplicato rispetto a prima, ossia pari a 1668800. In questo caso il DAC satura prima e per avere un maggior numero di punti si è diminuito il passo di spostamento a 0.1 mm.

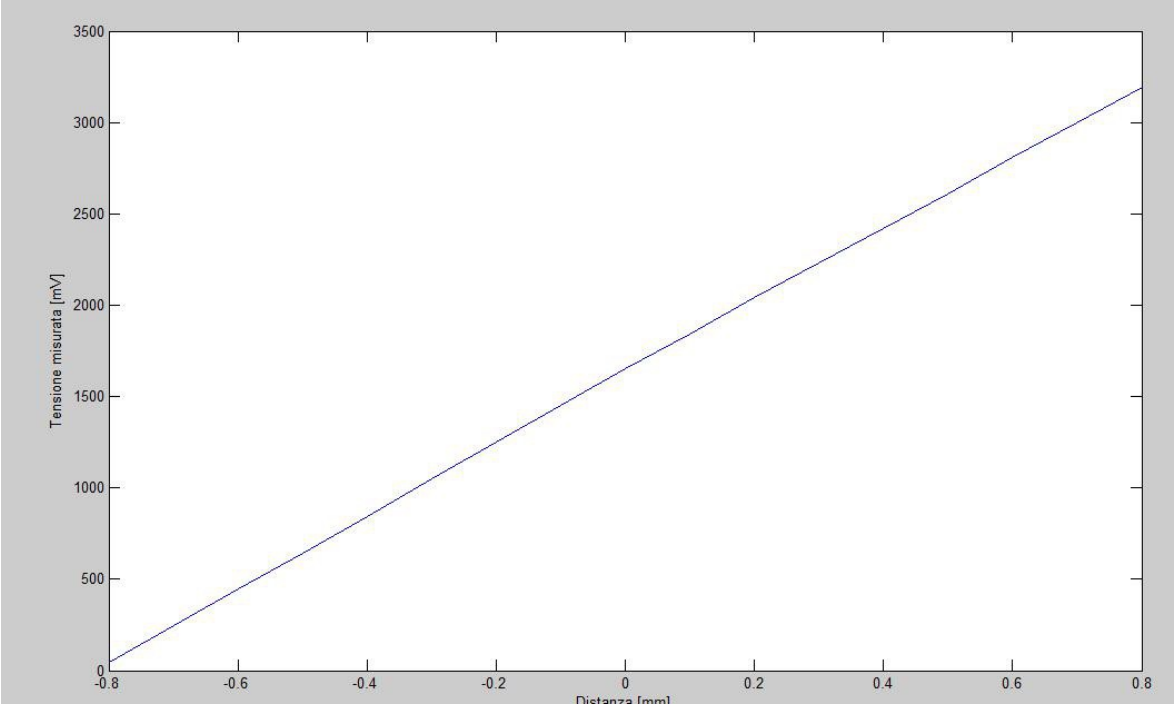



**Figura 5.10** Caratteristica statica triangolatore con transimpedenza con coefficiente i2 = 166800

Tra uno step e l'altro si ha mediamente una variazione di tensione pari a 200mV portando la sensibilità a

$$S = \frac{\Delta V}{\Delta L} = \frac{200\,mV}{0.1\,mm} = 2\,V/mm \qquad \textbf{(5.22}$$

La sensibilità è quadruplicata, portando però da un range di misura da circa 7 mm a poco meno di 2 mm eliminando però così i bordi dove il rapporto segnale-rumore è peggiore. L'errore quadratico medio a metà dinamica passa da 2.5 mV del caso precedente a 4.2 mV, non quadruplica quindi. Ciò significa che l'incertezza della misura è dovuta fino a un certo

valore, intorno ai 2.5mV, ad altre cause che intervengono nell'uscita al DAC (come i disturbi dovuti all'elettronica digitale) e non al rumore nello stadio analogico. Oltre questo valore la deviazione standard è determinata dal rumore.
Usando un coefficiente intermedio tra i due precedenti pari a 83400 si ottine una devizione standar pari a 3.5 mV, determinata quindi in parte dai disturbi e in parte dal rumore. Si ottiene una sensibilità di 1 V/mm che comporta un range di misura di circa 3.5 mm con una deviazione standard in distanza pari a 3.5 μm.
Ricapitoliamo nella **Tabella 5.1**

| i2 | 41700 | 83400 | 166800 |
|---|---|---|---|
| Deviazione standard | 5 μm | 3.5 μm | 2.1 μm |
| Range | 7.5 mm | 3.6 mm | 1.8 mm |
| Range/Dev. stand. | 1500 | 1000 | 850 |

**Tabella 5.1** Confronto prestazioni con inidici i2 diversi

Da cui si decide di scegliere i2 = 166800 che ha la migliore sensibilità con un range ancora accettabile.

## 5.3 Seconda configurazione

Passiamo ora alla realizzazione del triangolatore usando però lo stadio a gate integrator di cui riportiamo in seguito lo schema elettrico completo.

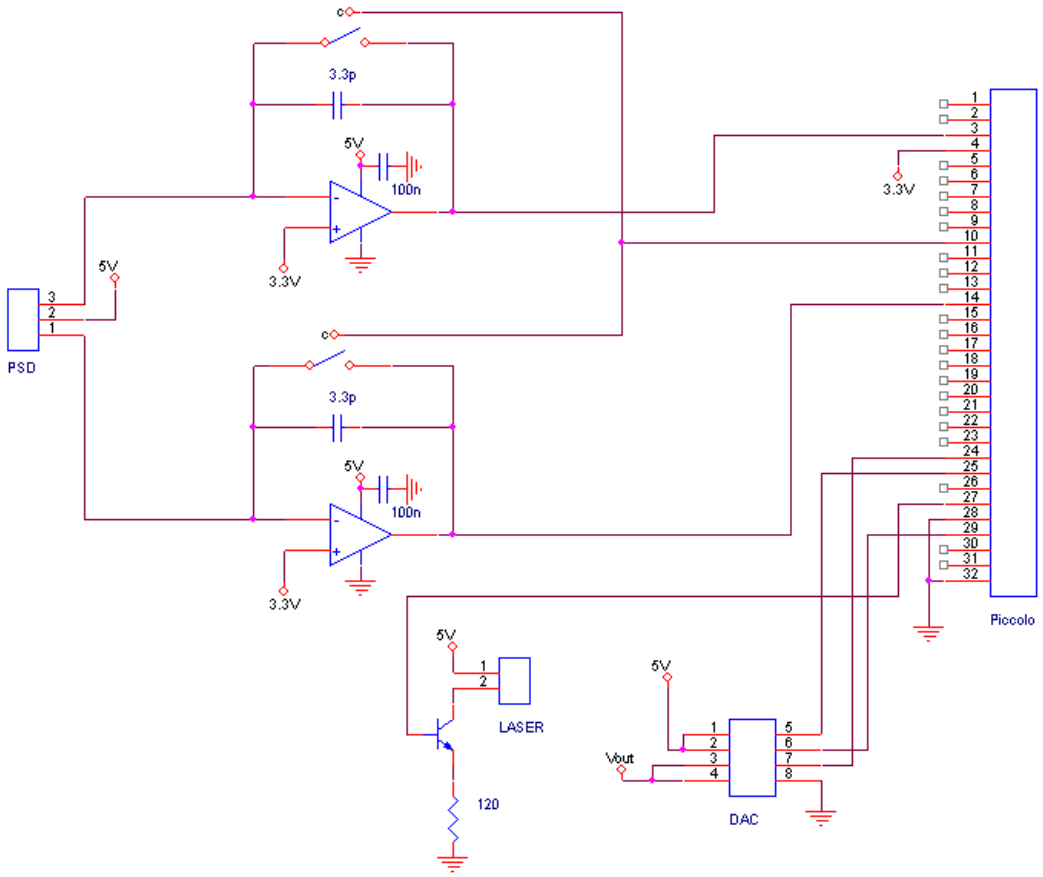



**Figura 5.11** Schema elettrico completo del triangolatore con gate integrator

A differenza del triangolatore con la transimpedenza qua serviranno le forme d'onda PWM. Il LASER funzionerà in maniera impulsata e il campionamento sarà sincrono col LASER. Per il resto, la configurazione del *Piccolo* resta identica.

In questo inoltre caso il *Piccolo* risulta fondamentale per il funzionamento dello stadio analogico, quindi solo a questo punto se ne potrà provare il funzionamento di tale stadio, mentre nello stadio a transimpedenza si era potuto verificare il funzionamento

istantaneamente.

Per precauzione si vedono le uscite degli operazionali con gli oscilloscopi prima di connetterli direttamente agli ingressi degli ADC per prevenire un possibile mal funzionamento che potrebbe portare a segnali troppo intensi che potrebbero bruciare gli stadi di ingresso degli ADC.

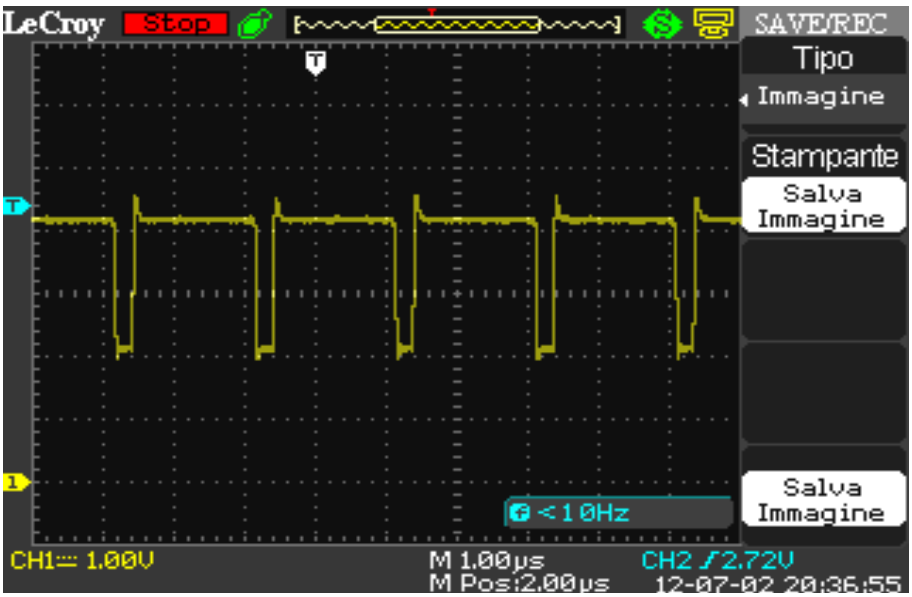



**Figura 5.12** Saturazione operazionali a causa della charge injection

Come si vede nella fase a buffer lo stadio funziona correttamente, mentro nella fase integrante le uscite degli operazionali saturano a 5 V. La causa di questo effetto si scoprirà essere la charge injection della transiimpendenza.

La charge injection è un effetto causato dalla capacità parassita presente nello switch, che per semplicità considereremo ora solo come un semplice nMOS.

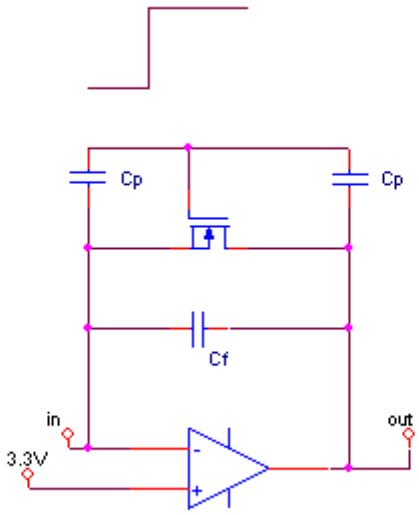



**Figura 5.13** Schema elettrico gate integrator con pass transistor con parassiti

Il gradino di tensione proveniente dal segnale di comando genererà una carica nelle due facce delle capacità parassite. Dato però che la carica non si crea e non si distrugge ci saranno anche le cariche complementari che dovranno scaricarsi da qualche parte per mantenere i potenziali d'uscita e del morsetto negativo costanti. Nella fase integrante in un caso le cariche andranno a scaricarsi nel generatore d'uscita dell'operazionale, mentre nell'altro la retroazione le farà finire nella capacità di retroazione alterando il valore dell'uscita. Nella fase a buffer tutte le cariche si scaricheranno nell'uscita dell'operazionale. Nel caso del trasmission gate si ha anche il pMOS che bilancia un po' l'effetto, ma la cancellazione non è perfetta.

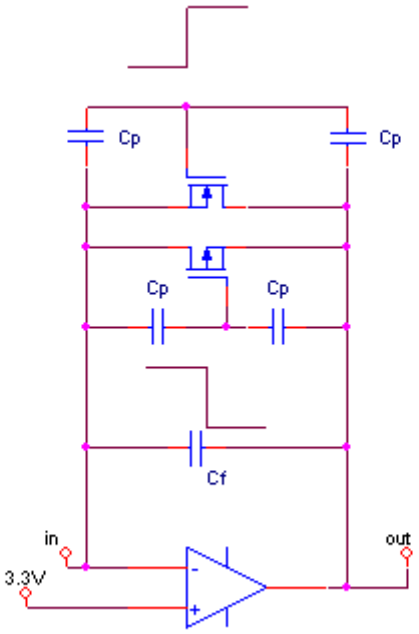



**Figura 5.14** Schema elettrico gate integrator con trasmission gate con parassiti

Guardando il datasheet del *MAX392* si vede che l'entità della charge injection è garantita minore di 5 pC che causerebbero una variazione di tensione pari a

$$\Delta V = \frac{\Delta Q}{C} = \frac{5\ pC}{3.3\ pF} = 1.5\ V \qquad \textbf{(5.23}$$

che fanno saturare l'operazionale.

Per contrastare questo problema si abbassa la tensione ai morsetti positivi degli operazionali con un trimmer, abbassando cosi di conseguenza anche la tensione ai morsetti negativi per via della retroazione cosi che la charge injection non faccia più saturare gli operazionali e porti il salto di tensione a valori ammissibili dalla dinamica d'ingresso degli ADC del *Piccolo*. Ciò perchè a disposizione si ha solo questo tipo di trasmission gate e aumentare l'entità della capacità in retroazione per rendere meno rilevante la variazione di tensione porterebbe ad un incremento notevole del periodo di integrazione per avere la

stessa sensibilià portando ad una forte limitazione in banda.

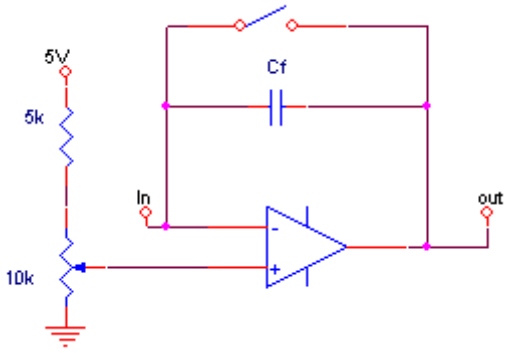



**Figura 5.15** Gate integrator con trimmer di regolazione

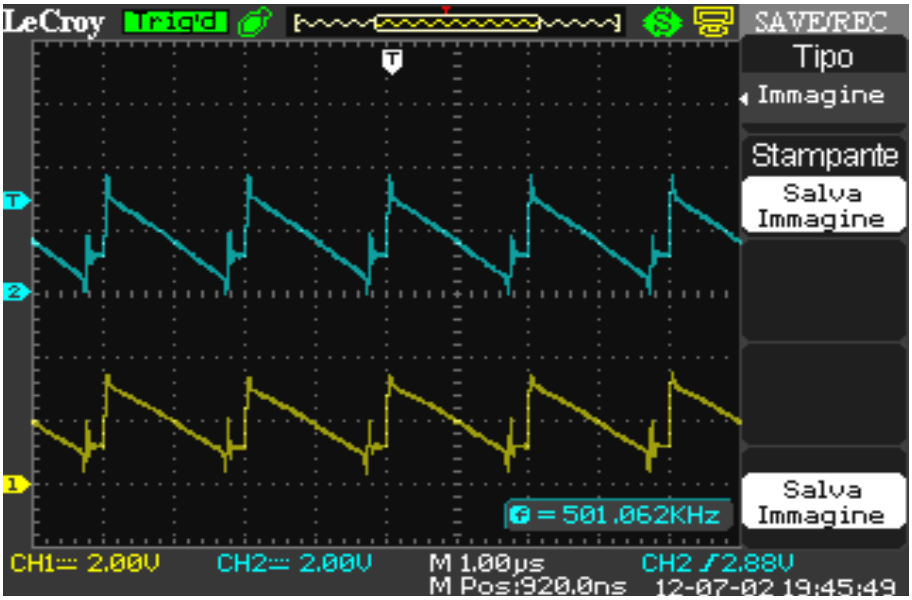



**Figura 5.16** Forme d'onda in uscita dai gate integrator con trimmer

Variando dinamicamente il valore del trimmer si possono vedere le forme d'onda cambiare fino a portare ad un risultato ottimale come quello portato in **Figura 5.16**. Si può vedere chiaramente l'effetto della corrente integrata e del gradino di tensione causato dal charge injection nella fase di apertura dello switch.

Valutando l'uscita del DAC e i registri degli ADC via software si vedono dei valori campionati molto perturbati che rendono la misura assolutamente inaffidabile. Ciò perchè la finestra di campionamento risulta troppo a ridosso dell'istante di chiusura dello switch che cortocircuitando la capacità in retroazione altera il valore del segnale campionato. Per rimediare è suffciene anticipare di qualche ciclo di clock l'istante in cui scatenare il segnale di trigger del SOC variando il valore di CMPA nei registri del PWM.

```
EPwm1Regs.CMPA.half.CMPA   = periodopwm - 30;
```

Adesso la misura risulta molto più stabile e affidabile ma con un fondo di rumore ancora troppo elevato.

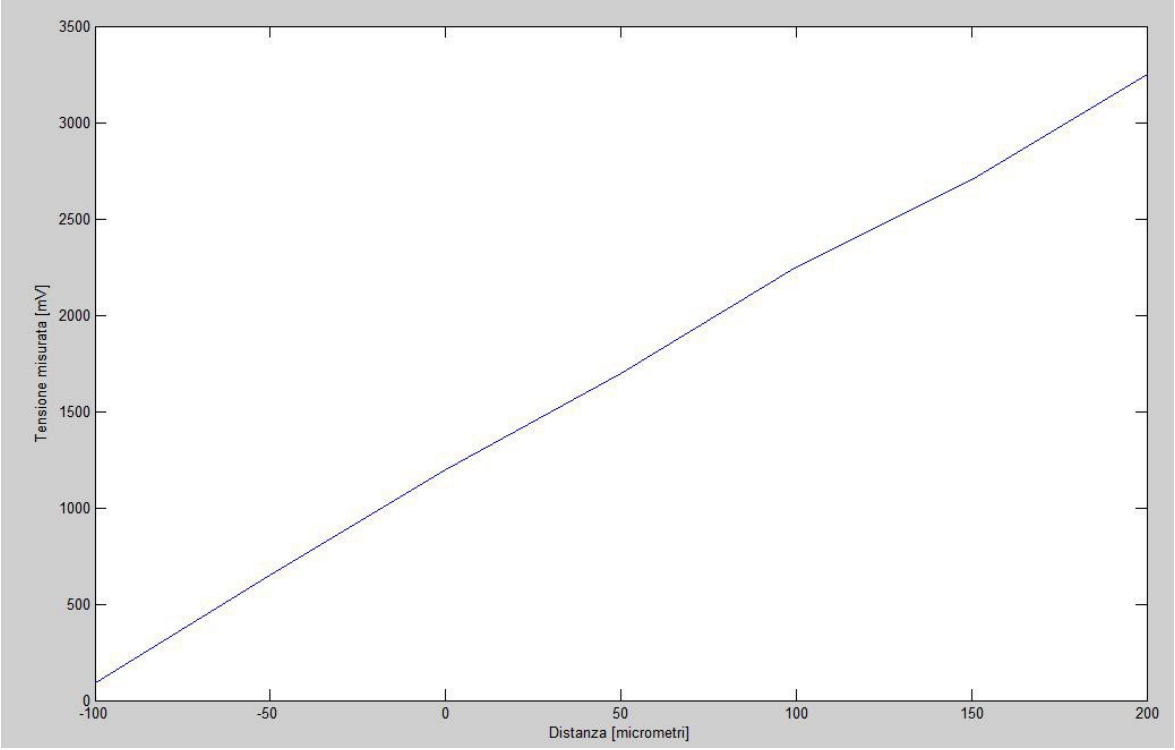



**Figura 5.17** Caratteristica statica triangolatore con gate integrator

Le prestazioni del triangolatore con transimpedenza risultano ancora superiori a quelle del triangolatore con stadio integrante.

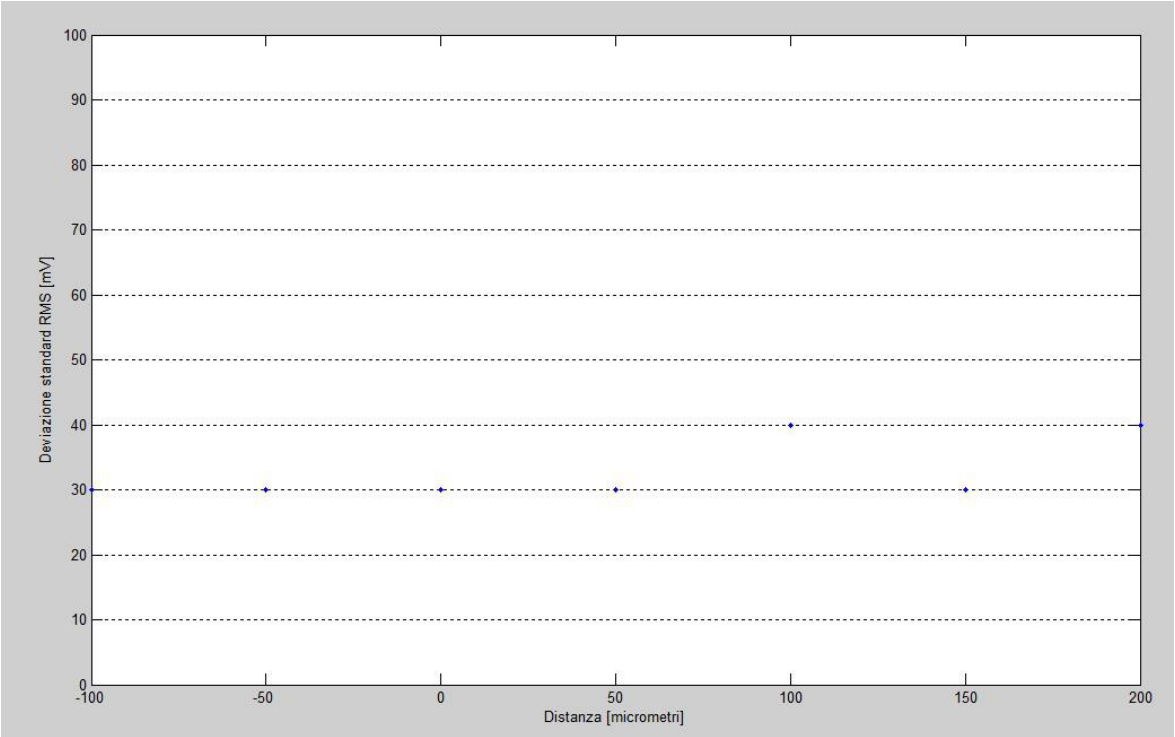



**Figura 5.18** Deviazione standard del triangolatore con gate integrator

La deviazione standard è circa costante in tutto il range e risulta pari a 27 μm, valore eccessivo. Qui si è impostato un coefficiente di moltiplicazione molto elevato dato che i segnali digitali che viaggiano nel triangolatore hanno incrementato i disturbi in uscita del DAC fino a 25 $mV_{RMS}$.

Per migliorare le cose si sfrutta le versatilità del DSP a cui basta applicare delle modifiche software. Si è pensato di eseguire un integrazione della corrente di buio che poi si sottrarrebbe al valore della misura eseguendo così un forte filtraggio passa alto.

L'acquisizione della misura si divide così in quattro fasi: la prima in cui viene integrata la corrente di buio più la fotocorrente generata dal LASER, la seconda in cui si cortocircuita la capacità di retroazione, la terza in cui il LASER è spento però l'operazionale viene fatto lavorare ancora come integratore per integrare la corrente di buio e in fine la quarta fase in cui viene ancora cortocircuitata la capacità di retroazione.

