UNIVERSITA' DEGLI STUDI DI CATANIA

FACOLTA' DI INGEGNERIA

Corso di Laurea di I livello in Ingegneria Informatica

TESI DI LAUREA

GESTIONE SOFTWARE DI UN SISTEMA DAC/ADC ED ORGANIZZAZIONE HARDWARE

Candidato:

Fortunato Oliveri
616/001666

Relatore:

Prof. Ing. Michele Malgeri

Correlatore:

Dott. Paolo Finocchiaro

Anno Accademico 2011 / 2012

Sommario

1. Introduzione

Lo scopo di questo lavoro è la creazione di un componente hardware/software in grado di controllare il sistema di monitoraggio, in tempo reale, di rifiuti radioattivi, nell'ambito del progetto DMNR (Detector Mesh for Nuclear Repositories), condotto dall' INFN[1] in collaborazione con Ansaldo Nucleare e in fase di installazione preliminare presso il deposito Sogin[2] di Sessa Aurunca (CE).

In ambito internazionale si definisce rifiuto radioattivo qualsiasi sostanza (in forma solida, liquida o gassosa) che contenga o sia contaminata da radionuclidi, a concentrazioni o livelli di radioattività superiori alle quantità permesse definite dalle autorità competenti. [IAEA 10]
Potremmo pensare a una classificazione dei rifiuti in base al tempo di decadimento della pericolosità, per l'ambiente e per l'essere umano. Secondo questa scala, quindi, possiamo distinguere il metodo di trattamento e di stoccaggio, e scegliere le procedure migliori e più sicure.
Le scorie nucleari [ANPA26], sono dunque un tipo particolare di rifiuto radioattivo, provenienti dal funzionamento o dallo smantellamento di reattori nucleari, esse oggi sono concentrate in particolari depositi, all'interno di fusti che ne dovrebbero garantire la messa sicurezza fino al totale decadimento della carica radioattiva.

[1] L'Istituto Nazionale di Fisica Nucleare è l'ente italiano che promuove, coordina ed effettua la ricerca scientifica nel campo della fisica nucleare, subnucleare e astro particellare, nonché lo sviluppo tecnologico necessario alle attività in tali settori. [INFN]

[2] Società Gestione Impianti Nucleari, S.p.A. italiana con il compito di controllare, smantellare, decontaminare e gestire i rifiuti radioattivi.

Le procedure odierne di monitoraggio prevedono controlli periodici, ad opera di operatori specializzati oppure di robots, dotati di strumenti di misura adeguati. Risultano evidenti le principali problematiche da affrontare, cioè quelle di un monitoraggio continuo e accurato delle scorie radioattive, che non coinvolga direttamente sul campo esseri umani e la convenienza economica del sistema.
Per questo si presenta la necessità di avere un sistema automatico di monitoraggio real-time, che non preveda l'intervento diretto di un essere umano, abbastanza accurato da essere capace di rilevare anche la minima fuga di radiazioni ed anche semplice ed economico in modo da poter essere applicato su larga scala con la minima spesa.

In questo senso si muove il progetto DMNR [DMNR], che sfrutta gli studi su un nuovo tipo di rivelatore di radiazioni ionizzanti, con caratteristiche simili ai contatori Geiger-Muller (ma a differenza di questi è notevolmente più economico e versatile), questo è composto da semplici fibre ottiche scintillanti che emettono flebili impulsi di luce quando vengono colpite da radiazioni gamma, nel nostro caso causate da una fuoriuscita di materiale radioattivo. Ai capi di questi fasci di fibre, che avvolgono un'intera pila di fusti secondo una particolare geometria, vengono posti due fotosensori al silicio (SiPM) di ultima generazione, capaci di rivelare i deboli segnali luminosi prodotti dall'interazione delle radiazioni con le fibre, e convertirli in impulsi elettrici veloci. I segnali generati dai SiPM hanno bisogno di una post-amplificazione con dei particolari amplificatori a larga banda, dunque verranno sottoposti in ingresso a dei discriminatori in cui possono essere modulate le soglie di