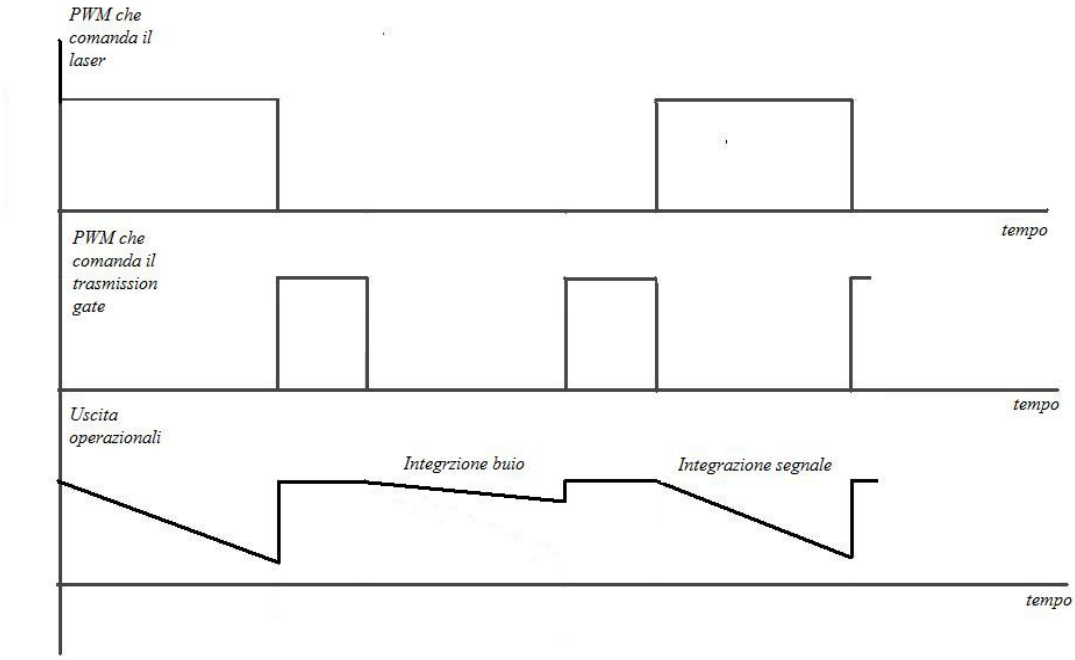



**Figura 5.19** Forme PWM e uscite degli operazionali

Viene creata un'altra coppia di SOC che lavorano in modo simultaneo per campionare i valori risultanti dall'integrazione della corrente di buio e salvare i valori cosi campionati su altri due registri.

```
AdcRegs.ADCSAMPLEMODE.bit.SIMULEN2 = 1;
```

Questi valori andranno poi sottratti ai valori risultanti dall'integrazione del fascio LASER.

```
ADCRESULTA0 = AdcResult.ADCRESULT0 - AdcResult.ADCRESULT2;
ADCRESULTB0 = AdcResult.ADCRESULT1 - AdcResult.ADCRESULT3;
```

Per realizzare questa coppia di SOC è necessario usare un altro submodulo PWM che generi un altro segnale di trigger. Inoltre bisognerà modificare l'istante in cui il primo submodulo genera il suo trigger per la prima coppia di SOC.

```
EPwm1Regs.ETSEL.bit.SOCAEN = 1;
EPwm1Regs.ETSEL.bit.SOCASEL     = 4;
EPwm1Regs.ETPS.bit.SOCAPRD      = 1;
EPwm1Regs.CMPA.half.CMPA   = periodopwm/2 - 30;
EPwm1Regs.TBCTL.bit.CTRMODE     = 0;
EPwm1Regs.TBPRD = periodopwm;
```

```
EPwm2Regs.ETSEL.bit.SOCAEN = 1;
EPwm2Regs.ETSEL.bit.SOCASEL     = 4;
EPwm2Regs.ETPS.bit.SOCAPRD      = 1;
EPwm2Regs.CMPA.half.CMPA   =  periodopwm - 30;
EPwm2Regs.TBCTL.bit.CTRMODE     = 0;
EPwm2Regs.TBPRD = periodopwm;
```

Ed ora non resta che impostare i registri della seconda coppia di SOC per campionare negli stessi pin della prima coppia di SOC usando però l'impulso proveniente dal secondo submodulo PWM come segnale di trigger.

```
AdcRegs.ADCSOC2CTL.bit.CHSEL    = 0;
AdcRegs.ADCSOC2CTL.bit.TRIGSEL  = 7;
AdcRegs.ADCSOC2CTL.bit.ACQPS    = 6;
```

La forma d'onda PWM che comanderà il trasmission gate dovrà avere però un periodo dimezzato rispetto a prima, quindi bisognerà usare un ulteriore submodulo PWM, sincrono agli altri due, però che lavori a frequenza doppia.

```
EPwm4Regs.CMPA.half.CMPA   = periodopwm/2 - 15;
EPwm4Regs.TBCTL.bit.CTRMODE     = 0;
EPwm4Regs.TBPRD = periodopwm/2;
EPwm4Regs.AQCTLA.bit.ZRO = 1;
EPwm4Regs.AQCTLA.bit.CAU = 2;
```

Tutte le necessarie modifiche software sono state realizzate, teoricamente resta solo un

ultima e unica modifica analogica da fare, dato che il perido di integrazione è circa dimezzato, bisognerà sostituire la capacità di retroazione con una più piccola di valore. Prima di fare tale operazione però si lascia quella da 3.3 pF per avere prima un idea sull'utilità delle modifiche fatte, e poi, in caso positivo, si procederà alla sostituzione della capacità.

In seguito vengono riportate le forme d'onda in uscita dagli operazionali e la caratteristica statica ottenuta.

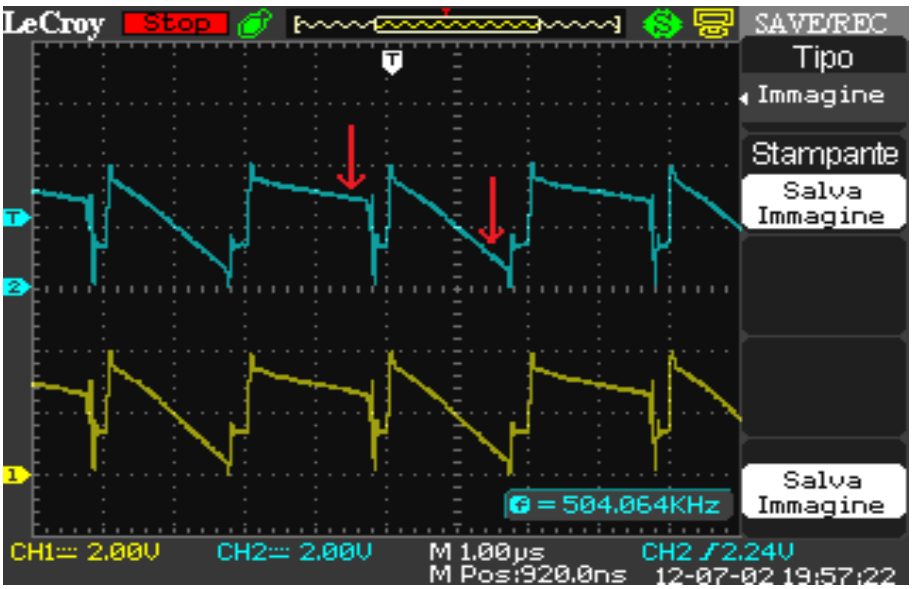



**Figura 5.20** Uscite operazionali a 4 fasi

Nella forme d'onda si vede che l'effetto della charge injection è differente nella fase di integrazione della corrente di buio rispetto a quella di integrazione della fotocorrente generata dal fascio laser. Probabilmente la forma d'onda PWM che pilota il LASER genera anche lei un induzione di carica che altera il valore della capacità in retroazione.

Le prestazioni del sistema sono praticamente uguali a prima, l'integrazione della corrente di buio non ha portato risultati dato che l'iniezione di carica non è costante, infatti guardano su un periodo di tempo più lungo si vede che i picchi variano, inoltre la sostituzione con una capacità più piccola renderebbe ancora più evidenti questi effetti.

Dopo alcune riflessioni si è notato che nelle forme d'onda in uscita dagli operazionali, la

cosa apparentemente più stabile sia la pendenza dell'integrazione della fotocorrente. Si è pensato perciò di tornare a sole due fasi, eliminando perciò l'integrazione della corrente di buio, però sfruttare la seconda coppia di SOC per effettuare un campionamento all'inizio della rampa, mentre un altro alla fine, per poi farne ancora la differenza.

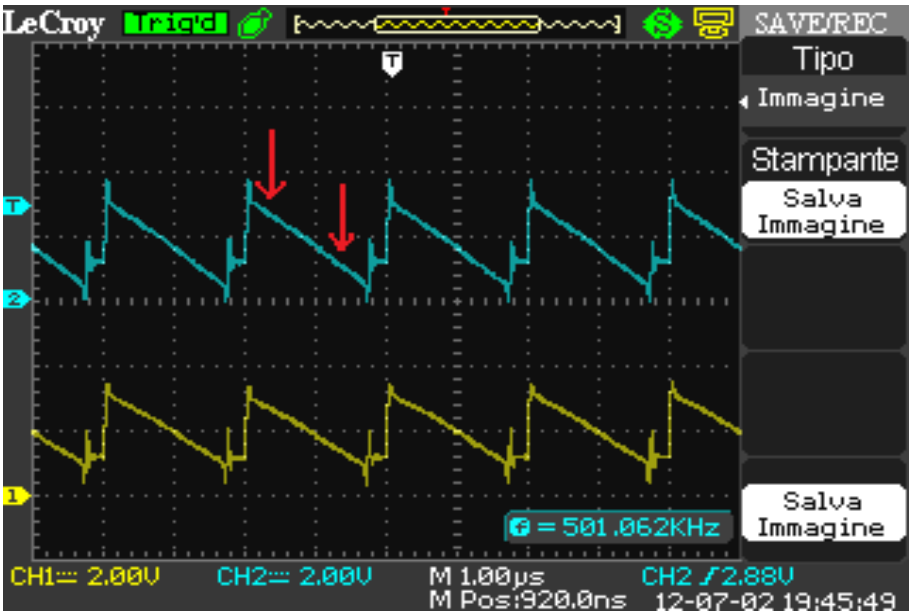



**Figura 5.21** Uscite operazionali a 2 fasi con doppio campionamento per periodo

Questa è risultata la mossa vincente, infatti le prestazioni sono notevolmente migliorate, si ottiene una deviazione standard di 1.2 µm costante in tutto il range con una dinamica di misura di 1 mm. In questa maniera si elimina completamente l'effetto della charge injection, infatti prima veniva sottratta la variazione di tensione causata dalla charge injection del periodo prima, mentre qua viene sottratta la variazione di tensione della charge injection dello stesso periodo.

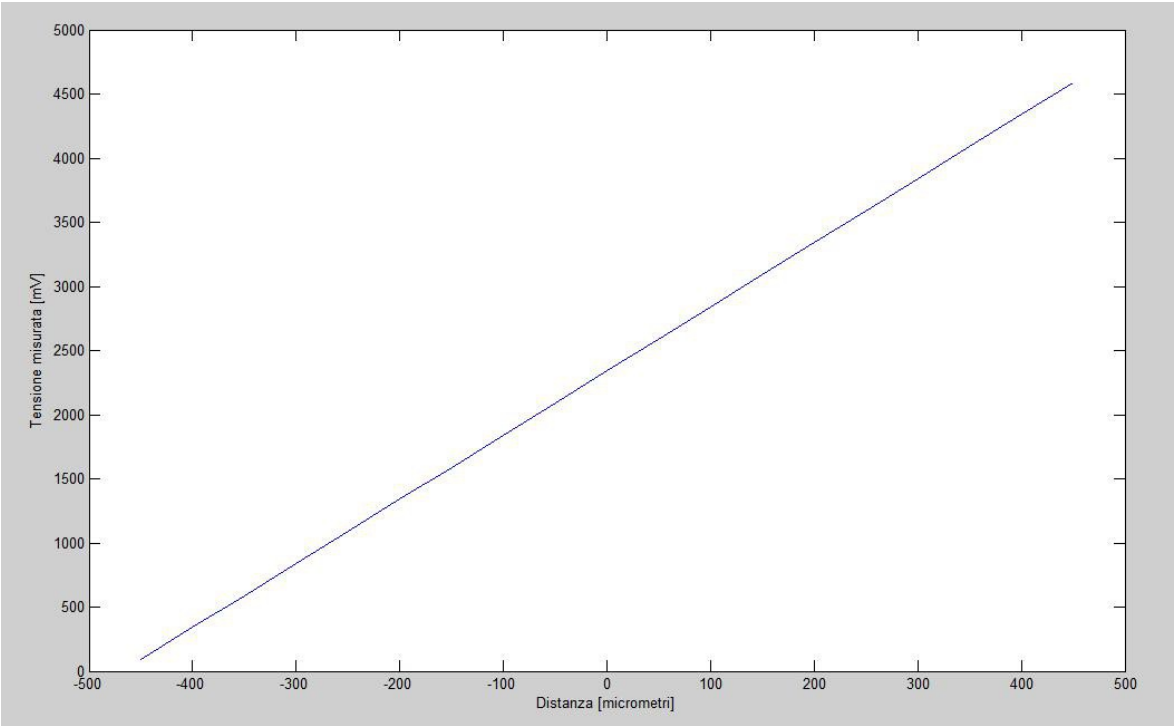



**Figura 5.22** Caratteristica statica del triangolatore finale

In questo caso si arriva a 5 V perchè si è pensato di modificare $V_{ref}$ al DAC per ottenere un segnale più ampio di tensione.

# CONCLUSIONI

Il lavoro di tesi è stato molto stimolante rendendo possibile l'applicazione della teoria ad un progetto realmente esistente e vedendo la presenza di effetti che non mostrano il loro reale peso nella progettazione accademica portando alla soddisfazione nel momento in cui le specifiche vengono ottenute.

Lo strumento è stato realizzato con prestazioni che superano quelle dei triangolatori al momento più veloci in mercato che raggiungono i 50 ksample/sec contro i quasi 250 ksample/sec di quello appena descritto.

Si è vista la difficoltà nella realizzazione di componenti switching discreti di cui è stato reso possibile l'impiego dato che le frequenze di lavoro non erano troppo elevate.

Si è vista inoltre la suscettibilità delle misure dai disturbi elettromagnetici ed elettrostatici e la suscettibilità degli operazionali, i quali oscillavano molto facilmente se non schermati adeguatamente dall'elettronica digitale.

In seguito viene mostrata una figura del triangolatore aperto.

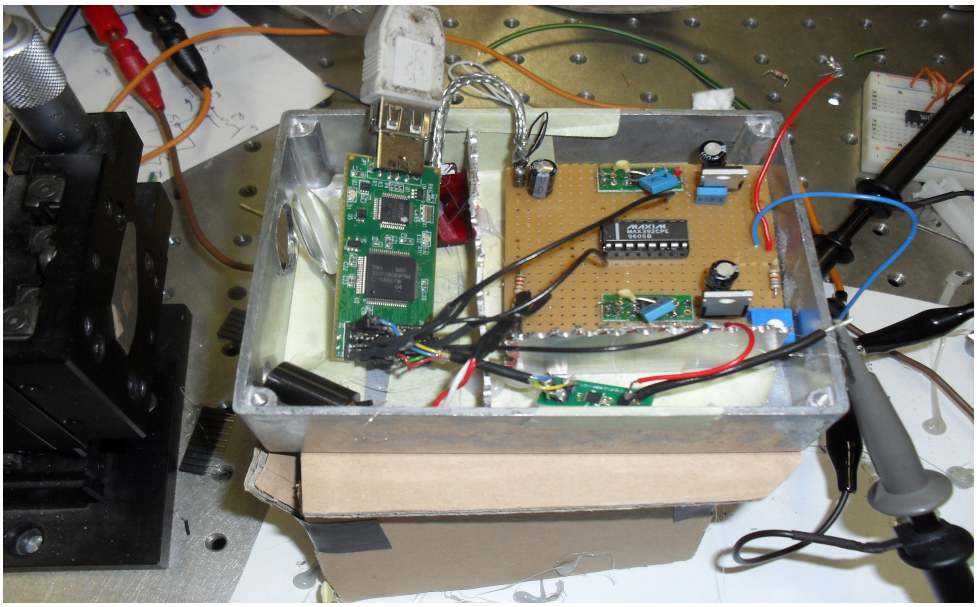


**Figura 6** Foto triangolatore

# BIBLIOGRAFIA

[1] G. Genta, "Vibration of Structures and Machines - Practical Aspects", Second Edition, Springer-Verlag

[2] Cyril M. Harris, "Shock and Vibration Handbook", Fourth Edition, Mc.GRAW-HILL

[3] S.Donati, “Electro-optical Instrumentation – Sensing and Measuring with Lasers”, Prentice Hall, Upper Saddle River

[4] M.Norgia, A.Pesatori, C.Svelto, “Novel Interferometric for the Measurement of Laser Wavelenght/Frequency-Modulation Sensitivity”

[5] A.L.Lacaita, M. Sampietro, “Circuiti elettronici”, CittàStudi

[6] F.Zappa, R.Zappa, “Sistemi Elettronici”, Esculapio

[7] S.Cova, “Dispense di Sensori, Segali e Rumore”

[8] C.Prati, “Segnali e sistemi per le telecomunicazioni”, Mc.GRAW- HILL

[9] C. Svelto, G. Galzerano, “*Frequency-Stabilized Near-Infrared Solid-State Lasers”*, Recent Research Developments in Applied Physics

[10] S.K.Mitra, “Digital Signal Processing”, Mc.GRAW-HILL

# RINGRAZIAMENTI

Giunto così all'ultima pagina della tesi giungo anche all'ultima pagina della mia carriera universitaria e quindi anche all'ultima pagina della mia vita da studente che i miei genitori hanno reso possibile. Desidero quindi ringraziare inizialmente loro che mi hanno sostenuto, rimproverato, mantenuto durante tutti questi anni e che mi hanno permesso di conseguire a questo titolo di studio così importante che spero renda un po' più facile il mio futuro.

Quindi ringrazio anche il Prof. Michele Norgia che mi ha concesso un posto nel Laboratorio di Misure dando vita a questi tesi di laurea, seguendomi e aiutandomi nei momenti di difficoltà che da solo avrei superato solo in un lungo tempo e oltre al suo aiuto non posso dimenticare quello di Alessandro Magnani che ringrazio profondamente per tutta la pazienza con cui rispondeva alle mie domande e alla grande disponibilità.

Un doveroso ringraziamento va alla mia dolce ragazza Domenica che mi ha sempre mostrato molta comprensione e pazienza, aiutato quando era in grado di farlo e soprattutto donato molto affetto.

Come dimenticare poi tutti i miei amici di università e di laboratorio che hanno reso lo studio e il lavoro un po' meno pesante e spesso divertente. In particolare ringrazio Simone, Dario, Marco M., Marco P., Edo, Mauro, Luca, Franceso, Paolo e Nico.

Ringrazio mio fratello Xhoe su cui ho sempre potuto contare e su cui so potrò contare sempre e tutti i miei amici fuori dall'università che mi hanno accompagnato in tutti questi anni di studi non solo universitari tra cui Manuel, Maisa, Mono, Simone, Andy, Lopre, Baro e Teo.

In fine non posso dimenticare l'unica figura che è sempre sempre stata presente in laboratorio, la Manichina del Laboratorio.

Misuratore laser a triangolazione a banda larga

# Appendice A

# Codice

```
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include
File

// Prototype statements for functions found within this file.
interrupt void adc_isr(void);
void Adc_Config(void);
void spi_fifo_init(void);
void spi_init(void);

//  Variabili globali:
float32 ADCRESULTA0=0;
float32 ADCRESULTB0=0;
float32 diffA0B0=0;

// variabili filtro del primo ordine

float32 y1=0;
float32 y2=0;
float32 x2=0;

// coefficienti filtro del primo ordine

float32 a1=0.334511170731938;
float32 a2=0.334511170731938;
float32 a3=0.330977658536123;

// coefficienti filtro del secondo ordine

float32 b1=0.143940904262370;
float32 b2=0.287881808524741;
float32 b3=0.143940904262370;
float32 b4=0.851513722288680;
float32 b5=-0.427277339338162;

// variabili filtro del secondo ordine

float32 t2=0;
float32 t3=0;
float32 w1=0;
float32 w2=0;
float32 w3=0;
```

```
int32 i=0;
int32 i2=1000;
Uint16 sdata=0xAAAA;  // send data
int32 periodopwm=80;

main()
{

// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the F2806x_SysCtrl.c file.
   InitSysCtrl(); // CLK_System=80MHz :  SysCtrlRegs.LOSPCP.all = 0x0000;


// Step 2. Initialize GPIO:
// This example function is found in the F2806x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio();  // Skipped for this example

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
   DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the F2806x_PieCtrl.c file.
   InitPieCtrl();

// Inizializzazioni SPI: (CLK=20MHz): SysCtrlRegs.XCLK.bit.XCLKOUTDIV=2;
   spi_fifo_init();       // Initialize the Spi FIFO
   spi_init();            // init SPI
   InitSpiaGpio();


// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;
   IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in F2806x_DefaultIsr.c.
// This function is found in F2806x_PieVect.c.
   InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
   EALLOW;  // This is needed to write to EALLOW protected register
   PieVectTable.ADCINT1 = &adc_isr;
   EDIS;    // This is needed to disable write to EALLOW protected
```

```
registers

// Step 4. Initialize all the Device Peripherals:
// This function is found in F2806x_InitPeripherals.c
// InitPeripherals(); // Not required for this example
   InitAdc();  // For this example, init the ADC

// Step 5. User specific code, enable interrupts:

// Enable ADCINT1 in PIE
//   PieCtrlRegs.PIEIER1.bit.INTx1 = 1;  // Enable INT 1.1 in the PIE
//   IER |= M_INT1;                                    // Enable CPU
Interrupt 1
   EINT;                                             // Enable Global
interrupt INTM
   ERTM;                                             // Enable Global
realtime interrupt DBGM

   EALLOW;
   GpioCtrlRegs.GPBMUX1.bit.GPIO34=0;   // è GPIO
   GpioCtrlRegs.GPBDIR.bit.GPIO34=1;  //imposta OUT.
   EDIS;

// Configure ADC
      EALLOW;
    AdcRegs.ADCCTL2.bit.ADCNONOVERLAP = 1;      // Enable non-overlap
mode
      AdcRegs.ADCCTL1.bit.INTPULSEPOS     = 1;  // ADCINT1 trips after
AdcResults latch
      AdcRegs.INTSEL1N2.bit.INT1E     = 1;      // Enabled ADCINT1
      AdcRegs.INTSEL1N2.bit.INT1CONT  = 0;      // Disable ADCINT1
Continuous mode
    AdcRegs.INTSEL1N2.bit.INT1SEL   = 3;    // setup EOC3 to trigger
ADCINT1 to fire
    AdcRegs.ADCSAMPLEMODE.bit.SIMULEN0 = 1; // accoppia SOC0 e SOC1

    AdcRegs.ADCSAMPLEMODE.bit.SIMULEN2 = 1;
//    AdcRegs.ADCSAMPLEMODE.bit.SIMULEN4 = 1;
//    AdcRegs.ADCSAMPLEMODE.bit.SIMULEN6 = 1;
//
//

    AdcRegs.ADCSOC0CTL.bit.CHSEL    = 0;    // set SOC0 channel select to
ADCINA0
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL  = 5;    // set SOC0 start trigger on
EPWM1A, due to round-robin SOC0 converts first then SOC1
      AdcRegs.ADCSOC0CTL.bit.ACQPS  = 6;  // set SOC0 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

//faccio altri soc

    AdcRegs.ADCSOC2CTL.bit.CHSEL    = 0;    // set SOC0 channel select to
ADCINA0
    AdcRegs.ADCSOC2CTL.bit.TRIGSEL  = 7;    // set SOC0 start trigger on
```

```
EPWM1A, due to round-robin SOC0 converts first then SOC1
      AdcRegs.ADCSOC2CTL.bit.ACQPS  = 6;  // set SOC0 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

//    AdcRegs.ADCSOC4CTL.bit.CHSEL  = 0;    // set SOC0 channel select to
ADCINA0
//    AdcRegs.ADCSOC4CTL.bit.TRIGSEL      = 9;    // set SOC0 start
trigger on EPWM1A, due to round-robin SOC0 converts first then SOC1
//    AdcRegs.ADCSOC4CTL.bit.ACQPS  = 6;  // set SOC0 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)
//
//    AdcRegs.ADCSOC6CTL.bit.CHSEL  = 0;    // set SOC0 channel select to
ADCINA0
//    AdcRegs.ADCSOC6CTL.bit.TRIGSEL      = 13;    // set SOC0 start
trigger on EPWM1A, due to round-robin SOC0 converts first then SOC1
//    AdcRegs.ADCSOC6CTL.bit.ACQPS  = 6;  // set SOC0 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

      EDIS;

// Configure PWM. Assumes ePWM1 clock is already enabled in
InitSysCtrl();



   EPwm1Regs.ETSEL.bit.SOCAEN = 1;        // Enable SOC on A group
   EPwm1Regs.ETSEL.bit.SOCASEL      = 4;        // Select SOC from CMPA
on upcount
   EPwm1Regs.ETPS.bit.SOCAPRD       = 1;        // Generate pulse on 1st
event    conta quanti eventi servono per generare l impulso
   EPwm1Regs.CMPA.half.CMPA   = periodopwm- 60; // Set compare A value
campiono l integrazione del laser
   EPwm1Regs.TBCTL.bit.CTRMODE      = 0;        // count up and start
   EPwm1Regs.TBPRD = periodopwm;                     // 40MHz/TBPRD =
Fadc = Fesecuz.  Fadc_MAX = 2 MSa/s !!!
                                                                  //
Quindi TBPRD>= 20 !!! Se non non gli sta dietro...
// creo 3 pwm di SOC

   EPwm2Regs.ETSEL.bit.SOCAEN = 1;        // Enable SOC on A group
   EPwm2Regs.ETSEL.bit.SOCASEL      = 4;        // Select SOC from CMPA
on upcount
   EPwm2Regs.ETPS.bit.SOCAPRD       = 1;        // Generate pulse on 1st
event
   EPwm2Regs.CMPA.half.CMPA   =  periodopwm - 25;     // Set compare A
value    campiono l integrazione della corrente di buio
   EPwm2Regs.TBCTL.bit.CTRMODE      = 0;        // count up and start
   EPwm2Regs.TBPRD = periodopwm;                     // 40MHz/TBPRD =
Fadc = Fesecuz.  Fadc_MAX = 2 MSa/s !!!
                                                                  //
Quindi TBPRD>= 20 !!! Se non non gli sta dietro...
```

```
//   inizio prove

   EPwm1Regs.DBCTL.bit.OUT_MODE    = 1;             // salta il ritardo
della deadband
   EPwm2Regs.DBCTL.bit.OUT_MODE    = 1;
   EPwm4Regs.DBCTL.bit.OUT_MODE    = 1;



   EPwm1Regs.TBCTL.bit.PHSEN  =  1;           // abilita la fase del pwm
(serve x sincronizzare)
   EPwm2Regs.TBCTL.bit.PHSEN  =  1;
   EPwm3Regs.TBCTL.bit.PHSEN  =  1;
   EPwm4Regs.TBCTL.bit.PHSEN  =  1;
   EPwm5Regs.TBCTL.bit.PHSEN  =  1;

   EPwm5Regs.TBCTL.bit.SWFSYNC     =  1;
   EPwm4Regs.TBCTL.bit.SWFSYNC     =  1;
   EPwm3Regs.TBCTL.bit.SWFSYNC     =  1;
   EPwm2Regs.TBCTL.bit.SWFSYNC     =  1;

   EPwm4Regs.TBCTL.bit.SYNCOSEL    =  0;
   EPwm3Regs.TBCTL.bit.SYNCOSEL    =  0;
   EPwm2Regs.TBCTL.bit.SYNCOSEL    =  0;
   EPwm1Regs.TBCTL.bit.SYNCOSEL    =  1;

//    fine prove

//
//   EPwm3Regs.ETSEL.bit.SOCAEN    = 1;        // Enable SOC on A group
//   EPwm3Regs.ETSEL.bit.SOCASEL   = 4;        // Select SOC from CMPA
on upcount
//   EPwm3Regs.ETPS.bit.SOCAPRD    = 1;        // Generate pulse on 1st
event
//   EPwm3Regs.CMPA.half.CMPA      = 1 + periodopwm/2;     // Set compare
A value
//   EPwm3Regs.TBCTL.bit.CTRMODE   = 0;        // count up and start
//   EPwm3Regs.TBPRD = periodopwm;                  // 40MHz/TBPRD =
Fadc = Fesecuz.  Fadc_MAX = 2 MSa/s !!!
//                                                              //
Quindi TBPRD>= 20 !!! Se non non gli sta dietro...
//   EPwm5Regs.ETSEL.bit.SOCAEN    = 1;        // Enable SOC on A group
//   EPwm5Regs.ETSEL.bit.SOCASEL   = 4;        // Select SOC from CMPA
on upcount
//   EPwm5Regs.ETPS.bit.SOCAPRD    = 1;        // Generate pulse on 1st
event
//   EPwm5Regs.CMPA.half.CMPA      = 1 + 3*(periodopwm/4); // Set compare
A value
//   EPwm5Regs.TBCTL.bit.CTRMODE   = 0;        // count up and start
//   EPwm5Regs.TBPRD = periodopwm;                  // 40MHz/TBPRD =
Fadc = Fesecuz.  Fadc_MAX = 2 MSa/s !!!
//                                                              //
Quindi TBPRD>= 20 !!! Se non non gli sta dietro...
```

```
//


 //pwm da mandare al gpio

 // pwm di prova che controlla l interruttore della capacità di
retroazione   pin 17

   EPwm1Regs.CMPB = periodopwm/2;
   EPwm1Regs.AQCTLA.bit.ZRO = 1;         // clear on zero
   EPwm1Regs.AQCTLA.bit.CBU = 2;         // set in cmpB

   EALLOW;
   GpioCtrlRegs.GPAMUX1.bit.GPIO0=1;   // è GPIO
   GpioCtrlRegs.GPADIR.bit.GPIO0=1;  //imposta OUT.
   EDIS;

// pwm che controlla il laser   pin 27

   EPwm2Regs.CMPB = periodopwm-10;
   EPwm2Regs.AQCTLA.bit.ZRO = 2;         // set on zero
   EPwm2Regs.AQCTLA.bit.CBU = 1;         // clear in cmpB

   EALLOW;
   GpioCtrlRegs.GPAMUX1.bit.GPIO2=1;   // è GPIO
   GpioCtrlRegs.GPADIR.bit.GPIO2=1;  //imposta OUT.
   EDIS;


// pwm che controllerà l interruttore della capacità di retroazione
pin 11

   EPwm4Regs.CMPA.half.CMPA   = periodopwm-10;

   EPwm4Regs.TBCTL.bit.CTRMODE      = 0;
   EPwm4Regs.TBPRD = periodopwm;
   EPwm4Regs.AQCTLA.bit.ZRO = 1;         // clear on zero
   EPwm4Regs.AQCTLA.bit.CAU = 2;         // set in cmpa


   EALLOW;
   GpioCtrlRegs.GPAMUX1.bit.GPIO6=1;   // è GPIO
   GpioCtrlRegs.GPADIR.bit.GPIO6=1;  //imposta OUT.
   EDIS;

// Wait for ADC interrupt
   for(;;)
   {
      while(AdcRegs.ADCINTFLG.bit.ADCINT1==0)
            {}
      EALLOW;
    AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;      //Clear ADCINT1 flag
reinitialize for next SOC
      // START PROGRAMMA:
```

```
      GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;

  ADCRESULTA0 = AdcResult.ADCRESULT0 - AdcResult.ADCRESULT2;
  ADCRESULTB0 = AdcResult.ADCRESULT1 - AdcResult.ADCRESULT3;


  diffA0B0=(ADCRESULTA0-ADCRESULTB0)/(ADCRESULTA0+ADCRESULTB0);
  diffA0B0=diffA0B0*i2;
  diffA0B0=diffA0B0 + 8192;


  //filtro butterworth del terzo ordine come cascata di un filtro del
primo ordine e uno del secondo
  //sostituzione bilineare

  y1=(a1*diffA0B0) + (a2*x2) + (a3*y2);
  x2=diffA0B0;
  y2=y1;


  w1=b1*y1 + b2*t2 + b3*t3 + b4*w2 + b5*w3;
  w3=w2;
  w2=w1;
  t3=t2;
  t2=y1;


  sdata=sdata + w1;


    if(i==4)
      {
//    sdata=30000;
      SpiaRegs.SPIDAT = sdata>>8;
    SpiaRegs.SPITXBUF = sdata<<8;
    sdata=0;

    i=0;
      }
      i++;
  EDIS;
      // FINE PROGRAMMA.

   } // Close for infinito
} // Close main



interrupt void  adc_isr(void)
{
//  EALLOW;
//  AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;      //Clear ADCINT1 flag
```

```
reinitialize for next SOC
//  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge interrupt to
PIE
//  ADCRESULTA0=AdcResult.ADCRESULT0;
//  ADCRESULTB0=AdcResult.ADCRESULT1;
//
//
//  diffA0B0=(ADCRESULTA0-ADCRESULTB0)/(ADCRESULTA0+ADCRESULTB0);
//  diffA0B0=diffA0B0*i2;
//  diffA0B0=diffA0B0 + 32767;
//
//  //filtro butterworth del terzo ordine come cascata di un filtro del
primo ordine e uno del secondo
//  //sostituzione bilineare
//
//  y1=a1*diffA0B0 + a2*x2 + a3*y2;
//  x2=diffA0B0;
//  y2=y1;
//
//
//  w1=b1*y1 + b2*t2 + b3*t3 + b4*w2 + b5*w3;
//  w3=w2;
//  w2=w1;
//  t3=t2;
//  t2=y1;
//
//  sdata=w1;
//
//  i=1;
//  EDIS;
  return;
}

void spi_init()
{
      SpiaRegs.SPICCR.all =0x000F;              // Reset on, rising
edge, 16-bit char bits
      SpiaRegs.SPICTL.all =0x0006;                  // Enable master
mode, normal phase, enable talk, and SPI int disabled.
      SpiaRegs.SPIBRR =0x0000;                                  //   BIT RATE
DIPENDENTE ANCHE DA LSPCLK

    SpiaRegs.SPICCR.all =0x009F;                  // Relinquish SPI from
Reset
    SpiaRegs.SPIPRI.bit.FREE = 1;               // Set so breakpoints
don't disturb xmission
    SpiaRegs.SPIPRI.bit.TRIWIRE = 1;
}


//  impostazioni iniziali
// void spi_init()
// {
//    SpiaRegs.SPICCR.all =0x000F;              // Reset on, rising
```

```
edge, 16-bit char bits
//    SpiaRegs.SPICTL.all =0x0006;                  // Enable master
mode, normal phase, enable talk, and SPI int disabled.
//    SpiaRegs.SPIBRR =0x0000;                                 //   BIT RATE
DIPENDENTE ANCHE DA LSPCLK

//      SpiaRegs.SPICCR.all =0x009F;                    // Relinquish
SPI from Reset
//      SpiaRegs.SPIPRI.bit.FREE = 1;                // Set so
breakpoints don't disturb xmission
// }


void spi_fifo_init()
{   //Initialize SPI FIFO registers
    SpiaRegs.SPIFFTX.all=0xA040;
    SpiaRegs.SPIFFRX.all=0x005f;
    SpiaRegs.SPIFFCT.all=0x0;
}



// TI File $Revision: /main/5 $
// Checkin $Date: March 18, 2011   14:04:59 $
//###########################################################################
####
//
// FILE:    F2806x_Adc.c
//
// TITLE:   F2806x ADC Initialization & Support Functions.
//
//###########################################################################
####
// $TI Release: 2806x C/C++ Header Files V1.10 $
// $Release Date: April 7, 2011 $
//###########################################################################
####

#include "F2806x_Device.h"     // F2806x Headerfile Include File
#include "F2806x_Examples.h"   // F2806x Examples Include File

#define ADC_usDELAY  1000L

//---------------------------------------------------------------------------
----
// InitAdc:
//---------------------------------------------------------------------------
----
// This function initializes ADC to a known state.
//
// NOTE: ADC INIT IS DIFFERENT ON F2806x DEVICES COMPARED TO OTHER 28X
DEVICES
//
```

```
void InitAdc(void)
{
    extern void DSP28x_usDelay(Uint32 Count);

    // *IMPORTANT*
    // The Device_cal function, which copies the ADC calibration values
from TI reserved
    // OTP into the ADCREFSEL and ADCOFFTRIM registers, occurs
automatically in the
    // Boot ROM. If the boot ROM code is bypassed during the debug
process, the
    // following function MUST be called for the ADC to function
according
    // to specification. The clocks to the ADC MUST be enabled before
calling this
    // function.
    // See the device data manual and/or the ADC Reference
    // Manual for more information.

        EALLOW;
        SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
        (*Device_cal)();
        EDIS;

    // To powerup the ADC the ADCENCLK bit should be set first to enable
    // clocks, followed by powering up the bandgap, reference circuitry,
and ADC core.
    // Before the first conversion is performed a 5ms delay must be
observed
    // after power up to give all analog circuits time to power up and
settle

    // Please note that for the delay function below to operate correctly
the
    // CPU_RATE define statement in the F2806x_Examples.h file must
    // contain the correct CPU clock period in nanoseconds.
    EALLOW;
    AdcRegs.ADCCTL1.bit.ADCBGPWD  = 1;      // Power ADC BG
    AdcRegs.ADCCTL1.bit.ADCREFPWD = 1;      // Power reference
    AdcRegs.ADCCTL1.bit.ADCPWDN   = 1;      // Power ADC
    AdcRegs.ADCCTL1.bit.ADCENABLE = 1;      // Enable ADC
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;      // Select interal BG
    EDIS;

    DELAY_US(ADC_usDELAY);         // Delay before converting ADC
channels

    EALLOW;
    AdcRegs.ADCCTL2.bit.CLKDIV2EN = 1;
    EDIS;

    DELAY_US(ADC_usDELAY);         // Delay before converting ADC
channels
}
```

```
void InitAdcAio()
{

   EALLOW;

/* Configure ADC pins using AIO regs*/
// This specifies which of the possible AIO pins will be Analog input
pins.
// NOTE: AIO1,3,5,7-9,11,13,15 are analog inputs in all AIOMUX1
configurations.
// Comment out other unwanted lines.

    GpioCtrlRegs.AIOMUX1.bit.AIO2 = 2;    // Configure AIO2 for A2
(analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO4 = 2;    // Configure AIO4 for A4
(analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO6 = 2;    // Configure AIO6 for A6
(analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO10 = 2;   // Configure AIO10 for B2
(analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO12 = 2;   // Configure AIO12 for B4
(analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO14 = 2;   // Configure AIO14 for B6
(analog input) operation

    EDIS;
}


/* AdcoffsetSelfCal-
   This function re-calibrates the ADC zero offset error by converting
the VREFLO reference with
   the ADC and modifying the ADCOFFTRIM register. VREFLO is sampled by
the ADC using an internal
   MUX select which connects VREFLO to A5 without sacrificing an external
ADC pin. This
   function calls two other functions:
   - AdcChanSelect(channel) - selects the ADC channel to convert
   - AdcConversion() - initiates several ADC conversions and returns the
average
*/
void AdcOffsetSelfCal()
{
    Uint16 AdcConvMean;
    EALLOW;
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;                 //Select internal
reference mode
    AdcRegs.ADCCTL1.bit.VREFLOCONV = 1;                //Select VREFLO
internal connection on B5
    AdcChanSelect(13);                                 //Select channel
B5 for all SOC
    AdcRegs.ADCOFFTRIM.bit.OFFTRIM = 80;               //Apply
artificial offset (+80) to account for a negative offset that may reside
```

```
in the ADC core
    AdcConvMean = AdcConversion();                      //Capture ADC
conversion on VREFLO
    AdcRegs.ADCOFFTRIM.bit.OFFTRIM = 80 - AdcConvMean;  //Set offtrim
register with new value (i.e remove artical offset (+80) and create a
two's compliment of the offset error)
    AdcRegs.ADCCTL1.bit.VREFLOCONV = 0;                 //Select external
ADCIN5 input pin on B5
    EDIS;
}

/*  AdcChanSelect-
    This function selects the ADC channel to convert by setting all SOC
channel selects to a single channel.

     * IMPORTANT * This function will overwrite previous SOC channel
select settings. Recommend saving
           the previous settings.
 */
void AdcChanSelect(Uint16 ch_no)
{
    AdcRegs.ADCSOC0CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC1CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC2CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC3CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC4CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC5CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC6CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC7CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC8CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC9CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC10CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC11CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC12CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC13CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC14CTL.bit.CHSEL= ch_no;
    AdcRegs.ADCSOC15CTL.bit.CHSEL= ch_no;
} //end AdcChanSelect

/* AdcConversion -
   This function initiates several ADC conversions and returns the
average. It uses ADCINT1 and ADCINT2
   to "ping-pong" between SOC0-7 and SOC8-15 and is referred to as "ping-
pong" sampling.

     * IMPORTANT * This function will overwrite previous ADC settings.
Recommend saving previous settings.
*/
Uint16 AdcConversion(void)
{
    Uint16 index, SampleSize, Mean, ACQPS_Value;
    Uint32 Sum;

    index       = 0;            //initialize index to 0
```

```
    SampleSize  = 256;          //set sample size to 256 (**NOTE: Sample
size must be multiples of 2^x where is an integer >= 4)
    Sum         = 0;            //set sum to 0
    Mean        = 999;          //initialize mean to known value

    //Set the ADC sample window to the desired value (Sample window =
ACQPS + 1)
    ACQPS_Value = 6;
    AdcRegs.ADCSOC0CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC1CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC2CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC3CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC4CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC5CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC6CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC7CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC8CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC9CTL.bit.ACQPS  = ACQPS_Value;
    AdcRegs.ADCSOC10CTL.bit.ACQPS = ACQPS_Value;
    AdcRegs.ADCSOC11CTL.bit.ACQPS = ACQPS_Value;
    AdcRegs.ADCSOC12CTL.bit.ACQPS = ACQPS_Value;
    AdcRegs.ADCSOC13CTL.bit.ACQPS = ACQPS_Value;
    AdcRegs.ADCSOC14CTL.bit.ACQPS = ACQPS_Value;
    AdcRegs.ADCSOC15CTL.bit.ACQPS = ACQPS_Value;


    //Enable ping-pong sampling

    // Enabled ADCINT1 and ADCINT2
    AdcRegs.INTSEL1N2.bit.INT1E = 1;
    AdcRegs.INTSEL1N2.bit.INT2E = 1;

    // Disable continuous sampling for ADCINT1 and ADCINT2
    AdcRegs.INTSEL1N2.bit.INT1CONT = 0;
    AdcRegs.INTSEL1N2.bit.INT2CONT = 0;

    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;    //ADCINTs trigger at end of
conversion

    // Setup ADCINT1 and ADCINT2 trigger source
    AdcRegs.INTSEL1N2.bit.INT1SEL = 6;      //EOC6 triggers ADCINT1
    AdcRegs.INTSEL1N2.bit.INT2SEL = 14;     //EOC14 triggers ADCINT2

    // Setup each SOC's ADCINT trigger source
    AdcRegs.ADCINTSOCSEL1.bit.SOC0  = 2;    //ADCINT2 starts SOC0-7
    AdcRegs.ADCINTSOCSEL1.bit.SOC1  = 2;
    AdcRegs.ADCINTSOCSEL1.bit.SOC2  = 2;
    AdcRegs.ADCINTSOCSEL1.bit.SOC3  = 2;
    AdcRegs.ADCINTSOCSEL1.bit.SOC4  = 2;
    AdcRegs.ADCINTSOCSEL1.bit.SOC5  = 2;
    AdcRegs.ADCINTSOCSEL1.bit.SOC6  = 2;
    AdcRegs.ADCINTSOCSEL1.bit.SOC7  = 2;
    AdcRegs.ADCINTSOCSEL2.bit.SOC8  = 1;    //ADCINT1 starts SOC8-15
    AdcRegs.ADCINTSOCSEL2.bit.SOC9  = 1;
```

```
    AdcRegs.ADCINTSOCSEL2.bit.SOC10 = 1;
    AdcRegs.ADCINTSOCSEL2.bit.SOC11 = 1;
    AdcRegs.ADCINTSOCSEL2.bit.SOC12 = 1;
    AdcRegs.ADCINTSOCSEL2.bit.SOC13 = 1;
    AdcRegs.ADCINTSOCSEL2.bit.SOC14 = 1;
    AdcRegs.ADCINTSOCSEL2.bit.SOC15 = 1;

    DELAY_US(ADC_usDELAY);                  // Delay before converting
ADC channels


    //ADC Conversion

    AdcRegs.ADCSOCFRC1.all = 0x00FF;  // Force Start SOC0-7 to begin
ping-pong sampling

    while( index < SampleSize ){

        //Wait for ADCINT1 to trigger, then add ADCRESULT0-7 registers to
sum
        while (AdcRegs.ADCINTFLG.bit.ADCINT1 == 0){}
        AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;   //Must clear ADCINT1 flag
since INT1CONT = 0
        Sum += AdcResult.ADCRESULT0;
        Sum += AdcResult.ADCRESULT1;
        Sum += AdcResult.ADCRESULT2;
        Sum += AdcResult.ADCRESULT3;
        Sum += AdcResult.ADCRESULT4;
        Sum += AdcResult.ADCRESULT5;
        Sum += AdcResult.ADCRESULT6;
        Sum += AdcResult.ADCRESULT7;

        //Wait for ADCINT2 to trigger, then add ADCRESULT8-15 registers
to sum
        while (AdcRegs.ADCINTFLG.bit.ADCINT2 == 0){}
        AdcRegs.ADCINTFLGCLR.bit.ADCINT2 = 1;   //Must clear ADCINT2 flag
since INT2CONT = 0
        Sum += AdcResult.ADCRESULT8;
        Sum += AdcResult.ADCRESULT9;
        Sum += AdcResult.ADCRESULT10;
        Sum += AdcResult.ADCRESULT11;
        Sum += AdcResult.ADCRESULT12;
        Sum += AdcResult.ADCRESULT13;
        Sum += AdcResult.ADCRESULT14;
        Sum += AdcResult.ADCRESULT15;

        index+=16;

    } // end data collection

    //Disable ADCINT1 and ADCINT2 to STOP the ping-pong sampling
    AdcRegs.INTSEL1N2.bit.INT1E = 0;
    AdcRegs.INTSEL1N2.bit.INT2E = 0;
```

```
    Mean = Sum / SampleSize;    //Calculate average ADC sample value

    return Mean;                //return the average

}//end AdcConversion

//===========================================================================
====
// End of file.
//===========================================================================
====


;// TI File $Revision: /main/2 $
;// Checkin $Date: January 4, 2011   10:10:05 $
;//###########################################################################
#####
;//
;// FILE:  F2806x_CodeStartBranch.asm
;//
;// TITLE: Branch for redirecting code execution after boot.
;//
;// For these examples, code_start is the first code that is executed
after
;// exiting the boot ROM code.
;//
;// The codestart section in the linker cmd file is used to physically
place
;// this code at the correct memory location.  This section should be
placed
;// at the location the BOOT ROM will re-direct the code to.  For
example,
;// for boot to FLASH this code will be located at 0x3f7ff6.
;//
;// In addition, the example F2806x projects are setup such that the
codegen
;// entry point is also set to the code_start label.  This is done by
linker
;// option -e in the project build options.  When the debugger loads the
code,
;// it will automatically set the PC to the "entry point" address
indicated by
;// the -e linker option.  In this case the debugger is simply assigning
the PC,
;// it is not the same as a full reset of the device.
;//
;// The compiler may warn that the entry point for the project is other
then
;//  _c_init00.  _c_init00 is the C environment setup and is run before
;// main() is entered. The code_start code will re-direct the execution
;// to _c_init00 and thus there is no worry and this warning can be
ignored.
;//
```

```
;//###########################################################################
#####
;// $TI Release: 2806x C/C++ Header Files V1.10 $
;// $Release Date: April 7, 2011 $
;//###########################################################################
#####

*****************************************************************************

WD_DISABLE  .set  1           ;set to 1 to disable WD, else set to 0

    .ref _c_int00
    .global code_start

*****************************************************************************
* Function: codestart section
*
* Description: Branch to code starting point
*****************************************************************************

    .sect "codestart"

code_start:
    .if WD_DISABLE == 1
        LB wd_disable       ;Branch to watchdog disable code
    .else
        LB _c_int00         ;Branch to start of boot.asm in RTS library
    .endif

;end codestart section

*****************************************************************************
* Function: wd_disable
*
* Description: Disables the watchdog timer
*****************************************************************************
    .if WD_DISABLE == 1

    .text
wd_disable:
    SETC OBJMODE        ;Set OBJMODE for 28x object code
    EALLOW              ;Enable EALLOW protected register access
    MOVZ DP, #7029h>>6  ;Set data page for WDCR register
    MOV @7029h, #0068h  ;Set WDDIS bit in WDCR to disable WD
    EDIS                ;Disable EALLOW protected register access
    LB _c_int00         ;Branch to start of boot.asm in RTS library

    .endif

;end wd_disable

      .end

;//===========================================================================
```

```
=====
;// End of file.
;//=========================================================================
=====


// TI File $Revision: /main/3 $
// Checkin $Date: February 22, 2011   17:19:34 $
//###########################################################################
####
//
// FILE:    F2806x_DefaultIsr.c
//
// TITLE:   F2806x Device Default Interrupt Service Routines.
//
// This file contains shell ISR routines for the 2803x PIE vector table.
// Typically these shell ISR routines can be used to populate the entire
PIE
// vector table during device debug.  In this manner if an interrupt is
taken
// during firmware development, there will always be an ISR to catch it.
//
// As develpment progresses, these ISR rotuines can be eliminated and
replaced
// with the user's own ISR routines for each interrupt.  Since these
shell ISRs
// include infinite loops they will typically not be included as-is in
the final
// production firmware.
//
//###########################################################################
####
// $TI Release: 2806x C/C++ Header Files V1.10 $
// $Release Date: April 7, 2011 $
//###########################################################################
####

#include "F2806x_Device.h"     // F2806x Headerfile Include File
#include "F2806x_Examples.h"   // F2806x Examples Include File

// Connected to INT13 of CPU (use MINT13 mask):
// ISR can be used by the user.
interrupt void INT13_ISR(void)     // INT13 or CPU-Timer1
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void INT14_ISR(void)     // INT14 or CPU-Timer2
```

```
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void DATALOG_ISR(void)    // Datalogging interrupt
{
   // Insert ISR Code here

   // Next two lines for debug only to halt the processor here
   // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

interrupt void RTOSINT_ISR(void)    // RTOS interrupt
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void EMUINT_ISR(void)     // Emulation interrupt
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void NMI_ISR(void)        // Non-maskable interrupt
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void ILLEGAL_ISR(void)    // Illegal operation TRAP
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
```

```
  // Remove after inserting ISR Code
  asm("          ESTOP0");
  for(;;);

}

interrupt void USER1_ISR(void)     // User Defined trap 1
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

interrupt void USER2_ISR(void)     // User Defined trap 2
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

interrupt void USER3_ISR(void)     // User Defined trap 3
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER4_ISR(void)     // User Defined trap 4
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER5_ISR(void)     // User Defined trap 5
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
```

```
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER6_ISR(void)     // User Defined trap 6
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER7_ISR(void)     // User Defined trap 7
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER8_ISR(void)     // User Defined trap 8
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER9_ISR(void)     // User Defined trap 9