discriminazione, che quindi consentono di ottenere soltanto i valori voluti.
Nell'ambito di DMNR è stata sviluppata la parte elettronica e di front-end che permette l'acquisizione e la manipolazione di questi segnali infatti, fasi essenziali sono quelle del conteggio degli impulsi prodotti dal SiPM e della modulazione delle soglie dei discriminatori in modo da permettere un cambiamento del fattore di scala ed estendere così il campo di misura. Tutti i dati acquisiti in fine verranno automaticamente impacchettati ed inviati da una semplice centralina a una console remota, per l'analisi e l'eventuale generazione di allarmi che richiederanno l'intervento di manutenzione.

Un progetto come DMNR, che punta a risolvere il problema del monitoraggio delle scorie radioattive secondo le particolari specifiche descritte in precedenza, tocca tanti aspetti interdisciplinari; si passa dalla fisica dei materiali che riguarda le fibre ottiche, alla micro elettronica dei SiPM, alla parte riguardante l'elettronica di front-end e per finire la parte informatica, che permette il trattamento dei dati e le varie regolazioni dei circuiti.

L'aspetto di cui mi sono occupato è quello di permettere interazioni tra hardware e software. Sviluppando un componente software che punti a rendere trasparente all'operatore l'intera architettura, ad esso è messa a disposizione un'interfaccia cui dare in ingresso soltanto dei valori voluti, oppure da cui leggere dei risultati cercati.
Nel prossimo capitolo intraprenderò una capillare descrizione dell'intero progetto, trattando della completa architettura di DMNR e caratterizzando uno ad uno tutti gli strumenti utilizzati per

conseguire l'obiettivo della creazione di un prototipo funzionante, in rispetto delle specifiche di progettazione.