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER10_ISR(void)    // User Defined trap 10
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}
```

```
interrupt void USER11_ISR(void)     // User Defined trap 11
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER12_ISR(void)     // User Defined trap 12
{
 // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// ---------------------------------------------------------
// PIE Group 1 - MUXed into CPU INT1
// ---------------------------------------------------------
// INT1.1
interrupt void ADCINT1_ISR(void)    // ADC  (Can also be ISR for INT10.1
when enabled)
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code

  asm ("      ESTOP0");
  for(;;);

}

// INT1.2
interrupt void ADCINT2_ISR(void)  // ADC  (Can also be ISR for INT10.2
when enabled)
{

  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
```

```
  // Remove after inserting ISR Code

  asm("       ESTOP0");
  for(;;);

}

// INT1.3 - Reserved

// INT1.4
interrupt void  XINT1_ISR(void)
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT1.5
interrupt void  XINT2_ISR(void)
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT1.6
interrupt void  ADCINT9_ISR(void)
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
```

```
}

// INT1.7
interrupt void  TINT0_ISR(void)      // CPU-Timer 0
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT1.8
interrupt void  WAKEINT_ISR(void)    // WD, LOW Power
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// ---------------------------------------------------------
// PIE Group 2 - MUXed into CPU INT2
// ---------------------------------------------------------

// INT2.1
interrupt void EPWM1_TZINT_ISR(void)    // EPWM Trip Zone-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.2
interrupt void EPWM2_TZINT_ISR(void)    // EPWM Trip Zone-2
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.3
interrupt void EPWM3_TZINT_ISR(void)    // EPWM Trip Zone-3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.4
interrupt void EPWM4_TZINT_ISR(void)    // EPWM Trip Zone-4
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.5
interrupt void EPWM5_TZINT_ISR(void)    // EPWM Trip Zone-5
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
```

```
  asm ("      ESTOP0");
  for(;;);
}

// INT2.6
interrupt void EPWM6_TZINT_ISR(void)    // EPWM Trip Zone-6
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.7
interrupt void EPWM7_TZINT_ISR(void)    // EPWM Trip Zone-7
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.8
interrupt void EPWM8_TZINT_ISR(void)    // EPWM Trip Zone-8
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// ---------------------------------------------------------
// PIE Group 3 - MUXed into CPU INT3
// ---------------------------------------------------------

// INT 3.1
```

```
interrupt void EPWM1_INT_ISR(void)     // EPWM-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.2
interrupt void EPWM2_INT_ISR(void)     // EPWM-2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.3
interrupt void EPWM3_INT_ISR(void)    // EPWM-3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.4
interrupt void EPWM4_INT_ISR(void)    // EPWM-4
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
```

```
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.5
interrupt void EPWM5_INT_ISR(void)     // EPWM-5
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.6
interrupt void EPWM6_INT_ISR(void)     // EPWM-6
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.7
interrupt void EPWM7_INT_ISR(void)     // EPWM-7
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.8
interrupt void EPWM8_INT_ISR(void)     // EPWM-8
{
  // Insert ISR Code here
```

```
  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// -----------------------------------------------------------
// PIE Group 4 - MUXed into CPU INT4
// -----------------------------------------------------------

// INT 4.1
interrupt void ECAP1_INT_ISR(void)    // ECAP-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT4.2
interrupt void ECAP2_INT_ISR(void)    // ECAP-2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT4.3
interrupt void ECAP3_INT_ISR(void)    // ECAP-3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
```

```
  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT4.4 - Reserved
// INT4.5 - Reserved
// INT4.6 - Reserved

// INT4.7
interrupt void HRCAP1_INT_ISR(void)     // HRCAP-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT4.8
interrupt void HRCAP2_INT_ISR(void)     // HRCAP-2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// ---------------------------------------------------------
// PIE Group 5 - MUXed into CPU INT5
// ---------------------------------------------------------

// INT 5.1
interrupt void EQEP1_INT_ISR(void)     // EQEP-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

  // Next two lines for debug only to halt the processor here
```

```
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT5.2
interrupt void EQEP2_INT_ISR(void)     // EQEP-2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT5.3 - Reserved

// INT5.4
interrupt void HRCAP3_INT_ISR(void)     // HRCAP-3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT5.5
interrupt void HRCAP4_INT_ISR(void)     // HRCAP-4
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT5.6 - Reserved
// INT5.7 - Reserved
```

```
// INT5.8
interrupt void USB0_INT_ISR(void)     // USB-0
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// ---------------------------------------------------------
// PIE Group 6 - MUXed into CPU INT6
// ---------------------------------------------------------

// INT6.1
interrupt void SPIRXINTA_ISR(void)     // SPI-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT6.2
interrupt void SPITXINTA_ISR(void)     // SPI-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT6.3
interrupt void SPIRXINTB_ISR(void)     // SPI-B
{
  // Insert ISR Code here
```

```
  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT6.4
interrupt void SPITXINTB_ISR(void)     // SPI-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT6.5
interrupt void MRINTA_ISR(void)     // McBSP-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT6.6
interrupt void MXINTA_ISR(void)     // McBSP-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
```

```
}

// INT6.7 - Reserved
// INT6.8 - Reserved

// ---------------------------------------------------------
// PIE Group 7 - MUXed into CPU INT7
// ---------------------------------------------------------

// INT7.1
interrupt void DINTCH1_ISR(void)     // DMA Channel 1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT7.2
interrupt void DINTCH2_ISR(void)     // DMA Channel 2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT7.3
interrupt void DINTCH3_ISR(void)     // DMA Channel 3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}
```

```
// INT7.4
interrupt void DINTCH4_ISR(void)     // DMA Channel 4
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT7.5
interrupt void DINTCH5_ISR(void)     // DMA Channel 5
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT7.6
interrupt void DINTCH6_ISR(void)     // DMA Channel 6
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT7.7 - Reserved
// INT7.8 - Reserved

// ---------------------------------------------------------
// PIE Group 8 - MUXed into CPU INT8
// ---------------------------------------------------------

// INT8.1
interrupt void I2CINT1A_ISR(void)     // I2C-A
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT8.2
interrupt void I2CINT2A_ISR(void)     // I2C-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT8.3 - Reserved
// INT8.4 - Reserved
// INT8.5 - Reserved
// INT8.6 - Reserved
// INT8.7 - Reserved
// INT8.8 - Reserved

// ---------------------------------------------------------
// PIE Group 9 - MUXed into CPU INT9
// ---------------------------------------------------------

// INT9.1
interrupt void SCIRXINTA_ISR(void)     // SCI-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}
```

```
// INT9.2
interrupt void SCITXINTA_ISR(void)     // SCI-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.3
interrupt void SCIRXINTB_ISR(void)     // SCI-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.4
interrupt void SCITXINTB_ISR(void)     // SCI-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.5
interrupt void ECAN0INTA_ISR(void)     // ECAN-A
{
  // Insert ISR Code here
```

```
  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.6
interrupt void ECAN1INTA_ISR(void)     // ECAN-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.7 - Reserved
// INT9.8 - Reserved

// ---------------------------------------------------------
// PIE Group 10 - MUXed into CPU INT10
// ---------------------------------------------------------

// INT10.1 - Reserved or ADCINT1_ISR
// INT10.2 - Reserved or ADCINT2_ISR

// INT10.3
interrupt void ADCINT3_ISR(void)    // ADC
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT10.4
interrupt void ADCINT4_ISR(void)    // ADC
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT10.5
interrupt void ADCINT5_ISR(void)     // ADC
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT10.6
interrupt void ADCINT6_ISR(void)     // ADC
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT10.7
interrupt void ADCINT7_ISR(void)     // ADC
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
```

```
  asm ("      ESTOP0");
  for(;;);
}

// INT10.8
interrupt void ADCINT8_ISR(void)     // ADC
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}


// -----------------------------------------------------------
// PIE Group 11 - MUXed into CPU INT11
// -----------------------------------------------------------

// INT11.1
interrupt void CLA1_INT1_ISR(void)   // MCLA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code

  asm ("      ESTOP0");
  for(;;);

}

// INT11.2
interrupt void CLA1_INT2_ISR(void)  // MCLA
{

  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
```

```
  asm("       ESTOP0");
  for(;;);

}

// INT11.3
interrupt void CLA1_INT3_ISR(void)     // MCLA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT11.4
interrupt void CLA1_INT4_ISR(void)     // MCLA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT11.5
interrupt void CLA1_INT5_ISR(void)     // MCLA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT11.6
interrupt void CLA1_INT6_ISR(void)     // MCLA
{
  // Insert ISR Code here
```

```
  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT11.7
interrupt void CLA1_INT7_ISR(void)     // MCLA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT11.8
interrupt void CLA1_INT8_ISR(void)     // MCLA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP11;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// ---------------------------------------------------------
// PIE Group 12 - MUXed into CPU INT12
// ---------------------------------------------------------

// INT12.1
interrupt void XINT3_ISR(void)  // External interrupt 3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
```

```
  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT12.2 - Reserved
// INT12.3 - Reserved
// INT12.4 - Reserved
// INT12.5 - Reserved
// INT12.6 - Reserved

// INT12.7
interrupt void LVF_ISR(void)  // Latched overflow
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT12.8
interrupt void LUF_ISR(void)  // Latched underflow
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this
interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

//---------------------------------------------------------------------------
----
// Catch All Default ISRs:
//

interrupt void EMPTY_ISR(void)  // Empty ISR - only does a return.
{

}
```

```
interrupt void PIE_RESERVED(void)  // Reserved space.  For test.
{
  asm ("      ESTOP0");
  for(;;);
}

interrupt void rsvd_ISR(void)      // For test
{
  asm ("      ESTOP0");
  for(;;);
}

//===========================================================================
====
// End of file.
//===========================================================================
====


// TI File $Revision: /main/1 $
// Checkin $Date: March 6, 2011   10:26:52 $
//###########################################################################
####
//
// FILE:    F2806x_GlobalVariableDefs.c
//
// TITLE:   F2806x Global Variables and Data Section Pragmas.
//
//###########################################################################
####
// $TI Release: 2806x C/C++ Header Files V1.10 $
// $Release Date: April 7, 2011 $
//###########################################################################
####

#include "F2806x_Device.h"     // F2806x Headerfile Include File

//---------------------------------------------------------------------------
----
// Define Global Peripheral Variables:
//
//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("AdcRegsFile")
#else
#pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
#endif
volatile struct ADC_REGS AdcRegs;

//---------------------------------------
#ifdef __cplusplus
```

```
#pragma DATA_SECTION("AdcResultFile")
#else
#pragma DATA_SECTION(AdcResult,"AdcResultFile");
#endif
volatile struct ADC_RESULT_REGS AdcResult;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("Cla1RegsFile")
#else
#pragma DATA_SECTION(Cla1Regs,"Cla1RegsFile");
#endif
volatile struct CLA_REGS Cla1Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("Comp1RegsFile")
#else
#pragma DATA_SECTION(Comp1Regs,"Comp1RegsFile");
#endif
volatile struct COMP_REGS Comp1Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("Comp2RegsFile")
#else
#pragma DATA_SECTION(Comp2Regs,"Comp2RegsFile");
#endif
volatile struct COMP_REGS Comp2Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("Comp3RegsFile")
#else
#pragma DATA_SECTION(Comp3Regs,"Comp3RegsFile");
#endif
volatile struct COMP_REGS Comp3Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CpuTimer0RegsFile")
#else
#pragma DATA_SECTION(CpuTimer0Regs,"CpuTimer0RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer0Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CpuTimer1RegsFile")
#else
#pragma DATA_SECTION(CpuTimer1Regs,"CpuTimer1RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer1Regs;
```

```
//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CpuTimer2RegsFile")
#else
#pragma DATA_SECTION(CpuTimer2Regs,"CpuTimer2RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer2Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CsmPwlFile")
#else
#pragma DATA_SECTION(CsmPwl,"CsmPwlFile");
#endif
volatile struct CSM_PWL CsmPwl;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CsmRegsFile")
#else
#pragma DATA_SECTION(CsmRegs,"CsmRegsFile");
#endif
volatile struct CSM_REGS CsmRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("DevEmuRegsFile")
#else
#pragma DATA_SECTION(DevEmuRegs,"DevEmuRegsFile");
#endif
volatile struct DEV_EMU_REGS DevEmuRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("DmaRegsFile")
#else
#pragma DATA_SECTION(DmaRegs,"DmaRegsFile");
#endif
volatile struct DMA_REGS DmaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaRegsFile")
#else
#pragma DATA_SECTION(ECanaRegs,"ECanaRegsFile");
#endif
volatile struct ECAN_REGS ECanaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMboxesFile")
#else
#pragma DATA_SECTION(ECanaMboxes,"ECanaMboxesFile");
#endif
```

```
volatile struct ECAN_MBOXES ECanaMboxes;

//-----------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaLAMRegsFile")
#else
#pragma DATA_SECTION(ECanaLAMRegs,"ECanaLAMRegsFile");
#endif
volatile struct LAM_REGS ECanaLAMRegs;

//-----------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMOTSRegsFile")
#else
#pragma DATA_SECTION(ECanaMOTSRegs,"ECanaMOTSRegsFile");
#endif
volatile struct MOTS_REGS ECanaMOTSRegs;

//-----------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMOTORegsFile")
#else
#pragma DATA_SECTION(ECanaMOTORegs,"ECanaMOTORegsFile");
#endif
volatile struct MOTO_REGS ECanaMOTORegs;

//-----------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm1RegsFile")
#else
#pragma DATA_SECTION(EPwm1Regs,"EPwm1RegsFile");
#endif
volatile struct EPWM_REGS EPwm1Regs;

//-----------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm2RegsFile")
#else
#pragma DATA_SECTION(EPwm2Regs,"EPwm2RegsFile");
#endif
volatile struct EPWM_REGS EPwm2Regs;

//-----------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm3RegsFile")
#else
#pragma DATA_SECTION(EPwm3Regs,"EPwm3RegsFile");
#endif
volatile struct EPWM_REGS EPwm3Regs;

//-----------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm4RegsFile")
#else
```

```
#pragma DATA_SECTION(EPwm4Regs,"EPwm4RegsFile");
#endif
volatile struct EPWM_REGS EPwm4Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm5RegsFile")
#else
#pragma DATA_SECTION(EPwm5Regs,"EPwm5RegsFile");
#endif
volatile struct EPWM_REGS EPwm5Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm6RegsFile")
#else
#pragma DATA_SECTION(EPwm6Regs,"EPwm6RegsFile");
#endif
volatile struct EPWM_REGS EPwm6Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm7RegsFile")
#else
#pragma DATA_SECTION(EPwm7Regs,"EPwm7RegsFile");
#endif
volatile struct EPWM_REGS EPwm7Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm8RegsFile")
#else
#pragma DATA_SECTION(EPwm8Regs,"EPwm8RegsFile");
#endif
volatile struct EPWM_REGS EPwm8Regs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap1RegsFile")
#else
#pragma DATA_SECTION(ECap1Regs,"ECap1RegsFile");
#endif
volatile struct ECAP_REGS ECap1Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap2RegsFile")
#else
#pragma DATA_SECTION(ECap2Regs,"ECap2RegsFile");
#endif
volatile struct ECAP_REGS ECap2Regs;

//---------------------------------------
```

```
#ifdef __cplusplus
#pragma DATA_SECTION("ECap3RegsFile")
#else
#pragma DATA_SECTION(ECap3Regs,"ECap3RegsFile");
#endif
volatile struct ECAP_REGS ECap3Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EQep1RegsFile")
#else
#pragma DATA_SECTION(EQep1Regs,"EQep1RegsFile");
#endif
volatile struct EQEP_REGS EQep1Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EQep2RegsFile")
#else
#pragma DATA_SECTION(EQep2Regs,"EQep2RegsFile");
#endif
volatile struct EQEP_REGS EQep2Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("FlashRegsFile")
#else
#pragma DATA_SECTION(FlashRegs,"FlashRegsFile");
#endif
volatile struct FLASH_REGS FlashRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioCtrlRegsFile")
#else
#pragma DATA_SECTION(GpioCtrlRegs,"GpioCtrlRegsFile");
#endif
volatile struct GPIO_CTRL_REGS GpioCtrlRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioDataRegsFile")
#else
#pragma DATA_SECTION(GpioDataRegs,"GpioDataRegsFile");
#endif
volatile struct GPIO_DATA_REGS GpioDataRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioIntRegsFile")
#else
#pragma DATA_SECTION(GpioIntRegs,"GpioIntRegsFile");
#endif
volatile struct GPIO_INT_REGS GpioIntRegs;
```

```
//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("HRCap1RegsFile")
#else
#pragma DATA_SECTION(HRCap1Regs,"HRCap1RegsFile");
#endif
volatile struct HRCAP_REGS HRCap1Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("HRCap2RegsFile")
#else
#pragma DATA_SECTION(HRCap2Regs,"HRCap2RegsFile");
#endif
volatile struct HRCAP_REGS HRCap2Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("HRCap3RegsFile")
#else
#pragma DATA_SECTION(HRCap3Regs,"HRCap3RegsFile");
#endif
volatile struct HRCAP_REGS HRCap3Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("HRCap4RegsFile")
#else
#pragma DATA_SECTION(HRCap4Regs,"HRCap4RegsFile");
#endif
volatile struct HRCAP_REGS HRCap4Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("I2caRegsFile")
#else
#pragma DATA_SECTION(I2caRegs,"I2caRegsFile");
#endif
volatile struct I2C_REGS I2caRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("McbspaRegsFile")
#else
#pragma DATA_SECTION(McbspaRegs,"McbspaRegsFile");
#endif
volatile struct McBSP_REGS McbspaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("NmiIntruptRegsFile")
#else
#pragma DATA_SECTION(NmiIntruptRegs,"NmiIntruptRegsFile");
```

```
#endif
volatile struct NMIINTRUPT_REGS NmiIntruptRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PartIdRegsFile")
#else
#pragma DATA_SECTION(PartIdRegs,"PartIdRegsFile");
#endif
volatile struct PARTID_REGS PartIdRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PieCtrlRegsFile")
#else
#pragma DATA_SECTION(PieCtrlRegs,"PieCtrlRegsFile");
#endif
volatile struct PIE_CTRL_REGS PieCtrlRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PieVectTableFile")
#else
#pragma DATA_SECTION(PieVectTable,"PieVectTableFile");
#endif
struct PIE_VECT_TABLE PieVectTable;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SciaRegsFile")
#else
#pragma DATA_SECTION(SciaRegs,"SciaRegsFile");
#endif
volatile struct SCI_REGS SciaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ScibRegsFile")
#else
#pragma DATA_SECTION(ScibRegs,"ScibRegsFile");
#endif
volatile struct SCI_REGS ScibRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SpiaRegsFile")
#else
#pragma DATA_SECTION(SpiaRegs,"SpiaRegsFile");
#endif
volatile struct SPI_REGS SpiaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SpibRegsFile")
```

```
#else
#pragma DATA_SECTION(SpibRegs,"SpibRegsFile");
#endif
volatile struct SPI_REGS SpibRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SysCtrlRegsFile")
#else
#pragma DATA_SECTION(SysCtrlRegs,"SysCtrlRegsFile");
#endif
volatile struct SYS_CTRL_REGS SysCtrlRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SysPwrCtrlRegsFile")
#else
#pragma DATA_SECTION(SysPwrCtrlRegs,"SysPwrCtrlRegsFile");
#endif
volatile struct SYS_PWR_CTRL_REGS SysPwrCtrlRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("Usb0RegsFile")
#else
#pragma DATA_SECTION(Usb0Regs,"Usb0RegsFile");
#endif
volatile struct USB_REGS Usb0Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("XIntruptRegsFile")
#else
#pragma DATA_SECTION(XIntruptRegs,"XIntruptRegsFile");
#endif
volatile struct XINTRUPT_REGS XIntruptRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EmuKeyVar");
#else
#pragma DATA_SECTION(EmuKey,"EmuKeyVar");
#endif
Uint16 EmuKey;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EmuBModeVar");
#else
#pragma DATA_SECTION(EmuBMode,"EmuBModeVar");
#endif
Uint16 EmuBMode;

//---------------------------------------
```

```
#ifdef __cplusplus
#pragma DATA_SECTION("FlashScalingVar");
#else
#pragma DATA_SECTION(Flash_CPUScaleFactor, "FlashScalingVar");
#endif
Uint32 Flash_CPUScaleFactor;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("FlashCallbackVar");
#else
#pragma DATA_SECTION(Flash_CallbackPtr, "FlashCallbackVar");
#endif
void (*Flash_CallbackPtr) (void);


//===========================================================================
====
// End of file.
//===========================================================================
====

// TI File $Revision: /main/2 $
// Checkin $Date: January 4, 2011   10:10:35 $
//###########################################################################
####
//
// FILE:    F2806x_PieCtrl.c
//
// TITLE:   F2806x Device PIE Control Register Initialization Functions.
//
//###########################################################################
####
// $TI Release: 2806x C/C++ Header Files V1.10 $
// $Release Date: April 7, 2011 $
//###########################################################################
####

#include "F2806x_Device.h"     // F2806x Headerfile Include File
#include "F2806x_Examples.h"   // F2806x Examples Include File

//---------------------------------------------------------------------------
----
// InitPieCtrl:
//---------------------------------------------------------------------------
----
// This function initializes the PIE control registers to a known state.
//
void InitPieCtrl(void)
{
    // Disable Interrupts at the CPU level:
    DINT;
```

```
    // Disable the PIE
    PieCtrlRegs.PIECTRL.bit.ENPIE = 0;

      // Clear all PIEIER registers:
      PieCtrlRegs.PIEIER1.all = 0;
      PieCtrlRegs.PIEIER2.all = 0;
      PieCtrlRegs.PIEIER3.all = 0;
      PieCtrlRegs.PIEIER4.all = 0;
      PieCtrlRegs.PIEIER5.all = 0;
      PieCtrlRegs.PIEIER6.all = 0;
      PieCtrlRegs.PIEIER7.all = 0;
      PieCtrlRegs.PIEIER8.all = 0;
      PieCtrlRegs.PIEIER9.all = 0;
      PieCtrlRegs.PIEIER10.all = 0;
      PieCtrlRegs.PIEIER11.all = 0;
      PieCtrlRegs.PIEIER12.all = 0;

      // Clear all PIEIFR registers:
      PieCtrlRegs.PIEIFR1.all = 0;
      PieCtrlRegs.PIEIFR2.all = 0;
      PieCtrlRegs.PIEIFR3.all = 0;
      PieCtrlRegs.PIEIFR4.all = 0;
      PieCtrlRegs.PIEIFR5.all = 0;
      PieCtrlRegs.PIEIFR6.all = 0;
      PieCtrlRegs.PIEIFR7.all = 0;
      PieCtrlRegs.PIEIFR8.all = 0;
      PieCtrlRegs.PIEIFR9.all = 0;
      PieCtrlRegs.PIEIFR10.all = 0;
      PieCtrlRegs.PIEIFR11.all = 0;
      PieCtrlRegs.PIEIFR12.all = 0;

}

//---------------------------------------------------------------------------
----
// EnableInterrupts:
//---------------------------------------------------------------------------
----
// This function enables the PIE module and CPU interrupts
//
void EnableInterrupts()
{

    // Enable the PIE
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1;

      // Enables PIE to drive a pulse into the CPU
      PieCtrlRegs.PIEACK.all = 0xFFFF;

      // Enable Interrupts at the CPU level
    EINT;

}
```

```
//===========================================================================
====
// End of file.
//===========================================================================
====


// TI File $Revision: /main/3 $
// Checkin $Date: February 22, 2011   17:21:22 $
//###########################################################################
####
//
// FILE:    F2806x_PieVect.c
//
// TITLE:   F2806x Devices PIE Vector Table Initialization Functions.
//
//###########################################################################
####
// $TI Release: 2806x C/C++ Header Files V1.10 $
// $Release Date: April 7, 2011 $
//###########################################################################
####

#include "F2806x_Device.h"     // F2806x Headerfile Include File
#include "F2806x_Examples.h"   // F2806x Examples Include File

const struct PIE_VECT_TABLE PieVectTableInit = {

      PIE_RESERVED,  // 1  Reserved space
      PIE_RESERVED,  // 2  Reserved space
      PIE_RESERVED,  // 3  Reserved space
      PIE_RESERVED,  // 4  Reserved space
      PIE_RESERVED,  // 5  Reserved space
      PIE_RESERVED,  // 6  Reserved space
      PIE_RESERVED,  // 7  Reserved space
      PIE_RESERVED,  // 8  Reserved space
      PIE_RESERVED,  // 9  Reserved space
      PIE_RESERVED,  // 10 Reserved space
      PIE_RESERVED,  // 11 Reserved space
      PIE_RESERVED,  // 12 Reserved space
      PIE_RESERVED,  // 13 Reserved space

// Non-Peripheral Interrupts
      INT13_ISR,     // CPU-Timer 1
      INT14_ISR,     // CPU-Timer 2
      DATALOG_ISR,   // Datalogging interrupt
      RTOSINT_ISR,   // RTOS interrupt
      EMUINT_ISR,    // Emulation interrupt
      NMI_ISR,       // Non-maskable interrupt
      ILLEGAL_ISR,   // Illegal operation TRAP
      USER1_ISR,     // User Defined trap 1
      USER2_ISR,     // User Defined trap 2
      USER3_ISR,     // User Defined trap 3
```

```
      USER4_ISR,     // User Defined trap 4
      USER5_ISR,     // User Defined trap 5
      USER6_ISR,     // User Defined trap 6
      USER7_ISR,     // User Defined trap 7
      USER8_ISR,     // User Defined trap 8
      USER9_ISR,     // User Defined trap 9
      USER10_ISR,    // User Defined trap 10
      USER11_ISR,    // User Defined trap 11
      USER12_ISR,    // User Defined trap 12

// Group 1 PIE Vectors
      ADCINT1_ISR,     // 1.1 ADC  ADC - make rsvd1_1 if ADCINT1 is
wanted in Group 10 instead.
      ADCINT2_ISR,     // 1.2 ADC  ADC - make rsvd1_2 if ADCINT2 is
wanted in Group 10 instead.
      rsvd_ISR,        // 1.3
      XINT1_ISR,       // 1.4 External Interrupt
      XINT2_ISR,       // 1.5 External Interrupt
      ADCINT9_ISR,     // 1.6 ADC Interrupt 9
      TINT0_ISR,       // 1.7 Timer 0
      WAKEINT_ISR,     // 1.8 WD, Low Power

// Group 2 PIE Vectors
      EPWM1_TZINT_ISR, // 2.1 EPWM-1 Trip Zone
      EPWM2_TZINT_ISR, // 2.2 EPWM-2 Trip Zone
      EPWM3_TZINT_ISR, // 2.3 EPWM-3 Trip Zone
      EPWM4_TZINT_ISR, // 2.4 EPWM-4 Trip Zone
      EPWM5_TZINT_ISR, // 2.5 EPWM-5 Trip Zone
      EPWM6_TZINT_ISR, // 2.6 EPWM-6 Trip Zone
      EPWM7_TZINT_ISR, // 2.7 EPWM-7 Trip Zone
      EPWM8_TZINT_ISR, // 2.8 EPWM-8 Trip Zone

// Group 3 PIE Vectors
      EPWM1_INT_ISR,   // 3.1 EPWM-1 Interrupt
      EPWM2_INT_ISR,   // 3.2 EPWM-2 Interrupt
      EPWM3_INT_ISR,   // 3.3 EPWM-3 Interrupt
      EPWM4_INT_ISR,   // 3.4 EPWM-4 Interrupt
      EPWM5_INT_ISR,   // 3.5 EPWM-5 Interrupt
      EPWM6_INT_ISR,   // 3.6 EPWM-6 Interrupt
      EPWM7_INT_ISR,   // 3.7 EPWM-7 Interrupt
      EPWM8_INT_ISR,   // 3.8 EPWM-8 Interrupt

// Group 4 PIE Vectors
      ECAP1_INT_ISR,   // 4.1 ECAP-1
      ECAP2_INT_ISR,   // 4.2 ECAP-2
      ECAP3_INT_ISR,   // 4.3 ECAP-3
      rsvd_ISR,           // 4.4
      rsvd_ISR,        // 4.5
      rsvd_ISR,        // 4.6
      HRCAP1_INT_ISR,  // 4.7 HRCAP-1
      HRCAP2_INT_ISR,  // 4.8 HRCAP-2

// Group 5 PIE Vectors
```

```
      EQEP1_INT_ISR,    // 5.1 EQEP-1
      EQEP2_INT_ISR,    // 5.2 EQEP-2
      rsvd_ISR,          // 5.3
      HRCAP3_INT_ISR,   // 5.4 HRCAP-3
      HRCAP4_INT_ISR,   // 5.5 HRCAP-4
      rsvd_ISR,         // 5.6
      rsvd_ISR,         // 5.7
      USB0_INT_ISR,     // 5.8 USB-0

// Group 6 PIE Vectors
      SPIRXINTA_ISR,    // 6.1 SPI-A
      SPITXINTA_ISR,    // 6.2 SPI-A
      SPIRXINTB_ISR,    // 6.3 SPI-B
      SPITXINTA_ISR,    // 6.4 SPI-B
      MRINTA_ISR,       // 6.5 McBSP-A
      MXINTA_ISR,       // 6.6 McBSP-A
      rsvd_ISR,         // 6.7
      rsvd_ISR,         // 6.8

// Group 7 PIE Vectors
      DINTCH1_ISR,      // 7.1 DMA Channel 1
      DINTCH2_ISR,      // 7.2 DMA Channel 2
      DINTCH3_ISR,      // 7.3 DMA Channel 3
      DINTCH4_ISR,      // 7.4 DMA Channel 4
      DINTCH5_ISR,      // 7.5 DMA Channel 5
      DINTCH6_ISR,      // 7.6 DMA Channel 6
      rsvd_ISR,         // 7.7
      rsvd_ISR,         // 7.8

// Group 8 PIE Vectors
      I2CINT1A_ISR,     // 8.1 I2C-A
      I2CINT2A_ISR,     // 8.2 I2C-A
      rsvd_ISR,         // 8.3
      rsvd_ISR,         // 8.4
      rsvd_ISR,         // 8.5
      rsvd_ISR,         // 8.6
      rsvd_ISR,         // 8.7
      rsvd_ISR,         // 8.8

// Group 9 PIE Vectors
      SCIRXINTA_ISR,    // 9.1 SCI-A
      SCITXINTA_ISR,    // 9.2 SCI-A
      SCIRXINTB_ISR,    // 9.3 SCI-B
      SCITXINTB_ISR,    // 9.4 SCI-B
      ECAN0INTA_ISR,    // 9.5 ECAN-A
      ECAN1INTA_ISR,    // 9.6 ECAN-A
      rsvd_ISR,         // 9.7
      rsvd_ISR,         // 9.8

// Group 10 PIE Vectors
      rsvd_ISR,         // 10.1 Can be ADCINT1, but must make ADCINT1 in
Group 1 space "reserved".
      rsvd_ISR,         // 10.2 Can be ADCINT2, but must make ADCINT2 in
Group 1 space "reserved".
```

```
      ADCINT3_ISR,      // 10.3 ADC
      ADCINT4_ISR,      // 10.4 ADC
      ADCINT5_ISR,      // 10.5 ADC
      ADCINT6_ISR,      // 10.6 ADC
      ADCINT7_ISR,      // 10.7 ADC
      ADCINT8_ISR,      // 10.8 ADC

// Group 11 PIE Vectors
      CLA1_INT1_ISR,    // 11.1 CLA1
        CLA1_INT2_ISR,    // 11.2 CLA1
        CLA1_INT3_ISR,    // 11.3 CLA1
        CLA1_INT4_ISR,    // 11.4 CLA1
        CLA1_INT5_ISR,    // 11.5 CLA1
        CLA1_INT6_ISR,    // 11.6 CLA1
        CLA1_INT7_ISR,    // 11.7 CLA1
      CLA1_INT8_ISR,    // 11.8 CLA1

// Group 12 PIE Vectors
      XINT3_ISR,        // 12.1 External Interrupt
      rsvd_ISR,         // 12.2
      rsvd_ISR,         // 12.3
      rsvd_ISR,         // 12.4
      rsvd_ISR,         // 12.5
      rsvd_ISR,         // 12.6
      LVF_ISR,          // 12.7 Latched Overflow
      LUF_ISR           // 12.8 Latched Underflow
};

//---------------------------------------------------------------------------
----
// InitPieVectTable:
//---------------------------------------------------------------------------
----
// This function initializes the PIE vector table to a known state.
// This function must be executed after boot time.
//

void InitPieVectTable(void)
{
      int16 i;
      Uint32 *Source = (void *) &PieVectTableInit;
      Uint32 *Dest = (void *) &PieVectTable;

      // Do not write over first 3 32-bit locations (these locations are
      // initialized by Boot ROM with boot variables)

      Source = Source + 3;
      Dest = Dest + 3;

      EALLOW;
      for(i=0; i < 125; i++)
            *Dest++ = *Source++;
      EDIS;
```

```
      // Enable the PIE Vector Table
      PieCtrlRegs.PIECTRL.bit.ENPIE = 1;

}

//===========================================================================
====
// End of file.
//===========================================================================
====


// TI File $Revision: /main/2 $
// Checkin $Date: January 4, 2011   10:10:42 $
//###########################################################################
####
//
// FILE:   F2806x_Spi.c
//
// TITLE:  F2806x SPI Initialization & Support Functions.
//
//###########################################################################
####
// $TI Release: 2806x C/C++ Header Files V1.10 $
// $Release Date: April 7, 2011 $
//###########################################################################
####

#include "F2806x_Device.h"     // F2806x Headerfile Include File
#include "F2806x_Examples.h"   // F2806x Examples Include File

//---------------------------------------------------------------------------
----
// InitSPI:
//---------------------------------------------------------------------------
----
// This function initializes the SPI(s) to a known state.
//
void InitSpi(void)
{
   // Initialize SPI-A/B

   //tbd...

}

//---------------------------------------------------------------------------
----
// Example: InitSpiGpio:
//---------------------------------------------------------------------------
----
// This function initializes GPIO pins to function as SPI pins
//
```

```
// Each GPIO pin can be configured as a GPIO pin or up to 3 different
// peripheral functional pins. By default all pins come up as GPIO
// inputs after reset.
//
// Caution:
// For each SPI peripheral
// Only one GPIO pin should be enabled for SPISIMO operation.
// Only one GPIO pin should be enabled for SPISOMI operation.
// Only one GPIO pin should be enabled for SPICLK  operation.
// Only one GPIO pin should be enabled for SPISTE  operation.
// Comment out other unwanted lines.

void InitSpiGpio()
{
   #if DSP28_SPIA
       InitSpiaGpio();
   #endif // endif DSP28_SPIA
   #if DSP28_SPIB
       InitSpibGpio();
   #endif // endif DSP28_SPIB
}

#if DSP28_SPIA
void InitSpiaGpio()
{

   EALLOW;
/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

//  GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0;    // Enable pull-up on GPIO3
(SPISOMIA)
//  GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;    // Enable pull-up on GPIO5
(SPISIMOA)

    GpioCtrlRegs.GPAPUD.bit.GPIO16 = 0;   // Enable pull-up on GPIO16
(SPISIMOA)
    GpioCtrlRegs.GPAPUD.bit.GPIO17 = 0;   // Enable pull-up on GPIO17
(SPISOMIA)
    GpioCtrlRegs.GPAPUD.bit.GPIO18 = 0;   // Enable pull-up on GPIO18
(SPICLKA)
    GpioCtrlRegs.GPAPUD.bit.GPIO19 = 0;   // Enable pull-up on GPIO19
(SPISTEA)

//  GpioCtrlRegs.GPBPUD.bit.GPIO54 = 0;   // Enable pull-up on GPIO54
(SPISIMOA)
//  GpioCtrlRegs.GPBPUD.bit.GPIO55 = 0;   // Enable pull-up on GPIO55
(SPISOMIA)
//  GpioCtrlRegs.GPBPUD.bit.GPIO56 = 0;   // Enable pull-up on GPIO56
(SPICLKA)
//  GpioCtrlRegs.GPBPUD.bit.GPIO57 = 0;   // Enable pull-up on GPIO57
(SPISTEA)
```

```
/* Set qualification for selected pins to asynch only */
// This will select asynch (no qualification) for the selected pins.
// Comment out other unwanted lines.

//    GpioCtrlRegs.GPAQSEL1.bit.GPIO3 = 3;  // Asynch input GPIO3
(SPISOMIA)
//    GpioCtrlRegs.GPAQSEL1.bit.GPIO5 = 3;  // Asynch input GPIO5
(SPISIMOA)

    GpioCtrlRegs.GPAQSEL2.bit.GPIO16 = 3; // Asynch input GPIO16
(SPISIMOA)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO17 = 3; // Asynch input GPIO17
(SPISOMIA)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO18 = 3; // Asynch input GPIO18
(SPICLKA)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO19 = 3; // Asynch input GPIO19
(SPISTEA)

//  GpioCtrlRegs.GPBQSEL2.bit.GPIO54 = 3; // Asynch input GPIO54
(SPISIMOA)
//  GpioCtrlRegs.GPBQSEL2.bit.GPIO55 = 3; // Asynch input GPIO55
(SPISOMIA)
//  GpioCtrlRegs.GPBQSEL2.bit.GPIO56 = 3; // Asynch input GPIO56
(SPICLKA)
//  GpioCtrlRegs.GPBQSEL2.bit.GPIO57 = 3; // Asynch input GPIO57
(SPISTEA)


/* Configure SPI-A pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be SPI functional
pins.
// Comment out other unwanted lines.

//  GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 2;  // Configure GPIO3 as SPISOMIA
//  GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 2;  // Configure GPIO5 as SPISIMOA

    GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 1; // Configure GPIO16 as SPISIMOA
    GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 1; // Configure GPIO17 as SPISOMIA
    GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 1; // Configure GPIO18 as SPICLKA
    GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 1; // Configure GPIO19 as SPISTEA

//  GpioCtrlRegs.GPBMUX2.bit.GPIO54 = 1; // Configure GPIO54 as SPISIMOA
//  GpioCtrlRegs.GPBMUX2.bit.GPIO55 = 1; // Configure GPIO55 as SPISOMIA
//  GpioCtrlRegs.GPBMUX2.bit.GPIO56 = 1; // Configure GPIO56 as SPICLKA
//  GpioCtrlRegs.GPBMUX2.bit.GPIO57 = 1; // Configure GPIO57 as SPISTEA

    EDIS;
}
#endif // endif DSP28_SPIA

#if DSP28_SPIB
void InitSpibGpio()
{
```

```
   EALLOW;
/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAPUD.bit.GPIO12 = 0;   // Enable pull-up on GPIO12
(SPISIMOB)
      GpioCtrlRegs.GPAPUD.bit.GPIO13 = 0;   // Enable pull-up on GPIO13
(SPISOMIB)
    GpioCtrlRegs.GPAPUD.bit.GPIO14 = 0;   // Enable pull-up on GPIO14
(SPICLKB)
    GpioCtrlRegs.GPAPUD.bit.GPIO15 = 0;   // Enable pull-up on GPIO15
(SPISTEB)

//    GpioCtrlRegs.GPAPUD.bit.GPIO24 = 0;   // Enable pull-up on GPIO24
(SPISIMOB)
//    GpioCtrlRegs.GPAPUD.bit.GPIO25 = 0;   // Enable pull-up on GPIO25
(SPISOMIB)
//  GpioCtrlRegs.GPAPUD.bit.GPIO26 = 0;   // Enable pull-up on GPIO26
(SPICLKB)
//  GpioCtrlRegs.GPAPUD.bit.GPIO27 = 0;   // Enable pull-up on GPIO27
(SPISTEB)

/* Set qualification for selected pins to asynch only */
// This will select asynch (no qualification) for the selected pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAQSEL1.bit.GPIO12 = 3; // Asynch input GPIO12
(SPISIMOB)
    GpioCtrlRegs.GPAQSEL1.bit.GPIO13 = 3; // Asynch input GPIO13
(SPISOMIB)
    GpioCtrlRegs.GPAQSEL1.bit.GPIO14 = 3; // Asynch input GPIO14
(SPICLKB)
    GpioCtrlRegs.GPAQSEL1.bit.GPIO15 = 3; // Asynch input GPIO15
(SPISTEB)

//    GpioCtrlRegs.GPAQSEL2.bit.GPIO24 = 3; // Asynch input GPIO24
(SPISIMOB)
//  GpioCtrlRegs.GPAQSEL2.bit.GPIO25 = 3; // Asynch input GPIO25
(SPISOMIB)
//  GpioCtrlRegs.GPAQSEL2.bit.GPIO26 = 3; // Asynch input GPIO26
(SPICLKB)
//  GpioCtrlRegs.GPAQSEL2.bit.GPIO27 = 3; // Asynch input GPIO27
(SPISTEB)

/* Configure SPI-B pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be SPI functional
pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 3; // Configure GPIO12 as SPISIMOB
    GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 3; // Configure GPIO13 as SPISOMIB
```

```
    GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 3; // Configure GPIO14 as SPICLKB
    GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 3; // Configure GPIO15 as SPISTEB

//  GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 3; // Configure GPIO24 as SPISIMOB
//  GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 3; // Configure GPIO25 as SPISOMIB
//  GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 3; // Configure GPIO26 as SPICLKB
//  GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 3; // Configure GPIO27 as SPISTEB

    EDIS;
}
#endif // endif DSP28_SPIB

//===========================================================================
====
// End of file.
//===========================================================================
====


// TI File $Revision: /main/1 $
// Checkin $Date: February 28, 2011   10:41:04 $
//###########################################################################
####
//
// FILE:   F2806x_SysCtrl.c
//
// TITLE:  F2806x Device System Control Initialization & Support
Functions.
//
// DESCRIPTION:
//
//         Example initialization of system resources.
//
//###########################################################################
####
// $TI Release: 2806x C/C++ Header Files V1.10 $
// $Release Date: April 7, 2011 $
//###########################################################################
####

#include "F2806x_Device.h"     // Headerfile Include File
#include "F2806x_Examples.h"   // Examples Include File

// Functions that will be run from RAM need to be assigned to
// a different section.  This section will then be mapped to a load and
// run address using the linker cmd file.

#pragma CODE_SECTION(InitFlash, "ramfuncs");

//---------------------------------------------------------------------------
----
// InitSysCtrl:
//---------------------------------------------------------------------------
```

```
----
// This function initializes the System Control registers to a known
state.
// - Disables the watchdog
// - Set the PLLCR for proper SYSCLKOUT frequency
// - Set the pre-scaler for the high and low frequency peripheral clocks
// - Enable the clocks to the peripherals

void InitSysCtrl(void)
{

   // Disable the watchdog
   DisableDog();

   // *IMPORTANT*
   // The Device_cal function, which copies the ADC & oscillator
calibration values
   // from TI reserved OTP into the appropriate trim registers, occurs
automatically
   // in the Boot ROM. If the boot ROM code is bypassed during the debug
process, the
   // following function MUST be called for the ADC and oscillators to
function according
   // to specification. The clocks to the ADC MUST be enabled before
calling this
   // function.
   // See the device data manual and/or the ADC Reference
   // Manual for more information.

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1; // Enable ADC peripheral clock
   (*Device_cal)();
   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0; // Return ADC clock to original
state
   EDIS;

   // Select Internal Oscillator 1 as Clock Source (default), and turn
off all unused clocks to
   // conserve power.
   IntOsc1Sel();

   // Initialize the PLL control: PLLCR and CLKINDIV
   // DSP28_PLLCR and DSP28_CLKINDIV are defined in F2806x_Examples.h
   InitPll(DSP28_PLLCR,DSP28_DIVSEL);

   // Initialize the peripheral clocks
   InitPeripheralClocks();
}

//---------------------------------------------------------------------------
----
// Example: InitFlash:
//---------------------------------------------------------------------------
----
```

```
// This function initializes the Flash Control registers

//                   CAUTION
// This function MUST be executed out of RAM. Executing it
// out of OTP/Flash will yield unpredictable results

void InitFlash(void)
{
   EALLOW;
   //Enable Flash Pipeline mode to improve performance
   //of code executed from Flash.
   FlashRegs.FOPT.bit.ENPIPE = 1;

   //                CAUTION
   //Minimum waitstates required for the flash operating
   //at a given CPU rate must be characterized by TI.
   //Refer to the datasheet for the latest information.

   //Set the Paged Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.PAGEWAIT = 2;

   //Set the Random Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.RANDWAIT = 2;

   //Set the Waitstate for the OTP
   FlashRegs.FOTPWAIT.bit.OTPWAIT = 2;

   //                CAUTION
   //ONLY THE DEFAULT VALUE FOR THESE 2 REGISTERS SHOULD BE USED
   FlashRegs.FSTDBYWAIT.bit.STDBYWAIT = 0x01FF;
   FlashRegs.FACTIVEWAIT.bit.ACTIVEWAIT = 0x01FF;
   EDIS;

   //Force a pipeline flush to ensure that the write to
   //the last register configured occurs before returning.

   asm(" RPT #7 || NOP");
}

//---------------------------------------------------------------------------
// Example: ServiceDog:
//---------------------------------------------------------------------------
// This function resets the watchdog timer.
// Enable this function for using ServiceDog in the application

void ServiceDog(void)
{
    EALLOW;
    SysCtrlRegs.WDKEY = 0x0055;
    SysCtrlRegs.WDKEY = 0x00AA;
    EDIS;
}
```

```
//---------------------------------------------------------------------------
// Example: DisableDog:
//---------------------------------------------------------------------------
// This function disables the watchdog timer.

void DisableDog(void)
{
    EALLOW;
    SysCtrlRegs.WDCR= 0x0068;
    EDIS;
}

//---------------------------------------------------------------------------
// Example: InitPll:
//---------------------------------------------------------------------------
// This function initializes the PLLCR register.

void InitPll(Uint16 val, Uint16 divsel)
{
   volatile Uint16 iVol;

   // Make sure the PLL is not running in limp mode
   if (SysCtrlRegs.PLLSTS.bit.MCLKSTS != 0)
   {
      EALLOW;
      // OSCCLKSRC1 failure detected. PLL running in limp mode.
      // Re-enable missing clock logic.
      SysCtrlRegs.PLLSTS.bit.MCLKCLR = 1;
      EDIS;
      // Replace this line with a call to an appropriate
      // SystemShutdown(); function.
      asm("        ESTOP0");     // Uncomment for debugging purposes
   }

   // DIVSEL MUST be 0 before PLLCR can be changed from
   // 0x0000. It is set to 0 by an external reset XRSn
   // This puts us in 1/4
   if (SysCtrlRegs.PLLSTS.bit.DIVSEL != 0)
   {
       EALLOW;
       SysCtrlRegs.PLLSTS.bit.DIVSEL = 0;
       EDIS;
   }

   // Change the PLLCR
   if (SysCtrlRegs.PLLCR.bit.DIV != val)
   {

      EALLOW;
```

```
        // Before setting PLLCR turn off missing clock detect logic
        SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;
        SysCtrlRegs.PLLCR.bit.DIV = val;
        EDIS;

        // Optional: Wait for PLL to lock.
        // During this time the CPU will switch to OSCCLK/2 until
        // the PLL is stable.  Once the PLL is stable the CPU will
        // switch to the new PLL value.
        //
        // This time-to-lock is monitored by a PLL lock counter.
        //
        // Code is not required to sit and wait for the PLL to lock.
        // However, if the code does anything that is timing critical,
        // and requires the correct clock be locked, then it is best to
        // wait until this switching has completed.

        // Wait for the PLL lock bit to be set.

        // The watchdog should be disabled before this loop, or fed within
        // the loop via ServiceDog().

        // Uncomment to disable the watchdog
        DisableDog();

        while(SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1)
        {
            // Uncomment to service the watchdog
            // ServiceDog();
        }

        EALLOW;
        SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;
        EDIS;
    }

    // If switching to 1/2
    if((divsel == 1)||(divsel == 2))
    {
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.DIVSEL = divsel;
        EDIS;
    }

    // If switching to 1/1
    // * First go to 1/2 and let the power settle
    //   The time required will depend on the system, this is only an
example
    // * Then switch to 1/1
    if(divsel == 3)
    {
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 2;
        DELAY_US(50L);
```

```
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 3;
        EDIS;
    }
}

//---------------------------------------------------------------------------
// Example: InitPll2:
//---------------------------------------------------------------------------
// This function initializes the PLL2 registers.

void InitPll2(Uint16 clksrc, Uint16 pllmult, Uint16 clkdiv)
{
      EALLOW;

      // Check if SYSCLK2DIV2DIS is in /2 mode
      if(DevEmuRegs.DEVICECNF.bit.SYSCLK2DIV2DIS != 0)
      {
            DevEmuRegs.DEVICECNF.bit.SYSCLK2DIV2DIS = 0;
      }

      // Enable PLL2
      SysCtrlRegs.PLL2CTL.bit.PLL2EN = 1;
      // Select clock source for PLL2
      SysCtrlRegs.PLL2CTL.bit.PLL2CLKSRCSEL = clksrc;
      // Set PLL2 Multiplier
      SysCtrlRegs.PLL2MULT.bit.PLL2MULT = pllmult;

      // Wait for PLL to lock.
      // Uncomment to disable the watchdog
        DisableDog();
      while(SysCtrlRegs.PLL2STS.bit.PLL2LOCKS!= 1)
      {
            // Uncomment to service the watchdog
        // ServiceDog();
      }

      // Set System Clock 2 divider
      DevEmuRegs.DEVICECNF.bit.SYSCLK2DIV2DIS = clkdiv;
      EDIS;
}

//---------------------------------------------------------------------------
// Example: InitPeripheralClocks:
//---------------------------------------------------------------------------
// This function initializes the clocks to the peripheral modules.
// First the high and low clock prescalers are set
// Second the clocks are enabled to each peripheral.
// To reduce power, leave clocks to unused peripherals disabled
//
// Note: If a peripherals clock is not enabled then you cannot
```

```
// read or write to the registers for that peripheral

void InitPeripheralClocks(void)
{
   EALLOW;

// LOSPCP prescale register settings, normally it will be set to default
values

   GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 3;  // GPIO18 = XCLKOUT
   SysCtrlRegs.LOSPCP.all = 0x0000;

// XCLKOUT to SYSCLKOUT ratio.  By default XCLKOUT = 1/4 SYSCLKOUT
   SysCtrlRegs.XCLK.bit.XCLKOUTDIV=2;

// Peripheral clock enables set for the selected peripherals.
// If you are not using a peripheral leave the clock off
// to save on power.
//
// Note: not all peripherals are available on all F2806x derivates.
// Refer to the datasheet for your particular device.
//
// This function is not written to be an example of efficient code.

   SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;    // ePWM1
   SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;    // ePWM2
   SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;    // ePWM3
   SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;    // ePWM4
   SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;    // ePWM5
   SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;    // ePWM6
   SysCtrlRegs.PCLKCR1.bit.EPWM7ENCLK = 1;    // ePWM7
   SysCtrlRegs.PCLKCR1.bit.EPWM8ENCLK = 1;    // ePWM8

   SysCtrlRegs.PCLKCR0.bit.HRPWMENCLK = 1;    // HRPWM
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;     // Enable TBCLK within the
ePWM

   SysCtrlRegs.PCLKCR1.bit.EQEP1ENCLK = 1;    // eQEP1
   SysCtrlRegs.PCLKCR1.bit.EQEP2ENCLK = 1;    // eQEP2

   SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 1;    // eCAP1
   SysCtrlRegs.PCLKCR1.bit.ECAP2ENCLK = 1;    // eCAP2
   SysCtrlRegs.PCLKCR1.bit.ECAP3ENCLK = 1;    // eCAP3

   SysCtrlRegs.PCLKCR2.bit.HRCAP1ENCLK = 1;      // HRCAP1
   SysCtrlRegs.PCLKCR2.bit.HRCAP2ENCLK = 1;      // HRCAP2
   SysCtrlRegs.PCLKCR2.bit.HRCAP3ENCLK = 1;      // HRCAP3
   SysCtrlRegs.PCLKCR2.bit.HRCAP4ENCLK = 1;   // HRCAP4

   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;      // ADC
   SysCtrlRegs.PCLKCR3.bit.COMP1ENCLK = 1;    // COMP1
   SysCtrlRegs.PCLKCR3.bit.COMP2ENCLK = 1;    // COMP2
   SysCtrlRegs.PCLKCR3.bit.COMP3ENCLK = 1;    // COMP3
```

```
   SysCtrlRegs.PCLKCR3.bit.CPUTIMER0ENCLK = 1; // CPU Timer 0
   SysCtrlRegs.PCLKCR3.bit.CPUTIMER1ENCLK = 1; // CPU Timer 1
   SysCtrlRegs.PCLKCR3.bit.CPUTIMER2ENCLK = 1; // CPU Timer 2

   SysCtrlRegs.PCLKCR3.bit.DMAENCLK = 1;      // DMA

   SysCtrlRegs.PCLKCR3.bit.CLA1ENCLK = 1;     // CLA1

   SysCtrlRegs.PCLKCR3.bit.USB0ENCLK = 1;   // USB0

   SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;     // I2C-A
   SysCtrlRegs.PCLKCR0.bit.SPIAENCLK = 1;     // SPI-A
   SysCtrlRegs.PCLKCR0.bit.SPIBENCLK = 1;     // SPI-B
   SysCtrlRegs.PCLKCR0.bit.SCIAENCLK = 1;     // SCI-A
   SysCtrlRegs.PCLKCR0.bit.SCIBENCLK = 1;     // SCI-B
   SysCtrlRegs.PCLKCR0.bit.MCBSPAENCLK = 1;   // McBSP-A
   SysCtrlRegs.PCLKCR0.bit.ECANAENCLK=1;      // eCAN-A

   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;     // Enable TBCLK within the
ePWM

   EDIS;
}

//---------------------------------------------------------------------
----
// Example: CsmUnlock:
//---------------------------------------------------------------------
----
// This function unlocks the CSM. User must replace 0xFFFF's with current
// password for the DSP. Returns 1 if unlock is successful.

#define STATUS_FAIL         0
#define STATUS_SUCCESS      1

Uint16 CsmUnlock()
{
    volatile Uint16 temp;

    // Load the key registers with the current password. The 0xFFFF's are
dummy
    // passwords.  User should replace them with the correct password for
the DSP.

    EALLOW;
    CsmRegs.KEY0 = 0xFFFF;
    CsmRegs.KEY1 = 0xFFFF;
    CsmRegs.KEY2 = 0xFFFF;
    CsmRegs.KEY3 = 0xFFFF;
    CsmRegs.KEY4 = 0xFFFF;
    CsmRegs.KEY5 = 0xFFFF;
    CsmRegs.KEY6 = 0xFFFF;
    CsmRegs.KEY7 = 0xFFFF;
    EDIS;
```

```
    // Perform a dummy read of the password locations
    // if they match the key values, the CSM will unlock

    temp = CsmPwl.PSWD0;
    temp = CsmPwl.PSWD1;
    temp = CsmPwl.PSWD2;
    temp = CsmPwl.PSWD3;
    temp = CsmPwl.PSWD4;
    temp = CsmPwl.PSWD5;
    temp = CsmPwl.PSWD6;
    temp = CsmPwl.PSWD7;

    // If the CSM unlocked, return succes, otherwise return
    // failure.
    if (CsmRegs.CSMSCR.bit.SECURE == 0) return STATUS_SUCCESS;
    else return STATUS_FAIL;

}

//---------------------------------------------------------------------------
// Example: IntOsc1Sel:
//---------------------------------------------------------------------------
// This function switches to Internal Oscillator 1 and turns off all
other clock
// sources to minimize power consumption

void IntOsc1Sel (void) {
    EALLOW;
    SysCtrlRegs.CLKCTL.bit.INTOSC1OFF = 0;
    SysCtrlRegs.CLKCTL.bit.OSCCLKSRCSEL=0;  // Clk Src = INTOSC1
    SysCtrlRegs.CLKCTL.bit.XCLKINOFF=1;     // Turn off XCLKIN
    SysCtrlRegs.CLKCTL.bit.XTALOSCOFF=1;    // Turn off XTALOSC
    SysCtrlRegs.CLKCTL.bit.INTOSC2OFF=1;    // Turn off INTOSC2
    EDIS;
}

//---------------------------------------------------------------------------
// Example: IntOsc2Sel:
//---------------------------------------------------------------------------
// This function switches to Internal oscillator 2 from External
Oscillator
// and turns off all other clock sources to minimize power consumption
// NOTE: If there is no external clock connection, when switching from
//       INTOSC1 to INTOSC2, EXTOSC and XLCKIN must be turned OFF prior
//       to switching to internal oscillator 1

void IntOsc2Sel (void) {
    EALLOW;
    SysCtrlRegs.CLKCTL.bit.INTOSC2OFF = 0;     // Turn on INTOSC2
```

```
    SysCtrlRegs.CLKCTL.bit.OSCCLKSRC2SEL = 1;  // Switch to INTOSC2
    SysCtrlRegs.CLKCTL.bit.XCLKINOFF = 1;      // Turn off XCLKIN
    SysCtrlRegs.CLKCTL.bit.XTALOSCOFF = 1;     // Turn off XTALOSC
    SysCtrlRegs.CLKCTL.bit.OSCCLKSRCSEL = 1;   // Switch to Internal
Oscillator 2
    SysCtrlRegs.CLKCTL.bit.WDCLKSRCSEL = 1;    // Switch Watchdog Clk Src
to INTOSC2
    SysCtrlRegs.CLKCTL.bit.INTOSC1OFF = 1;     // Turn off INTOSC1
    EDIS;
}

//---------------------------------------------------------------------------
// Example: XtalOscSel:
//---------------------------------------------------------------------------
// This function switches to External CRYSTAL oscillator and turns off
all other clock
// sources to minimize power consumption. This option may not be
available on all
// device packages

void XtalOscSel (void)  {
     EALLOW;
     SysCtrlRegs.CLKCTL.bit.XTALOSCOFF = 0;     // Turn on XTALOSC
     SysCtrlRegs.CLKCTL.bit.XCLKINOFF = 1;      // Turn off XCLKIN
     SysCtrlRegs.CLKCTL.bit.OSCCLKSRC2SEL = 0;  // Switch to external
clock
     SysCtrlRegs.CLKCTL.bit.OSCCLKSRCSEL = 1;   // Switch from INTOSC1 to
INTOSC2/ext clk
     SysCtrlRegs.CLKCTL.bit.WDCLKSRCSEL = 1;    // Switch Watchdog Clk
Src to external clock
     SysCtrlRegs.CLKCTL.bit.INTOSC2OFF = 1;     // Turn off INTOSC2
     SysCtrlRegs.CLKCTL.bit.INTOSC1OFF = 1;     // Turn off INTOSC1
     EDIS;
}


//---------------------------------------------------------------------------
// Example: ExtOscSel:
//---------------------------------------------------------------------------
// This function switches to External oscillator and turns off all other
clock
// sources to minimize power consumption.

void ExtOscSel (void)  {
     EALLOW;
     SysCtrlRegs.XCLK.bit.XCLKINSEL = 1;       // 1-GPIO19 = XCLKIN, 0-
GPIO38 = XCLKIN
     SysCtrlRegs.CLKCTL.bit.XTALOSCOFF = 1;    // Turn on XTALOSC
     SysCtrlRegs.CLKCTL.bit.XCLKINOFF = 0;     // Turn on XCLKIN
     SysCtrlRegs.CLKCTL.bit.OSCCLKSRC2SEL = 0; // Switch to external
```

```
clock
     SysCtrlRegs.CLKCTL.bit.OSCCLKSRCSEL = 1;  // Switch from INTOSC1 to
INTOSC2/ext clk
     SysCtrlRegs.CLKCTL.bit.WDCLKSRCSEL = 1;   // Switch Watchdog Clk Src
to external clock
     SysCtrlRegs.CLKCTL.bit.INTOSC2OFF = 1;    // Turn off INTOSC2
     SysCtrlRegs.CLKCTL.bit.INTOSC1OFF = 1;    // Turn off INTOSC1
     EDIS;
}



//===========================================================================
====
// End of file.
//===========================================================================
=====



;// TI File $Revision: /main/3 $
;// Checkin $Date: January 4, 2011   10:10:51 $
;//###########################################################################
#####
;//
;// FILE:  F2806x_usDelay.asm
;//
;// TITLE: Simple delay function
;//
;// DESCRIPTION:
;//
;// This is a simple delay function that can be used to insert a
specified
;// delay into code.
;//
;// This function is only accurate if executed from internal zero-
waitstate
;// SARAM. If it is executed from waitstate memory then the delay will be
;// longer then specified.
;//
;// To use this function:
;//
;//  1 - update the CPU clock speed in the F2806x_Examples.h
;//    file. For example:
;//    #define CPU_RATE 12.500L // for an 80MHz CPU clock speed
;//
;//  2 - Call this function by using the DELAY_US(A) macro
;//    that is defined in the F2806x_Examples.h file.  This macro
;//    will convert the number of microseconds specified
;//    into a loop count for use with this function.
;//    This count will be based on the CPU frequency you specify.
;//
;//  3 - For the most accurate delay
```

```
;//    - Execute this function in 0 waitstate RAM.
;//    - Disable interrupts before calling the function
;//      If you do not disable interrupts, then think of
;//      this as an "at least" delay function as the actual
;//      delay may be longer.
;//
;//  The C assembly call from the DELAY_US(time) macro will
;//  look as follows:
;//
;//  extern void Delay(long LoopCount);
;//
;//        MOV   AL,#LowLoopCount
;//        MOV   AH,#HighLoopCount
;//        LCR   _Delay
;//
;//  Or as follows (if count is less then 16-bits):
;//
;//        MOV   ACC,#LoopCount
;//        LCR   _Delay
;//
;//
;//###########################################################################
#####
;// $TI Release: 2806x C/C++ Header Files V1.10 $
;// $Release Date: April 7, 2011 $
;//###########################################################################
#####

       .def _DSP28x_usDelay
       .sect "ramfuncs"

        .global  __DSP28x_usDelay
_DSP28x_usDelay:
        SUB    ACC,#1
        BF     _DSP28x_usDelay,GEQ    ;; Loop if ACC >= 0
        LRETR

;There is a 9/10 cycle overhead and each loop
;takes five cycles. The LoopCount is given by
;the following formula:
;  DELAY_CPU_CYCLES = 9 + 5*LoopCount
; LoopCount = (DELAY_CPU_CYCLES - 9) / 5
; The macro DELAY_US(A) performs this calculation for you
;
;//===========================================================================
=====
;// End of file.
;//===========================================================================
=====
```