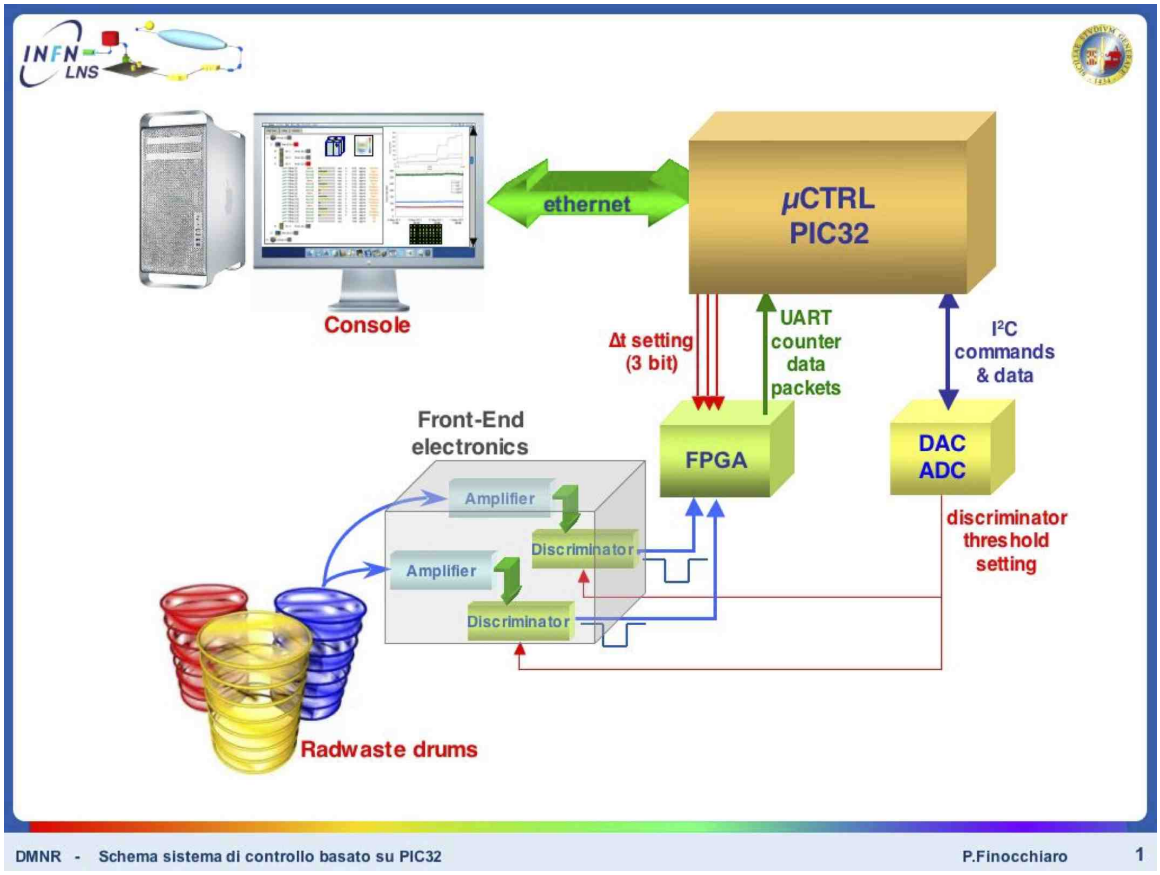


Figura 1 Architettura generale di DMNR

Lo schema di massima appena illustrato permette di comprendere facilmente i blocchi funzionali di DMNR. Le frecce indicano il verso dei flussi di dati e naturalmente è stato omessa, per semplicità, la parte dei rivelatori che devono stare su ogni singolo fusto. Nei prossimi capitoli descriverò uno ad uno questi componenti, riferendomi a questo schema, che è proprio quello per la creazione del prototipo finale.

2. Architettura e componenti hardware

L'esigenza di controllo da cui sono partito è quella che riguarda i discriminatori. Questi sono dei circuiti elettronici che si comportano come un trigger, hanno in ingresso i segnali amplificati proveniente dai SiPM, e accettano l'impostazione di un valore di tensione di discriminazione, tale che, superata questa soglia sarà generato un impulso che verrà conteggiato tramite una logica particolare sviluppata con un FPGA. In seconda battuta si presenta la necessità di leggere il valore di soglia impostato, anche quest'operazione è stata prevista durante la stesura del codice.

È stato necessario l'inserimento dello stadio di elettronica di front-end, tra i SiPM e il contatore di impulsi, perché la sezione di rivelazione composta da fasci di fibre ottiche e fotosensori produce quantità eccessive di rumore, inteso nel senso di rilevazioni inutili allo scopo, dovute all'elevata sensibilità dei SiPM oppure alla presenza di radiazioni gamma provenienti da altre fonti (ad esempio i raggi cosmici). Per cui, il compito dell'elettronica di front-end è proprio quello di filtrare ciò che non serve, e trasmettere soltanto le informazioni utili.
Abbiamo pensato anche alla situazione in cui, ad esempio, si ha una perdita da un fusto di scorie, quindi partendo da uno stato di misurazione nulla, si passerà a un allarme, allora reimpostando correttamente il valore di soglia del discriminatore, si ha la possibilità di quantizzare l'entità del danno, in base ad opportuni valori di riferimento. Con questa procedura si offre la possibilità di escludere eventuali allarmi inutili, e coordinare le operazioni di manutenzione nel migliore modo possibile.