# Appendice B

# Datasheet

19-0236; Rev 1; 6/99

MAXIM

## Precision, Quad, SPST Analog Switches



### General Description

The MAX391/MAX392/MAX393 are precision, quad, single-pole/single-throw (SPST) analog switches designed to operate at +3V, +5V, or ±5V. The MAX391 has four normally closed (NC) switches, and the MAX392 has four normally open (NO) switches. The MAX393 has two NO and two NC switches. All three devices offer low leakage (100pA max) and fast switching speeds ($t_{ON} \le 130ns$, $t_{OFF} \le 75ns$). Power consumption is just 1µW—ideal for battery-operated equipment. All devices operate from a single +3V to +15V supply or from dual ±3.0V to ±8V supplies.

With ±5V supplies, the MAX391/MAX392/MAX393 offer guaranteed 2Ω max channel-to-channel matching, 30Ω max on-resistance ($R_{ON}$), and 4Ω max $R_{ON}$ flatness over the specified range.

These switches are also fully specified for single +5V operation, with 2Ω max $R_{ON}$ match, 60Ω max $R_{ON}$, and 6Ω max flatness.

These low-voltage switches also offer 5pC max charge injection, and ESD protection is greater than 2000V, per method 3015.7.

### Applications

| | |
|---|---|
| Battery-Operated Systems | Sample-and-Hold Circuits |
| Heads-Up Displays | Guidance and Control Systems |
| Audio and Video Switching | Military Radios |
| Test Equipment | Communications Systems |
| ±5V DACs and ADCs | PBX, PABX |

### Features

- ♦ **Low On-Resistance, 20Ω Typical**
- ♦ **Guaranteed On-Resistance Match Between Channels, <2Ω**
- ♦ **Guaranteed On-Resistance Flatness Over Signal Range, 4Ω Max**
- ♦ **Guaranteed Charge Injection, <5pC**
- ♦ **Improved Leakage Over Temperature, <2.5nA at +85°C**
- ♦ **Electrostatic Discharge >2000V per Method 3015.7**
- ♦ **Single-Supply Operation (+3V to +15V) Bipolar-Supply Operation (±3V to ±8V)**
- ♦ **Low Power Consumption, <1µW**
- ♦ **TTL/CMOS-Logic Compatible**

### Ordering Information

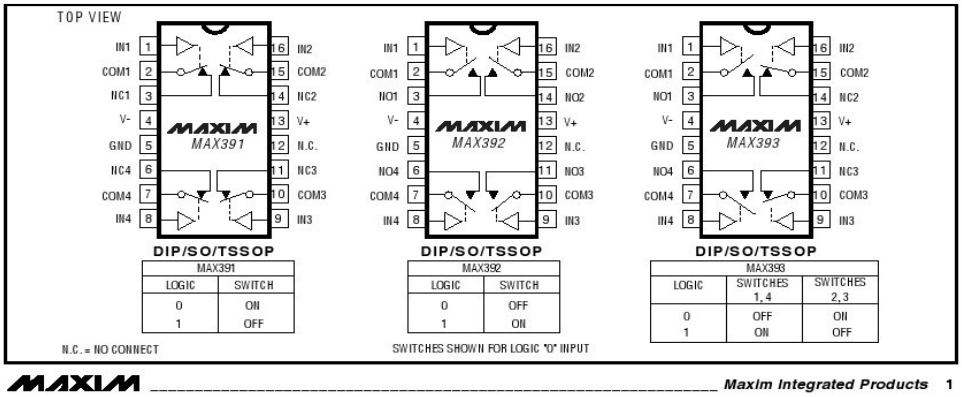
| PART | TEMP. RANGE | PIN-PACKAGE |
|---|---|---|
| **MAX391**CPE | 0°C to +70°C | 16 Plastic DIP |
| MAX391CSE | 0°C to +70°C | 16 Narrow SO |
| MAX391CUE | 0°C to +70°C | 16 TSSOP |
| MAX391C/D | 0°C to +70°C | Dice* |
| MAX391EPE | -40°C to +85°C | 16 Plastic DIP |
| MAX391ESE | -40°C to +85°C | 16 Narrow SO |
| MAX391EUE | -40°C to +85°C | 16 TSSOP |

***Ordering Information continued at end of data sheet.***

** Contact factory for dice specifications.*

*** Contact factory for availability and processing to MIL-STD-883.*

### Pin Configurations/Functional Diagrams/Truth Tables



## ABSOLUTE MAXIMUM RATINGS

Voltage Referenced to V-
V+ ..... -0.3V to +17V
GND ..... -0.3V to +17V
GND ..... -0.3V to (V+ + 0.3V)
$V_{IN}$, $V_{COM}$, $V_{NC}$, $V_{NO}$ (Note 1) ..... V- to V+
Current (any terminal) ..... 30mA
Peak Current, COM, NO, NC
(pulsed at 1ms, 10% duty cycle max) ..... 100mA
ESD per Method 3015.7 ..... >2000V

Continuous Power Dissipation ($T_A$ = +70°C)
Plastic DIP (derate 10.53mW/°C above+70°C) ..... 842mW
Narrow SO (derate 8.70mW/°C above +70°C) ..... 696mW
TSSOP (derate 6.7mW/°C above +70°C) ..... 457mW
CERDIP (derate 10.00mW/°C above +70°C) ..... 800mW
Operating Temperature Ranges
MAX39_C_ _ ..... 0°C to +70°C
MAX39_E_ _ ..... -40°C to +85°C
MAX39_M_ _ ..... -55°C to +125°C
Storage Temperature Range ..... -65°C to +150°C
Lead Temperature (soldering, 10s) ..... +300°C

**Note 1:** Signals on NC, NO, COM, or IN exceeding V+ or V- are clamped by internal diodes. Limit forward diode current to maximum current rating.

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## ELECTRICAL CHARACTERISTICS—Dual Supplies

(V+ = +5V ±10%, V- = -5V ±10%, GND = 0V, $V_{INH}$ = 2.4V, $V_{INL}$ = 0.8V, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise noted.)

| PARAMETER | SYMBOL | CONDITIONS | | | MIN | TYP (Note 2) | MAX | UNITS |
|---|---|---|---|---|---|---|---|---|
| **ANALOG SWITCH** | | | | | | | | |
| Analog Signal Range | $V_{COM}$, $V_{NO}$, $V_{NC}$ | (Note 3) | | | V- | | V+ | V |
| On-Resistance | $R_{ON}$ | V+ = 4.5V, V- = -4.5V, $I_{COM}$ = -10mA, $V_{NO}$ or $V_{NC}$ = ±3.5V | $T_A$ = +25°C | C, E | | 20 | 35 | Ω |
| | | | | M | | 20 | 30 | |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 45 | |
| On-Resistance Match Between Channels (Note 4) | $\Delta R_{ON}$ | V+ = 5V, V- = -5V, $I_{COM}$ = -10mA, $V_{NO}$ or $V_{NC}$ = ±3V | $T_A$ = +25°C | | | 0.3 | 2 | Ω |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 4 | |
| On-Resistance Flatness (Note 5) | $R_{FLAT(ON)}$ | V+ = 5V, V- = -5V, $I_{COM}$ = -10mA, $V_{NO}$ or $V_{NC}$ = ±3V | $T_A$ = +25°C | | | 1 | 4 | Ω |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 6 | |
| NO or NC Off Leakage Current (Note 6) | $I_{NO(OFF)}$ or $I_{NC(OFF)}$ | V+ = 5.5V, V- = -5.5V, $V_{COM}$ = ±4.5V, $V_{NO}$ or $V_{NC}$ = ∓4.5V | $T_A$ = +25°C | | -0.1 | 0.01 | 0.1 | nA |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | C, E | -2.5 | | 2.5 | |
| | | | | M | -5 | | 5 | |
| COM Off Leakage Current (Note 6) | $I_{COM(OFF)}$ | V+ = 5.5V, V- = -5.5V, $V_{COM}$ = ±4.5V, $V_{NO}$ or $V_{NC}$ = ∓4.5V | $T_A$ = +25°C | | -0.1 | 0.01 | 0.1 | nA |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | C, E | -2.5 | | 2.5 | |
| | | | | M | -5 | | 5 | |
| COM On Leakage Current (Note 6) | $I_{COM(ON)}$ | V+ = 5.5V, V- = -5.5V, $V_{COM}$ = ±4.5V, $V_{NO}$ or $V_{NC}$ = ±4.5V | $T_A$ = +25°C | | -0.2 | 0.01 | 0.2 | nA |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | C, E | -5.0 | | 5.0 | |
| | | | | M | -20 | | 20 | |

## ELECTRICAL CHARACTERISTICS—Dual Supplies (continued)

(V+ = +5V ±10%, V- = -5V ±10%, GND = 0V, $V_{INH}$ = 2.4V, $V_{INL}$ = 0.8V, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise noted.)

| PARAMETER | SYMBOL | CONDITIONS | | MIN | TYP (Note 2) | MAX | UNITS |
|---|---|---|---|---|---|---|---|
| **LOGIC INPUT** | | | | | | | |
| Input Current with Input Voltage High | $I_{INH}$ | IN = 2.4V, all others = 0.8V | | -0.5 | 0.005 | 0.5 | µA |
| Input Current with Input Voltage Low | $I_{INL}$ | IN = 0.8V, all others = 2.4V | | -0.5 | 0.005 | 0.5 | µA |
| **DYNAMIC** | | | | | | | |
| Turn-On Time | $t_{ON}$ | $V_{COM}$ = ±3V, Figure 2 | $T_A$ = +25°C | | 65 | 130 | ns |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | 175 | |
| Turn-Off Time | $t_{OFF}$ | $V_{COM}$ = ±3V, Figure 2 | $T_A$ = +25°C | | 35 | 75 | ns |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | 100 | |
| Break-Before-Make Time Delay (Note 3) | $t_D$ | MAX393 only, $R_L$ = 300Ω, $C_L$ = 35pF, Figure 3 | | 5 | 10 | | ns |
| Charge Injection (Note 3) | Q | $C_L$ = 1.0nF, $V_{GEN}$ = 0V, $R_{GEN}$ = 0Ω, Figure 4 | $T_A$ = +25°C | | 2 | 5 | pC |
| Off Isolation (Note 7) | OIRR | $R_L$ = 50Ω, $C_L$ = 5pF, f = 1MHz, Figure 5 | $T_A$ = +25°C | | 72 | | dB |
| Crosstalk (Note 8) | | $R_L$ = 50Ω, $C_L$ = 5pF, f = 1MHz, Figure 6 | $T_A$ = +25°C | | 85 | | dB |
| NC or NO Capacitance | $C_{(OFF)}$ | f = 1MHz, Figure 7 | $T_A$ = +25°C | | 9 | | pF |
| COM Off Capacitance | $C_{COM(OFF)}$ | f = 1MHz, Figure 7 | $T_A$ = +25°C | | 9 | | pF |
| COM On Capacitance | $C_{COM(ON)}$ | f = 1MHz, Figure 8 | $T_A$ = +25°C | | 22 | | pF |
| **SUPPLY** | | | | | | | |
| Power-Supply Range | | | | -8.0 | | +8.0 | V |
| Positive Supply Current | I+ | V+ = 5.5V, V- = -5.5V, $V_{IN}$ = 0V or V+, All channels on or off | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | -1 | | 1 | µA |
| Negative Supply Current | I- | V+ = 5.5V, V- = -5.5V, $V_{IN}$ = 0V or V+, All channels on or off | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | -1 | | 1 | µA |

## ELECTRICAL CHARACTERISTICS—Single +5V Supply

(V+ = +5V ±10%, V- = 0V ±10%, GND = 0V, $V_{INH}$ = 2.4V, $V_{INL}$ = 0.8V, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise noted.)

| PARAMETER | SYMBOL | CONDITIONS | | | MIN | TYP (Note 2) | MAX | UNITS |
|---|---|---|---|---|---|---|---|---|
| **ANALOG SWITCH** | | | | | | | | |
| Analog Signal Range | $V_{COM}$, $V_{NO}$, $V_{NC}$ | (Note 3) | | | 0 | | V+ | V |
| On-Resistance | $R_{ON}$ | V+ = 4.5V, $I_{COM}$ = -10mA, $V_{NO}$ or $V_{NC}$ = 3.5V | $T_A$ = +25°C | | | 30 | 60 | Ω |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 75 | |
| On-Resistance Match Between Channels (Note 4) | $\Delta R_{ON}$ | V+ = 5V, $I_{COM}$ = -1.0mA, $V_{NO}$ or $V_{NC}$ = 3V | $T_A$ = +25°C | | | 0.8 | 2 | Ω |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 4 | |
| On-Resistance Flatness (Notes 3, 5) | $R_{FLAT(ON)}$ | V+ = 5V, $I_{COM}$ = -1.0mA, $V_{NO}$ or $V_{NC}$ = 1V, 3V | $T_A$ = +25°C | | | 2 | 6 | Ω |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 8 | |
| NO or NC Off Leakage Current (Note 9) | $I_{NO(OFF)}$ or $I_{NC(OFF)}$ | V+ = 5.5V, $V_{COM}$ = 0V, $V_{NO}$ or $V_{NC}$ = 4.5V | $T_A$ = +25°C | | -0.25 | 0.01 | 0.25 | nA |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | C, E | -0.1 | | 0.1 | |
| | | | | M | -2.5 | | 2.5 | |
| COM Off Leakage Current (Note 9) | $I_{COM(OFF)}$ | V+ = 5.5V, $V_{COM}$ = 0V, $V_{NO}$ or $V_{NC}$ = 4.5V | $T_A$ = +25°C | | -0.1 | | 0.1 | nA |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | C, E | -2.5 | | 2.5 | |
| | | | | M | -5.0 | | 5.0 | |
| COM On Leakage Current (Note 9) | $I_{COM(ON)}$ | V+ = 5.5V, $V_{COM}$ = 5V, $V_{NO}$ or $V_{NC}$ = 4.5V | $T_A$ = +25°C | | -0.2 | | 0.2 | nA |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | C, E | -5.0 | | 5.0 | |
| | | | | M | -20 | | 20 | |
| **DYNAMIC** | | | | | | | | |
| Turn-On Time | $t_{ON}$ | $V_{NO}$ or $V_{NC}$ = 3V | $T_A$ = +25°C | | | 85 | 170 | ns |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 240 | |
| Turn-Off Time | $t_{OFF}$ | $V_{NO}$ or $V_{NC}$ = 3V | $T_A$ = +25°C | | | 25 | 50 | ns |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | | 100 | |
| Break-Before-Make Time Delay (Note 3) | $t_D$ | MAX393 only, $R_L$ = 300Ω, $C_L$ = 35pF | | | 10 | | | ns |
| Charge Injection (Note 3) | Q | $C_L$ = 1.0nF, $V_{GEN}$ = 0V, $R_{GEN}$ = 0Ω, Figure 4 | $T_A$ = +25°C | | | 1 | 5 | pC |
| **SUPPLY** | | | | | | | | |
| Positive Supply Current | I+ | V+ = 5.5V, $V_{IN}$ = 0V or V+, all channels on or off | | | -1 | | 1 | µA |
| Negative Supply Current | I- | V+ = 5.5V, $V_{IN}$ = 0V or V+, all channels on or off | | | -1 | | 1 | µA |

## ELECTRICAL CHARACTERISTICS—Single +3.3V Supply

(V+ = +3.0V to +3.6V, GND = 0V, $V_{INH}$ = 2.4V, $V_{INL}$ = 0.8V, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise noted.)

| PARAMETER | SYMBOL | CONDITIONS | | MIN | TYP (Note 2) | MAX | UNITS |
|---|---|---|---|---|---|---|---|
| **ANALOG SWITCH** | | | | | | | |
| Analog Signal Range | $V_{COM}$, $V_{NO}$, $V_{NC}$ | (Note 3) | | 0 | | V+ | V |
| Channel On-Resistance | $R_{ON}$ | V+ = 3V, $I_{COM}$ = -1.0mA, $V_{NO}$ or $V_{NC}$ = 1.5V | $T_A$ = +25°C | | 83 | 175 | Ω |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | 275 | |
| **DYNAMIC** | | | | | | | |
| Turn-On Time (Note 3) | $t_{ON}$ | $V_{NO}$ or $V_{NC}$ = 1.5V | $T_A$ = +25°C | | 160 | 400 | ns |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | 500 | |
| Turn-Off Time (Note 3) | $t_{OFF}$ | $V_{NO}$ or $V_{NC}$ = 1.5V | $T_A$ = +25°C | | 40 | 125 | ns |
| | | | $T_A$ = $T_{MIN}$ to $T_{MAX}$ | | | 175 | |
| Break-Before-Make Time Delay (Note 3) | $t_D$ | MAX393 only, $R_L$ = 300Ω, $C_L$ = 35pF | $T_A$ = +25°C | 20 | | | ns |
| Charge Injection (Note 3) | Q | $C_L$ = 1.0nF, $V_{GEN}$ = 0V, $R_{GEN}$ = 0V | $T_A$ = +25°C | | 1 | 5 | pC |
| **SUPPLY** | | | | | | | |
| Positive Supply Current | I+ | V+ = 3.6V, $V_{IN}$ = 0V or V+, all channels on or off | | -1 | | 1 | µA |
| Negative Supply Current | I- | V+ = 3.6V, $V_{IN}$ = 0V or V+, all channels on or off | | -1 | | 1 | µA |

**Note 2:** The algebraic convention, where the most negative value is a minimum and the most positive value a maximum, is used in this data sheet.
**Note 3:** Guaranteed by design.
**Note 4:** $\Delta R_{ON}$ = $\Delta R_{ON}$ max - $\Delta R_{ON}$ min.
**Note 5:** Flatness is defined as the difference between the maximum and minimum value of on-resistance as measured over the specified analog signal range.
**Note 6:** Leakage parameters are 100% tested at maximum rated hot temperature and guaranteed by correlation at +25°C.
**Note 7:** Off Isolation = $20\log_{10}$ [ $V_{COM}$ / ($V_{NC}$ or $V_{NO}$) ], $V_{COM}$ = output, $V_{NC}$ or $V_{NO}$ = input to off switch.
**Note 8:** Between any two switches.
**Note 9:** Leakage testing at single supply is guaranteed by testing with dual singles.

Burr-Brown Products from Texas Instruments

# DAC8531

SBAS192B – MARCH 2001 – REVISED JUNE 2003

# Low-Power, Rail-to-Rail Output, 16-Bit Serial Input DIGITAL-TO-ANALOG CONVERTER

## FEATURES

- *micro*Power OPERATION: 250μA at 5V
- POWER-ON RESET TO ZERO
- POWER SUPPLY: +2.7V to +5.5V
- ENSURED MONOTONIC BY DESIGN
- SETTLING TIME: 10μs to ±0.003 FSR
- LOW-POWER SERIAL INTERFACE WITH SCHMITT-TRIGGERED INPUTS
- ON-CHIP OUTPUT BUFFER AMPLIFIER, RAIL-TO-RAIL OPERATION
- $\overline{\text{SYNC}}$ INTERRUPT FACILITY
- PACKAGES: MSOP-8 and 3x3 SON-8 (same size as QFN)

## APPLICATIONS

- PROCESS CONTROL
- DATA ACQUISITION SYSTEMS
- CLOSED-LOOP SERVO-CONTROL
- PC PERIPHERALS
- PORTABLE INSTRUMENTATION
- PROGRAMMABLE ATTENUATION

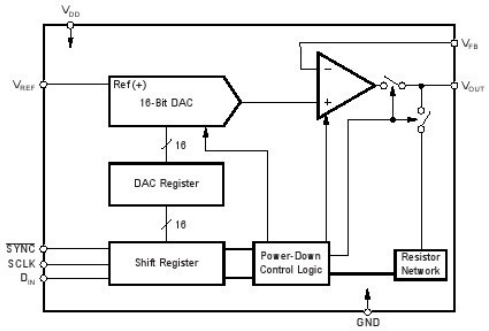
## DESCRIPTION

The DAC8531 is a low-power, single, 16-bit buffered voltage output Digital-to-Analog Converter (DAC). Its on-chip precision output amplifier allows rail-to-rail output swing to be achieved. The DAC8531 uses a versatile three-wire serial interface that operates at clock rates up to 30MHz and is compatible with standard SPI™, QSPI™, Microwire™, and Digital Signal Processor (DSP) interfaces.

The DAC8531 requires an external reference voltage to set the output range of the DAC. The DAC8531 incorporates a power-on reset circuit that ensures that the DAC output powers up at 0V and remains there until a valid write takes place to the device. The DAC8531 contains a power-down feature, accessed over the serial interface, that reduces the current consumption of the device to 200nA at 5V.

The low power consumption of this part in normal operation makes it ideally suited to portable battery-operated equipment. The power consumption is 2mW at 5V reducing to 1μW in power-down mode.

The DAC8531 is available in both MSOP-8 and 3x3 SON-8 (same size as QFN) packages.



### ABSOLUTE MAXIMUM RATINGS(1)

| | |
|---|---|
| $V_{DD}$ to GND | –0.3V to +6V |
| Digital Input Voltage to GND | –0.3V to $+V_{DD}$ + 0.3V |
| $V_{OUT}$ to GND | –0.3V to $+V_{DD}$ + 0.3V |
| Operating Temperature Range | –40°C to +105°C |
| Storage Temperature Range | –65°C to +150°C |
| Junction Temperature Range ($T_J$ max) | +150°C |
| Power Dissipation | $(T_J$ max — $T_A)/\theta_{JA}$ |
| $\theta_{JA}$ Thermal Impedance | 206°C/W |
| $\theta_{JC}$ Thermal Impedance | 44°C/W |
| Lead Temperature, Soldering: | |
| Vapor Phase (60s) | +215°C |
| Infrared (15s) | +220°C |

NOTE: (1) Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. Exposure to absolute maximum conditions for extended periods may affect device reliability.

## ELECTROSTATIC DISCHARGE SENSITIVITY

This integrated circuit can be damaged by ESD. Texas Instruments recommends that all integrated circuits be handled with appropriate precautions. Failure to observe proper handling and installation procedures can cause damage.

ESD damage can range from subtle performance degradation to complete device failure. Precision integrated circuits may be more susceptible to damage because very small parametric changes could cause the device not to meet its published specifications.

### PACKAGE/ORDERING INFORMATION

| PRODUCT | MINIMUM RELATIVE ACCURACY (LSB) | DIFFERENTIAL NONLINEARITY (LSB) | PACKAGE-LEAD | PACKAGE DESIGNATOR(1) | SPECIFICATION TEMPERATURE RANGE | PACKAGE MARKING | ORDERING NUMBER | TRANSPORT MEDIA, QUANTITY |
|---|---|---|---|---|---|---|---|---|
| DAC8531E | ±64 | ±1 | MSOP-8 | DGK | –40°C to +105°C | D31 | DAC8531E/250 | Tape and Reel, 250 |
| " | " | " | " | " | " | " | DAC8531E/2K5 | Tape and Reel, 2500 |
| DAC8531I | ±64 | ±1 | SON-8 | DRB | –40°C to +105°C | D31 | DAC8531IDRBT | Tape and Reel, 250 |
| DAC8531I | " | " | " | " | " | " | DAC8531IDRBR | Tape and Reel, 2500 |

NOTE: (1) For the most current specifications and package information, refer to our web site at www.ti.com.

## ELECTRICAL CHARACTERISTICS

$V_{DD}$ = +2.7V to +5.5V. –40°C to +105°C, unless otherwise specified.

| PARAMETER | CONDITIONS | DAC8531E | | | UNITS |
|---|---|---|---|---|---|
| | | MIN | TYP | MAX | |
| **STATIC PERFORMANCE** (1) | | | | | |
| Resolution | | 16 | | | Bits |
| Relative Accuracy | | | | ±0.098 | % of FSR |
| Differential Nonlinearity | Ensured Monotonic by Design | | | ±1 | LSB |
| Zero Code Error | All Zeroes Loaded to DAC Register | | +5 | +20 | mV |
| Full-Scale Error | All Ones Loaded to DAC Register | | –0.15 | –1.25 | % of FSR |
| Gain Error | | | | ±1.25 | % of FSR |
| Zero Code Error Drift | | | ±20 | | µV/°C |
| Gain Temperature Coefficient | | | ±5 | | ppm of FSR/°C |
| **OUTPUT CHARACTERISTICS** (2) | | | | | |
| Output Voltage Range | | 0 | | $V_{REF}$ | V |
| Output Voltage Settling Time | To ±0.003% FSR | | | | |
| | $0200_H$ to $FD00_H$ | | 8 | 10 | µs |
| | $R_L$ = 2kΩ; 0pF < $C_L$ < 200pF | | | | |
| | $R_L$ = 2kΩ; $C_L$ = 500pF | | 12 | | µs |
| Slew Rate | | | 1 | | V/µs |
| Capacitive Load Stability | $R_L$ = ∞ | | 470 | | pF |
| | $R_L$ = 2kΩ | | 1000 | | pF |
| Code Change Glitch Impulse | 1LSB Change Around Major Carry | | 20 | | nV-s |
| Digital Feedthrough | | | 0.5 | | nV-s |
| DC Output Impedance | | | 1 | | Ω |
| Short-Circuit Current | $V_{DD}$ = +5V | | 50 | | mA |
| | $V_{DD}$ = +3V | | 20 | | mA |
| Power-Up Time | Coming Out of Power-Down Mode | | | | |
| | $V_{DD}$ = +5V | | 2.5 | | µs |
| | Coming Out of Power-Down Mode | | | | |
| | $V_{DD}$ = +3V | | 5 | | µs |
| **REFERENCE INPUT** | | | | | |
| Reference Current | $V_{REF}$ = $V_{DD}$ = +5V | | 35 | 45 | µA |
| | $V_{REF}$ = $V_{DD}$ = +3.6V | | 20 | 30 | µA |
| Reference Input Range | | 0 | | $V_{DD}$ | V |
| Reference Input Impedance | | | 150 | | kΩ |

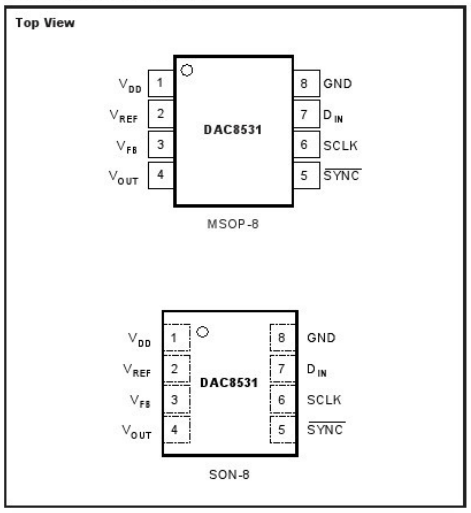
NOTES: (1) Linearity calculated using a reduced code range of 485 to 64714; output unloaded. (2) Ensured by design and characterization, not production tested.

## ELECTRICAL CHARACTERISTICS (Cont.)

$V_{DD}$ = +2.7V to +5.5V, –40°C to +105°C, unless otherwise specified.

| PARAMETER | CONDITIONS | DAC8531E MIN | DAC8531E TYP | DAC8531E MAX | UNITS |
|---|---|---|---|---|---|
| **LOGIC INPUTS**(2) | | | | | |
| Input Current | | | | ±1 | µA |
| $V_{IN}L$, Input LOW Voltage | $V_{DD}$ = +5V | | | 0.8 | V |
| $V_{IN}L$, Input LOW Voltage | $V_{DD}$ = +3V | | | 0.6 | V |
| $V_{IN}H$, Input HIGH Voltage | $V_{DD}$ = +5V | 2.4 | | | V |
| $V_{IN}H$, Input HIGH Voltage | $V_{DD}$ = +3V | 2.1 | | | V |
| Pin Capacitance | | | | 3 | pF |
| **POWER REQUIREMENTS** | | | | | |
| $V_{DD}$ | | 2.7 | | 5.5 | V |
| $I_{DD}$ (normal mode) | DAC Active and Excluding Load Current | | | | |
| $V_{DD}$ = +3.6V to +5.5V | $V_{IH} = V_{DD}$ and $V_{IL}$ = GND | | 250 | 400 | µA |
| $V_{DD}$ = +2.7V to +3.6V | $V_{IH} = V_{DD}$ and $V_{IL}$ = GND | | 240 | 390 | µA |
| $I_{DD}$ (all power-down modes) | | | | | |
| $V_{DD}$ = +3.6V to +5.5V | $V_{IH} = V_{DD}$ and $V_{IL}$ = GND | | 0.2 | 1 | µA |
| $V_{DD}$ = +2.7V to +3.6V | $V_{IH} = V_{DD}$ and $V_{IL}$ = GND | | 0.05 | 1 | µA |
| **POWER EFFICIENCY** | | | | | |
| $I_{OUT}/I_{DD}$ | $I_{LOAD}$ = 2mA, $V_{DD}$ = +5V | | 89 | | % |
| **TEMPERATURE RANGE** | | | | | |
| Specified Performance | | –40 | | +105 | °C |

### PIN CONFIGURATIONS




### PIN DESCRIPTION

| PIN | NAME | DESCRIPTION |
|---|---|---|
| 1 | $V_{DD}$ | Power-Supply Input, +2.7V to +5.5V. |
| 2 | $V_{REF}$ | Reference Voltage Input |
| 3 | $V_{FB}$ | Feedback connection for the output amplifier. |
| 4 | $V_{OUT}$ | Analog output voltage from DAC. The output amplifier has rail-to-rail operation. |
| 5 | $\overline{SYNC}$ | Level-triggered control input (active LOW). This is the frame sychronization signal for the input data. When $\overline{SYNC}$ goes LOW, it enables the input shift register and data is transferred in on the falling edges of the following clocks. The DAC is updated following the 24th clock cycle unless $\overline{SYNC}$ is taken HIGH before this edge, in which case the rising edge of $\overline{SYNC}$ acts as an interrupt and the write sequence is ignored by the DAC8531. |
| 6 | SCLK | Serial Clock Input. Data can be transferred at rates up to 30MHz. |
| 7 | $D_{IN}$ | Serial Data Input. Data is clocked into the 24-bit input shift register on the falling edge of the serial clock input. |
| 8 | GND | Ground reference point for all circuitry on the part. |

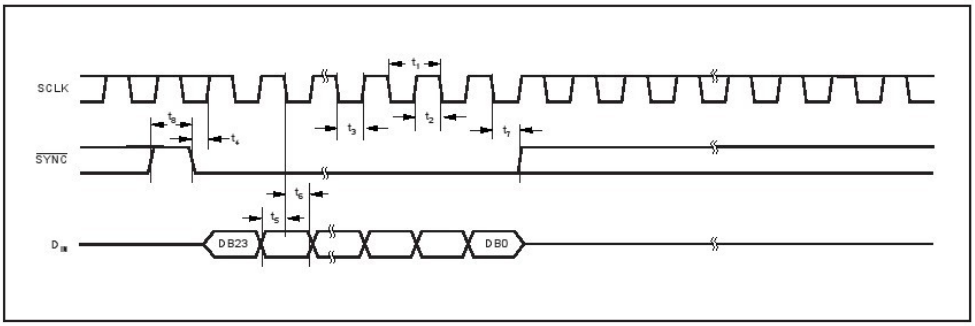
## TIMING CHARACTERISTICS(1, 2)

$V_{DD}$ = +2.7V to +5.5V; all specifications –40°C to +105°C unless otherwise noted.

| PARAMETER | DESCRIPTION | CONDITIONS | DAC8531E MIN | DAC8531E TYP | DAC8531E MAX | UNITS |
|---|---|---|---|---|---|---|
| $t_1$(3) | SCLK Cycle Time | $V_{DD}$ = 2.7V to 3.6V | 50 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 33 | | | ns |
| $t_2$ | SCLK HIGH Time | $V_{DD}$ = 2.7V to 3.6V | 13 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 13 | | | ns |
| $t_3$ | SCLK LOW Time | $V_{DD}$ = 2.7V to 3.6V | 22.5 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 13 | | | ns |
| $t_4$ | $\overline{SYNC}$ to SCLK Rising Edge Setup Time | $V_{DD}$ = 2.7V to 3.6V | 0 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 0 | | | ns |
| $t_5$ | Data Setup Time | $V_{DD}$ = 2.7V to 3.6V | 5 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 5 | | | ns |
| $t_6$ | Data Hold Time | $V_{DD}$ = 2.7V to 3.6V | 4.5 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 4.5 | | | ns |
| $t_7$ | SCLK Falling Edge to $\overline{SYNC}$ Rising Edge | $V_{DD}$ = 2.7V to 3.6V | 0 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 0 | | | ns |
| $t_8$ | Minimum $\overline{SYNC}$ HIGH Time | $V_{DD}$ = 2.7V to 3.6V | 50 | | | ns |
| | | $V_{DD}$ = 3.6V to 5.5V | 33 | | | ns |

NOTES: (1) All input signals are specified with $t_R = t_F$ = 5ns (10% to 90% of $V_{DD}$) and timed from a voltage level of $(V_{IL} + V_{IH})/2$. (2) See Serial Write Operation timing diagram, below. (3) Maximum SCLK frequency is 30MHz at $V_{DD}$ = +3.6V to +5.5V and 20MHz at $V_{DD}$ = +2.7V to +3.6V.

### SERIAL WRITE OPERATION

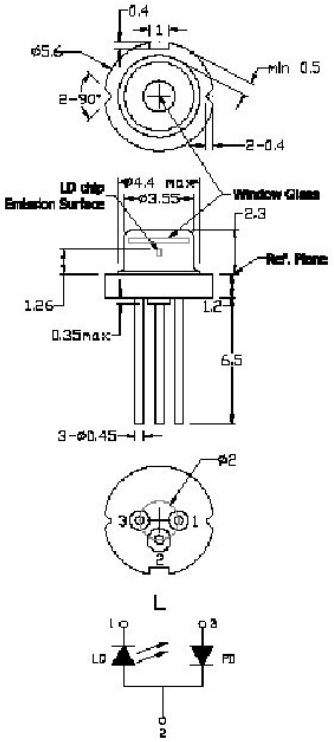


ROITHNER LASERTECHNIK GmbH
WIEDNER HAUPTSTRASSE 76 1040 VIENNA AUSTRIA
TEL. +43 1 586 52 43 -0, FAX. -44, OFFICE@ROITHNER-LASER.COM

# AlGaInP Visible Laser Diode — ADL-65052TL

DATE : 2005/10/18 Ver 1.0

## ★650nm 5mW 50°C Low Current Operation




- **Features**
  1. Low operating current
  2. High efficiency
  3. FFP single lateral mode

- **Applications**
  1. Laser pointers
  2. Industrial laser markers / measuring instruments
  3. Bar code readers

- **Absolute maximum ratings**

| Parameter | Symbol | Condition | Rating | Unit |
|---|---|---|---|---|
| Light output power | $P_O$ | CW | 7 | mW |
| Reverse voltage (LD) | $V_{RL}$ | - | 2 | V |
| Reverse voltage (PD) | $V_{RD}$ | - | 30 | V |
| Forward current (PD) | $I_{FD}$ | - | 10 | mA |
| Case temperature | $T_C$ | - | -10~+50 | °C |
| Storage temperature | $T_S$ | - | -40~+85 | °C |

- **Electrical and optical characteristics ($T_c$=25 °C)**

| Parameter | Symbol | Min. | Typ. | Max. | Unit | Conditions |
|---|---|---|---|---|---|---|
| Peak wavelength | $\lambda$ | 645 | 655 | 660 | nm | $P_o$=5mW |
| Threshold current | $I_{th}$ | - | 15 | 20 | mA | |
| Operating current | $I_{op}$ | - | 20 | 22 | mA | $P_o$=5mW |
| Operating voltage | $V_{op}$ | - | 2.2 | 2.5 | V | $P_o$=5mW |
| Differential efficiency | $\eta$ | 0.5 | 1.0 | 1.4 | mW/mA | $P_o$=3-5mW |
| Monitor current | $I_m$ | 0.05 | 0.1 | 0.3 | mA | $P_o$=5mW, $V_{RD}$=5V |
| Parallel divergence angle | $\theta_{\parallel}$ | 6 | 8 | 12 | deg | $P_o$=5mW |
| Perpendicular divergence angle | $\theta_{\perp}$ | 24 | 27 | 32 | deg | |
| Parallel FFP deviation angle | $\Delta\theta_{\parallel}$ | -3 | 0 | +3 | deg | |
| Perpendicular FFP deviation angle | $\Delta\theta_{\perp}$ | -3 | 0 | +3 | deg | |
| Emission point accuracy | $\Delta x \Delta y \Delta z$ | -80 | 0 | +80 | um | |

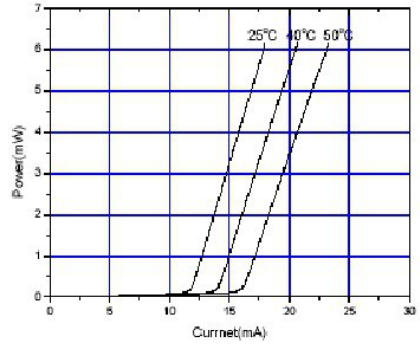
- **Precautions**
- Do not operate the device above maximum ratings. Doing so may cause unexpected and permanent damage to the device.
- Take precautions to avoid electrostatic discharge and/or momentary power spikes. A change in the characteristics of the laser or premature failure may result.
- Proper heat sinking of the device assures stability and lifetime. Always ensure that maximum operating temperatures are not exceeded.
- Observing visible or invisible laser beams with the human eye directly, or indirectly, can cause permanent damage. Use a camera to observe the laser.
- No laser device should be used in any application or situation where life or property is at risk in event of device failure.
- Specifications are subject to change without notice. Ensure that you have the latest specification by contacting us prior to purchase or use of the product.

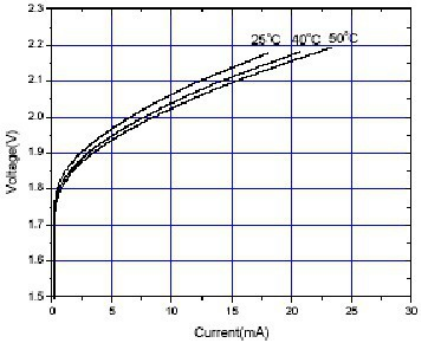
Tel:+43 1 586 52 43 -0 Fax: -44 office@roithner-laser.com 

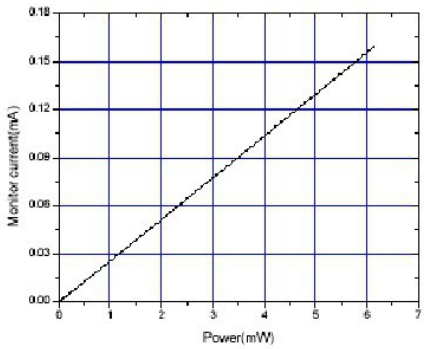
ROITHNER LASERTECHNIK GmbH
WIEDNER HAUPTSTRASSE 76 1040 VIENNA AUSTRIA
TEL. +43 1 586 52 43 -0, FAX. -44, OFFICE@ROITHNER-LASER.COM

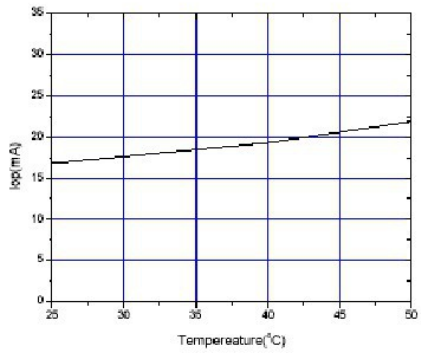
**AlGaInP** *Visible Laser Diode* **ADL-65052TL**

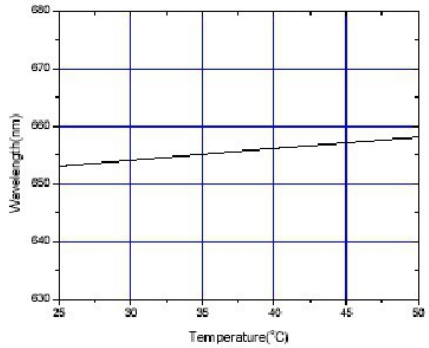
DATE : 2005/10/18 Ver 1.0

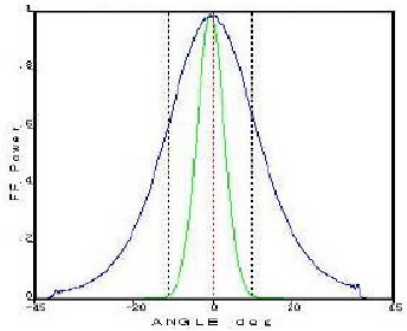


# MC33078, MC33079

## Low Noise Dual/Quad Operational Amplifiers

The MC33078/9 series is a family of high quality monolithic amplifiers employing Bipolar technology with innovative high performance concepts for quality audio and data signal processing applications. This family incorporates the use of high frequency PNP input transistors to produce amplifiers exhibiting low input voltage noise with high gain bandwidth product and slew rate. The all NPN output stage exhibits no deadband crossover distortion, large output voltage swing, excellent phase and gain margins, low open loop high frequency output impedance and symmetrical source and sink AC frequency performance.

The MC33078/9 family offers both dual and quad amplifier versions and is available in the plastic DIP and SOIC packages (P and D suffixes).