Le scelte fatte dai progettisti dell'hardware per adempiere al meglio alle necessità progettuali (definite tra i team impegnati nello sviluppo), sono state quelle di usare due semplici tipi di componenti elettronici, cioè un convertitore digitale-analogico (DAC) e uno analogico-digitale (ADC), più un microcontrollore che coordina tutte le varie sezioni del sistema. Il primo non è altro che un integrato pilotabile tramite un bus I^2C [NFP] capace di produrre in uscita un valore di tensione richiesto con un'accuratezza abbastanza elevata, nell'ordine dei mV. Questo valore sarà poi usato come soglia di discriminazione del discriminatore. Il secondo componente invece, fa l'esatto contrario, cioè legge un valore analogico di tensione e lo trasmette secondo una codifica digitale. Questo integrato avendo un ruolo meno importante rispetto al primo, è stato bersaglio di una variazione in corso d'opera. Infatti inizialmente si era pensato di usare un ADC esterno, prodotto dalla stessa casa produttrice del DAC, accuratissimo e versatile, ma in un'ottica di abbassamento dei costi e di semplificazione della circuiteria si è scelto di usare il convertitore analogico-digitale presente all'interno del microcontrollore, che pur non essendo accurato come l'ADC esterno, riesce comunque a svolgere il suo compito di "lettura", e permette la verifica di ogni valore di soglia impostato per ogni singolo discriminatore. Infine il componente principale per fare funzionare tutto il sistema è un microcontrollore, in cui convergono e da cui partono tutti i flussi d'informazione da e per la sezione di rivelazione, e da e per la console remota. Questo è un processore cui vengono affidati i compiti di elaborazione e gestione delle varie periferiche che vi sono inserite.

Naturalmente nella versione finale del progetto tutti i componenti elettronici saranno collocati all'interno di un'unica mainboard, che prevedrà tutte le connessioni necessarie, ma nella fase di sviluppo si

è proceduto lavorando su blocchi singoli. Infatti, ogni casa produttrice di parti hardware fornisce agli sviluppatori particolari schede di sviluppo (dev. board), su cui eseguire i test preliminari di realizzabilità delle specifiche, queste permettono un veloce accesso ai pin d'ingresso e uscita, e spesso sono corredate da speciali ambienti di sviluppo, indispensabili per la scrittura del codice necessario a fare funzionare il tutto.

2.1 Problematiche da risolvere

Durante tutto il periodo di sviluppo di un progetto complesso come DMNR si sono dovute affrontare svariate problematiche, soprattutto dal punto di vista dell'hardware da utilizzare, che deve soddisfare particolari necessità, ma anche, una volta definita l'architettura di massima, quelle dell'implementazione del software che deve, in un certo senso, unire tutti i vari componenti per ottenere il risultato voluto.

Le principali problematiche affrontate, quindi sono state prettamente legate alla comprensione del funzionamento del microcontrollore, dei decodificatori digitale-analogico ed analogico-digitale e del bus I^2C. I vari componenti usati, anche se prodotti da società differenti, utilizzano standard ben definiti per la comunicazione. Allora il problema della gestione delle interazioni tra le varie interfacce è stato risolto applicando alla lettera le specifiche degli standard usati. Questo mi ha portato ad uno studio approfondito delle varie architetture, l'esclusione dei conflitti tra le periferiche, la corretta gestione degli interrupt del processore. Per ogni dev. board c'è stato bisogno di particolari settaggi, per renderla

il più possibile fruibile al mio scopo. Si sono dovuti costruire i bus necessari, rispettando le specifiche dello standard I^2C. Un'altra importante specifica era quella di mantenere la risoluzione massima dei convertitori, e rendere il tutto il più veloce possibile, in modo da non occupare troppo allungo risorse preziose (in termini di tempo di CPU) per il conteggio degli impulsi. Ho dato particolare peso anche alla leggibilità del codice, scrivendo funzioni parametriche, e riunendo tutti i parametri legati all'hardware in un'unica pagina dedicata ai settaggi. Così facendo, anche cambiando tipi di convertitori, il codice potrà essere riutilizzato. Tutti questi contributi hanno portato alla creazione di un prototipo funzionante, nel rispetto delle specifiche.

DMNR, partito col progetto strategico INFN-Energia, e giunto ormai ad un accordo con SOGIN di durata biennale, si lancia, sin dall'inizio, nella creazione di uno strumento tecnologico innovativo per il monitoraggio real-time di scorie nucleari, il tutto mantenendo un occhio di riguardo all'aspetto economico, per tenere bassi il più possibile i costi di gestione e messa in opera.