**Features**

- Dual Supply Operation: ±5.0 V to ±18 V
- Low Voltage Noise: 4.5 nV/$\sqrt{Hz}$
- Low Input Offset Voltage: 0.15 mV
- Low T.C. of Input Offset Voltage: 2.0 μV/°C
- Low Total Harmonic Distortion: 0.002%
- High Gain Bandwidth Product: 16 MHz
- High Slew Rate: 7.0 V/μs
- High Open Loop AC Gain: 800 @ 20 kHz
- Excellent Frequency Stability
- Large Output Voltage Swing: +14.1 V/ −14.6 V
- ESD Diodes Provided on the Inputs
- Pb−Free Packages are Available

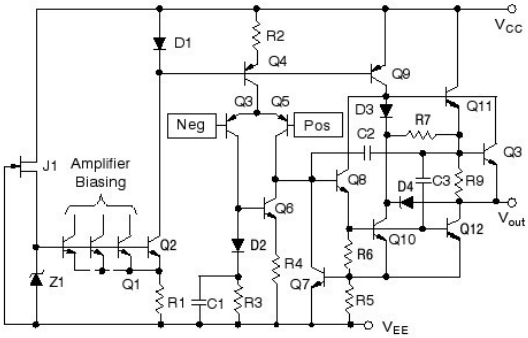


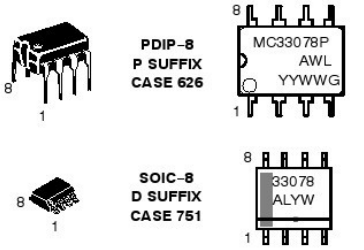

**Figure 1. Representative Schematic Diagram**
(Each Amplifier)

**ON Semiconductor®**

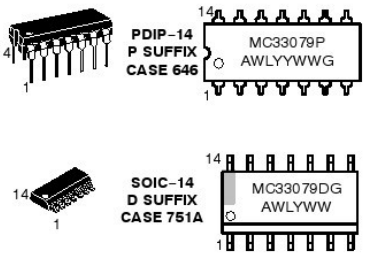
**http://onsemi.com**

**MARKING DIAGRAMS**

**DUAL**




**QUAD**




A = Assembly Location
WL, L = Wafer Lot
YY, Y = Year
WW, W = Work Week
G or ▪ = Pb−Free Package

**ORDERING INFORMATION**

See detailed ordering and shipping information in the package dimensions section on page 10 of this data sheet.

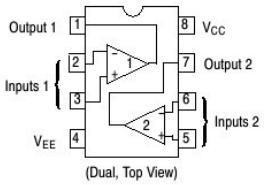

October, 2006 − Rev. 7

Publication Order Number:
MC33078/D

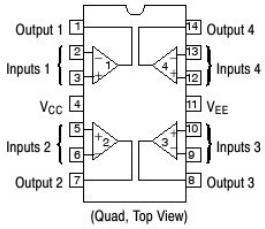
## MC33078, MC33079

### PIN CONNECTIONS

**DUAL**
**CASE 626/751**




**QUAD**
**CASE 646/751A**




### MAXIMUM RATINGS

| Rating | | Symbol | Value | Unit |
|---|---|---|---|---|
| Supply Voltage ($V_{CC}$ to $V_{EE}$) | | $V_S$ | +36 | V |
| Input Differential Voltage Range | | $V_{IDR}$ | Note 1 | V |
| Input Voltage Range | | $V_{IR}$ | Note 1 | V |
| Output Short Circuit Duration (Note 2) | | $t_{SC}$ | Indefinite | sec |
| Maximum Junction Temperature | | $T_J$ | +150 | °C |
| Storage Temperature | | $T_{stg}$ | −60 to +150 | °C |
| ESD Protection at any Pin | | $V_{esd}$ | | V |
| MC33078 | − Human Body Model | | 600 | |
| | − Machine Model | | 200 | |
| MC33079 | − Human Body Model | | 550 | |
| | − Machine Model | | 150 | |
| Maximum Power Dissipation | | $P_D$ | Note 2 | mW |
| Operating Temperature Range | | $T_A$ | −40 to +85 | °C |

Stresses exceeding Maximum Ratings may damage the device. Maximum Ratings are stress ratings only. Functional operation above the Recommended Operating Conditions is not implied. Extended exposure to stresses above the Recommended Operating Conditions may affect device reliability.

1. Either or both input voltages must not exceed the magnitude of $V_{CC}$ or $V_{EE}$.
2. Power dissipation must be considered to ensure maximum junction temperature ($T_J$) is not exceeded (see Figure 2).

**DC ELECTRICAL CHARACTERISTICS** ($V_{CC}$ = +15 V, $V_{EE}$ = −15 V, $T_A$ = 25°C, unless otherwise noted.)

| Characteristics | Symbol | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| Input Offset Voltage ($R_S$ = 10 Ω, $V_{CM}$ = 0 V, $V_O$ = 0 V) | $\|V_{IO}\|$ | | | | mV |
| (MC33078) $T_A$ = +25°C | | – | 0.15 | 2.0 | |
| $T_A$ = −40° to +85°C | | – | – | 3.0 | |
| (MC33079) $T_A$ = +25°C | | – | 0.15 | 2.5 | |
| $T_A$ = −40° to +85°C | | – | – | 3.5 | |
| Average Temperature Coefficient of Input Offset Voltage $R_S$ = 10 Ω, $V_{CM}$ = 0 V, $V_O$ = 0 V, $T_A$ = $T_{low}$ to $T_{high}$ | $\Delta V_{IO}/\Delta T$ | – | 2.0 | – | μV/°C |
| Input Bias Current ($V_{CM}$ = 0 V, $V_O$ = 0 V) | $I_{IB}$ | | | | nA |
| $T_A$ = +25°C | | – | 300 | 750 | |
| $T_A$ = −40° to +85°C | | – | – | 800 | |
| Input Offset Current ($V_{CM}$ = 0 V, $V_O$ = 0 V) | $I_{IO}$ | | | | nA |
| $T_A$ = +25°C | | – | 25 | 150 | |
| $T_A$ = −40° to +85°C | | – | – | 175 | |
| Common Mode Input Voltage Range ($\Delta V_{IO}$ = 5.0 mV, $V_O$ = 0 V) | $V_{ICR}$ | ±13 | ±14 | – | V |
| Large Signal Voltage Gain ($V_O$ = ±10 V, $R_L$ = 2.0 kΩ) | $A_{VOL}$ | | | | dB |
| $T_A$ = +25°C | | 90 | 110 | – | |
| $T_A$ = −40° to +85°C | | 85 | – | – | |
| Output Voltage Swing ($V_{ID}$ = ±1.0V) | | | | | V |
| $R_L$ = 600 Ω | $V_O$+ | – | +10.7 | – | |
| $R_L$ = 600 Ω | $V_O$− | – | −11.9 | – | |
| $R_L$ = 2.0 kΩ | $V_O$+ | +13.2 | +13.8 | – | |
| $R_L$ = 2.0 kΩ | $V_O$− | – | −13.7 | −13.2 | |
| $R_L$ = 10 kΩ | $V_O$+ | +13.5 | +14.1 | – | |
| $R_L$ = 10 kΩ | $V_O$− | – | −14.6 | −14 | |
| Common Mode Rejection ($V_{in}$ = ±13V) | CMR | 80 | 100 | – | dB |
| Power Supply Rejection (Note 3) | PSR | 80 | 105 | – | dB |
| $V_{CC}/V_{EE}$ = +15 V/ −15 V to +5.0 V/ −5.0 V | | | | | |
| Output Short Circuit Current ($V_{ID}$ = 1.0 V, Output to Ground) | $I_{SC}$ | | | | mA |
| Source | | +15 | +29 | – | |
| Sink | | −20 | −37 | – | |
| Power Supply Current ($V_O$ = 0 V, All Amplifiers) | $I_D$ | | | | mA |
| (MC33078) $T_A$ = +25°C | | – | 4.1 | 5.0 | |
| $T_A$ = −40° to +85°C | | – | – | 5.5 | |
| (MC33079) $T_A$ = +25°C | | – | 8.4 | 10 | |
| $T_A$ = −40° to +85°C | | – | – | 11 | |

3. Measured with $V_{CC}$ and $V_{EE}$ differentially varied simultaneously.

Burr-Brown Products
from Texas Instruments

# OPA355
# OPA2355
# OPA3355

SBOS195D – MARCH 2001 – REVISED JANUARY 2004

# 200MHz, CMOS OPERATIONAL AMPLIFIER WITH SHUTDOWN

## FEATURES

- UNITY-GAIN BANDWIDTH: 450MHz
- WIDE BANDWIDTH: 200MHz GBW
- HIGH SLEW RATE: 360V/μs
- LOW NOISE: 5.8nV/√Hz
- EXCELLENT VIDEO PERFORMANCE:
  DIFF GAIN: 0.02%, DIFF PHASE: 0.05°
  0.1dB GAIN FLATNESS: 75MHz
- INPUT RANGE INCLUDES GROUND
- RAIL-TO-RAIL OUTPUT (within 100mV)
- LOW INPUT BIAS CURRENT: 3pA
- LOW SHUTDOWN CURRENT: 3.4μA
- ENABLE/DISABLE TIME: 100ns/30ns
- THERMAL SHUTDOWN
- SINGLE-SUPPLY OPERATING RANGE: 2.5V to 5.5V
- *Micro*SIZE PACKAGES

## APPLICATIONS

- VIDEO PROCESSING
- ULTRASOUND
- OPTICAL NETWORKING, TUNABLE LASERS
- PHOTODIODE TRANSIMPEDANCE AMPS
- ACTIVE FILTERS
- HIGH-SPEED INTEGRATORS
- ANALOG-TO-DIGITAL (A/D) CONVERTER INPUT BUFFERS
- DIGITAL-TO-ANALOG (D/A) CONVERTER OUTPUT AMPLIFIERS
- BARCODE SCANNERS
- COMMUNICATIONS

## DESCRIPTION

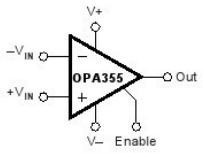
The OPA355 series high-speed, voltage-feedback CMOS operational amplifiers are designed for video and other applications requiring wide bandwidth. The OPA355 is unity-gain stable and can drive large output currents. In addition, the OPA355 has a digital shutdown (Enable) function. This feature provides power savings during idle periods and places the output in a high-impedance state to support output multiplexing. Differential gain is 0.02% and differential phase is 0.05°. Quiescent current is only 8.3mA per channel.

The OPA355 is optimized for operation on single or dual supplies as low as 2.5V (±1.25V) and up to 5.5V (±2.75V). Common-mode input range for the OPA355 extends 100mV below ground and up to 1.5V from V+. The output swing is within 100mV of the rails, supporting wide dynamic range.

The OPA355 series is available in single (SOT23-6 and SO-8), dual (MSOP-10), and triple (TSSOP-14 and SO-14) versions. Multichannel versions feature completely independent circuitry for lowest crosstalk and freedom from interaction. All are specified over the extended –40°C to +125°C range.

### OPA355 RELATED PRODUCTS

| FEATURES | PRODUCT |
|---|---|
| 200MHz, Rail-to-Rail Output, CMOS, No Shutdown | OPA356 |
| 38MHz, Rail-to-Rail Input/Output, CMOS | OPAx350 |
| 75MHz, Rail-to-Rail Output | OPAx631 |
| 150MHz, Rail-to-Rail Output | OPAx634 |
| Differential Input/Output, 3.3V Supply | THS412x |



## ABSOLUTE MAXIMUM RATINGS(1)

| | |
|---|---|
| Supply Voltage, V+ to V– | 7.5V |
| Signal Input Terminals, Voltage(2) | (V–) – 0.5V to (V+) + 0.5V |
| Current(2) | 10mA |
| Enable Input | (V–) – 0.5V to (V+) + 0.5V |
| Output Short-Circuit(3) | Continuous |
| Operating Temperature | –55°C to +150°C |
| Storage Temperature | –65°C to +150°C |
| Junction Temperature | +160°C |
| Lead Temperature (soldering, 10s) | +300°C |

NOTES: (1) Stresses above these ratings may cause permanent damage. Exposure to absolute maximum conditions for extended periods may degrade device reliability. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those specified is not implied. (2) Input terminals are diode-clamped to the power-supply rails. Input signals that can swing more than 0.5V beyond the supply rails should be current limited to 10mA or less. (3) Short-circuit to ground, one amplifier per package.

## ELECTROSTATIC DISCHARGE SENSITIVITY

This integrated circuit can be damaged by ESD. Texas Instruments recommends that all integrated circuits be handled with appropriate precautions. Failure to observe proper handling and installation procedures can cause damage.

ESD damage can range from subtle performance degradation to complete device failure. Precision integrated circuits may be more susceptible to damage because very small parametric changes could cause the device not to meet its published specifications.

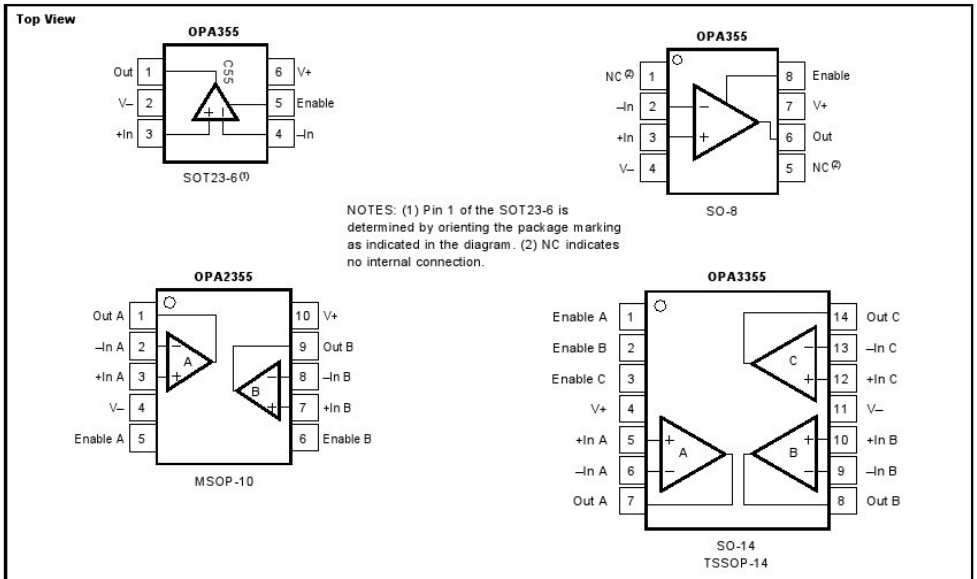
## PACKAGE/ORDERING INFORMATION(1)

| PRODUCT | PACKAGE-LEAD | PACKAGE MARKING |
|---|---|---|
| OPA355 | SOT23-6 | C55 |
| " | " | " |
| OPA355 | SO-8 | OPA355UA |
| " | " | " |
| OPA2355 | MSOP-10 | D55 |
| " | " | " |
| OPA3355 | TSSOP-14 | OPA3355EA |
| " | " | " |
| OPA3355 | SO-14 | OPA3355UA |
| " | " | " |

NOTE: (1) For the most current package and ordering information, see the Package Option Addendum located at the end of this data sheet.

## PIN CONFIGURATIONS

Top View



## ELECTRICAL CHARACTERISTICS: $V_S$ = +2.7V to +5.5V Single-Supply

**Boldface** limits apply over the specified temperature range, $T_A$ = –40°C to +125°C.

At $T_A$ = +25°C, $R_F$ = 604Ω, $R_L$ = 150Ω, and connected to $V_S$/2, unless otherwise noted.

| PARAMETER | | CONDITION | OPA355 OPA2355 OPA3355 MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|
| **OFFSET VOLTAGE** | | | | | | |
| Input Offset Voltage | $V_{OS}$ | $V_S$ = +5V | | ±2 | ±9 | mV |
| | | **Specified Temperature Range** | | | **±15** | **mV** |
| **vs Temperature** | **$dV_{OS}/dT$** | **Specified Temperature Range** | | **±7** | | **µV/°C** |
| vs Power Supply | PSRR | $V_S$ = +2.7V to +5.5V, $V_{CM}$ = $V_S$/2 – 0.15V | | ±80 | ±350 | µV/V |
| **INPUT BIAS CURRENT** | | | | | | |
| Input Bias Current | $I_B$ | | | 3 | ±50 | pA |
| Input Offset Current | $I_{OS}$ | | | ±1 | ±50 | pA |
| **NOISE** | | | | | | |
| Input Noise Voltage Density | $e_n$ | f = 1MHz | | 5.8 | | $nV/\sqrt{Hz}$ |
| Current Noise Density | $i_n$ | f = 1MHz | | 50 | | $fA/\sqrt{Hz}$ |
| **INPUT VOLTAGE RANGE** | | | | | | |
| Common-Mode Voltage Range | $V_{CM}$ | | (V–) – 0.1 | | (V+) – 1.5 | V |
| Common-Mode Rejection Ratio | CMRR | $V_S$ = +5.5V, –0.1V < $V_{CM}$ < +4.0V | 66 | 80 | | dB |
| | | **Specified Temperature Range** | **66** | | | **dB** |
| **INPUT IMPEDANCE** | | | | | | |
| Differential | | | | $10^{13}$ \|\| 1.5 | | Ω \|\| pF |
| Common-Mode | | | | $10^{13}$ \|\| 1.5 | | Ω \|\| pF |
| **OPEN-LOOP GAIN** | | $V_S$ = +5V, 0.3V < $V_O$ < 4.7V | 84 | 92 | | dB |
| **OPA355** | | **$V_S$ = +5V, 0.3V < $V_O$ < 4.7V** | **80** | | | **dB** |
| **OPA2355, OPA3355** | | **$V_S$ = +5V, 0.4V < $V_O$ < 4.6V** | **80** | | | **dB** |
| **FREQUENCY RESPONSE** | | | | | | |
| Small-Signal Bandwidth | $f_{-3dB}$ | G = +1, $V_O$ = 100mVp-p, $R_F$ = 0Ω | | 450 | | MHz |
| | $f_{-3dB}$ | G = +2, $V_O$ = 100mVp-p, $R_L$ = 50Ω | | 100 | | MHz |
| | $f_{-3dB}$ | G = +2, $V_O$ = 100mVp-p, $R_L$ = 150Ω | | 170 | | MHz |
| | $f_{-3dB}$ | G = +2, $V_O$ = 100mVp-p, $R_L$ = 1kΩ | | 200 | | MHz |
| Gain-Bandwidth Product | GBW | G = +10, $R_L$ = 1kΩ | | 200 | | MHz |
| Bandwidth for 0.1dB Gain Flatness | $f_{0.1dB}$ | G = +2, $V_O$ = 100mVp-p, $R_F$ = 560Ω | | 75 | | MHz |
| Slew Rate | SR | $V_S$ = +5V, G = +2, 4V Output Step | | 300/–360 | | V/µs |
| Rise-and-Fall Time | | G = +2, $V_O$ = 200mVp-p, 10% to 90% | | 2.4 | | ns |
| | | G = +2, $V_O$ = 2Vp-p, 10% to 90% | | 8 | | ns |
| Settling Time, 0.1% | | $V_S$ = +5V, G = +2, 2V Output Step | | 30 | | ns |
| 0.01% | | $V_S$ = +5V, G = +2, 2V Output Step | | 120 | | ns |
| Overload Recovery Time | | $V_{IN}$ • Gain = $V_S$ | | 8 | | ns |
| Harmonic Distortion | | | | | | |
| 2nd-Harmonic | | G = +2, f = 1MHz, $V_O$ = 2Vp-p, $R_L$ = 200Ω | | –81 | | dBc |
| 3rd-Harmonic | | G = +2, f = 1MHz, $V_O$ = 2Vp-p, $R_L$ = 200Ω | | –93 | | dBc |
| Differential Gain Error | | NTSC, $R_L$ = 150Ω | | 0.02 | | % |
| Differential Phase Error | | NTSC, $R_L$ = 150Ω | | 0.05 | | degrees |
| Channel-to-Channel Crosstalk | OPA2355 | f = 5MHz | | –90 | | dB |
| | OPA3355 | f = 5MHz | | –70 | | dB |
| **OUTPUT** | | | | | | |
| Voltage Output Swing from Rail | | $V_S$ = +5V, $R_L$ = 150Ω, $A_{OL}$ > 84dB | | 0.2 | 0.3 | V |
| Voltage Output Swing from Rail | | $V_S$ = +5V, $R_L$ = 1kΩ | | 0.1 | | V |
| Output Current, Continuous(1) | $I_O$ | | | ±60 | | mA |
| Output Current, Peak(1) | $I_O$ | $V_S$ = +5V | | ±100 | | mA |
| Output Current, Peak(1) | $I_O$ | $V_S$ = +3V | | ±80 | | mA |
| Closed-Loop Output Impedance | | f < 100kHz | | 0.02 | | Ω |
| **POWER SUPPLY** | | | | | | |
| Specified Voltage Range | $V_S$ | | 2.7 | | 5.5 | V |
| Operating Voltage Range | | | | 2.5 to 5.5 | | V |
| Quiescent Current (per amplifier) | $I_Q$ | $V_S$ = +5V, Enabled, $I_O$ = 0 | | 8.3 | 11 | mA |
| | | **Specified Temperature Range** | | | **14** | **mA** |

NOTES: (1) See typical characteristic *Output Voltage Swing vs Output Current*. (2) Logic LOW and HIGH levels are CMOS logic compatible. They are referenced to V–.

## ELECTRICAL CHARACTERISTICS: $V_S$ = +2.7V to +5.5V Single-Supply (Cont.)

**Boldface** limits apply over the specified temperature range, $T_A$ = –40°C to +125°C.

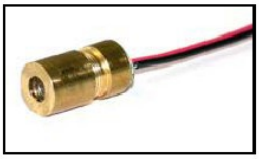
At $T_A$ = +25°C, $R_F$ = 604Ω, $R_L$ = 150Ω, and connected to $V_S$/2, unless otherwise noted.

| PARAMETER | CONDITION | OPA355 OPA2355 OPA3355 MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|
| **SHUTDOWN** | | | | | |
| Disabled (Logic-LOW Threshold)(2) | | | | 0.8 | V |
| Enabled (Logic-HIGH Threshold)(2) | | 2 | | | V |
| Enable Time | | | 100 | | ns |
| Disable Time | | | 30 | | ns |
| Shutdown Current (per amplifier) | $V_S$ = +5V, Disabled | | 3.4 | 6 | µA |
| **THERMAL SHUTDOWN** | | | | | |
| Junction Temperature | | | | | |
| Shutdown | | | 160 | | °C |
| Reset from Shutdown | | | 140 | | °C |
| **TEMPERATURE RANGE** | | | | | |
| Specified Range | | –40 | | 125 | °C |
| Operating Range | | –55 | | 150 | °C |
| Storage Range | | –65 | | 150 | °C |
| Thermal Resistance $\theta_{JA}$ | | | | | °C/W |
| SOT-23-6, MSOP-10 | | | 150 | | °C/W |
| SO-8 | | | 125 | | °C/W |
| SO-14, TSSOP-14 | | | 100 | | °C/W |

NOTES: (1) See typical characteristic *Output Voltage Swing vs Output Current*. (2) Logic LOW and HIGH levels are CMOS logic compatible. They are referenced to V–.

ROITHNER LASERTECHNIK GmbH
WIEDNER HAUPTSTRASSE 76 1040 VIENNA AUSTRIA
TEL. +43 1 586 52 43 -0, FAX. -44, OFFICE@ROITHNER-LASER.COM

# APCD-650-02-C3



## TECHNICAL DATA

## Red diode laser module

**APCD-650-02 is a multi purpose small size red diode laser module featuring a fixfocus acrylic lens, with integrated APC circuitry for long time stable operation**

### *Features*

- **Small size (Ø 6.2 x 11.0 mm)**
- **Focussable acryl lens**
- **APC (auto power control) IC integrated**
- **Low current consumption**
- **Surge current protection**
- **Excellent beam quality**

### *Absolute Maximum Ratings ($T_C$=25°C)*

| Item | Symbol | Value | Unit |
|---|---|---|---|
| Power Supply Voltage | $V_{CC}$ | 3.3 | V |
| Output Power | $P_O$ | <3 | mW |
| Operating Temperature | $T_C$ | 0 ... +40 | °C |
| Storage Temperature | $T_{stg}$ | 0 ... +60 | °C |

### *Specifications ($T_C$=25°C, $P_O$<3mW, $V_{CC}$=3V)*

| | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|
| **Optical** | | | | |
| Center Wavelength $\lambda_c$ | - | 655 | - | nm |
| Output Power | - | - | 3.0 | mW |
| Divergence angle | | 1.1 | | mrad |
| Output Aparture | | 2.4 | | mm |
| Beam Size at 10M | | 10 | | mm |
| **Electrical** | | | | |
| Current draw | - | - | 35 | mA |
| Supply voltage | 2.5 | - | 3.3 | V |
| **General** | | | | |
| Body | | Brass | | |
| Dimensions | | 6.2 x 11.0 | | mm |
| Lens | | Acryl | | |
| Mean time to failure (MTTF) | | >10000 | | h |

The above specifications are for reference purpose only and subjected to change without prior notice

ROITHNER LASERTECHNIK GmbH
WIEDNER HAUPTSTRASSE 76 1040 VIENNA AUSTRIA
TEL. +43 1 586 52 43 -0, FAX. -44, OFFICE@ROITHNER-LASER.COM

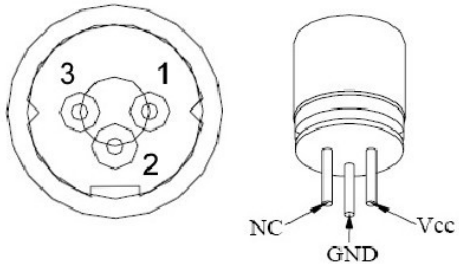
***Electrical Connection :***

Heat sink stand (–)

**Pin 1** : Vcc (RED lead)

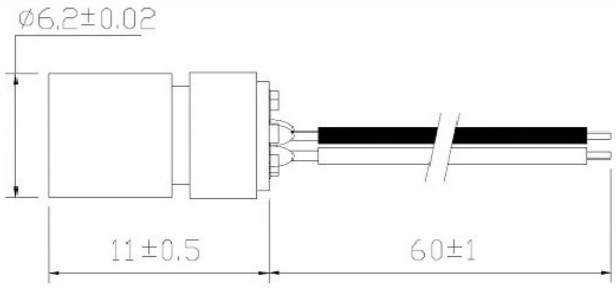
**Pin 2** : GND (BLACK lead)

**Pin 3** : NC ( No external connection )




***Outline Dimension :***

*Unit: mm*




***Cautions***

1. Do not operate the device above the macimum rating condition, even momentarily. It may cause unexpected permanent damage to the device.
2. Semiconductor laser device is very sensitive to electrostatic discharge. High voltage spike current may change the characteristics of the device, or malfunction at any time during its sercice periode. Therefor, proper measures for precenting electrostatic discharge are strongly recommended.
3. Do not look into the laser beam directly with the naked eyes. The laser beam may cause severe damage to human eyes.