In questo capitolo descriverò dettagliatamente l'architettura generale del progetto DMNR, trattando per prima la parte relativa al controllo hardware, in seguito descriverò l'ambiente di sviluppo ed il software creato per adempiere al mio scopo. Saranno affrontate in modo capillare tutte le tematiche che hanno fatto parte del mio lavoro di tesi, cercando di mantenere chiara la trattazione di argomenti anche abbastanza ostici, particolarità dovuta all'inclinazione sperimentale del progetto, e alle caratteristiche dei componenti impiegati.

Tutti i componenti hardware usati, sono stati selezionati dai responsabili del laboratorio di elettronica del Laboratorio Nazionale del Sud, in base alle specifiche da soddisfare. I produttori scelti sono tutti leader nel settore, e forniscono ogni tipo di supporto per ogni evenienza.
Intraprenderò la descrizione del microcontrollore PIC32 [PFDS] commercializzato da Microchip[3], del bus I²C[4], del DAC e dell'ADC prodotti da AnalogDevices[5]. Per tutta la trattazione userò materiale ricavato dai vari datasheet forniti dai diversi produttori.
Comincerò proprio dal più importante.

2.2 Il microcontrollore PIC32MX795F512L

Il nucleo centrale di tutta la struttura di DMNR è il microcontrollore, questo dovrà essere abbastanza potente da poter gestire tutti i compiti che gli attribuiremo, in qualsiasi momento siano richiesti. Per questo è stato scelto un PIC32 della Microchip Technology, una casa produttrice di semiconduttori, statunitense, leader nel mercato. Il modello specifico è il PIC32MX795F512L;
Le varie parti del nome ci indicano le caratteristiche essenziali del microchip.

- PIC32 MX: si riferisce al fatto che il componente usato è fornito di un'architettura a 32 bit, quindi simile a quella dei processori dei vecchi personal computer, ma naturalmente con un set d'istruzioni ridotto, RISC, e con un core di tipo MCU (Micro Controller Unit).

- 795: indica la famiglia di microcontrollori cui appartiene.

[3] http://www.microchip.com
[4] http://www.nxp.com/documents/other/39340011.pdf
[5] http://www.analog.com/en/index.html

- F: significa che il chip è dotato di una memoria Flash per il caricamento del programma da eseguire, ciò significa che è possibile scrivere e riscrivere questa memoria senza particolari complicazioni, ed in oltre i dati scritti rimarranno in memoria anche senza alimentazione elettrica esterna, questo semplifica senza dubbio la circuiteria e l'uso generale del componente.

- 512: indica la dimensione della memoria discussa nel punto precedente, in questo caso ci dice che la memoria dedicata al caricamento del programma è di 512 KB, oltre a questa si hanno anche 12 KB di spazio su memoria flash riservati per il boot, cioè l'avvio del codice da eseguire.

- L: informa che il chip usato è la versione con 100 Pin.

In oltre possiamo avere anche informazioni riguardo la velocita del clock principale, che per questa famiglia è di 80 MHz, il range di temperatura in cui può operare il microcontrollore, che va da -40° C a +85° C, e sul package di fabbricazione che nel nostro caso è TQFP (Thin Quad Flat Pack) caratterizzato da un basso spessore.
In oltre questo modello di microcontrollore ha a disposizione: un'interfaccia USB, un'Ethernet, sei interfacce UART, cinque I^2C e 16 canali per un Convertitore Analogico/Digitale con 10 Bit di risoluzione ed 1 Msps (Million Samples Per Second) come velocità. Tutte queste interfacce (sono solamente quelle che servono per il nostro progetto) possono avere dei pin in condivisione, per questo si è intrapreso uno studio attento riguardo ai conflitti tra le periferiche, in modo da evitare le interferenze. Si è visto che è possibile tranquillamente supplire alle nostre necessità d'uso.

Il su citato chip, con tutto il suo carico di funzioni è montato su di una cosiddetta Ethernet Starter Kit, una scheda utile soltanto in ambito dello sviluppo, che sarà descritta di seguito.

2.3 Ethernet Starter Kit

La suite hardware di sviluppo usata durante tutto il periodo di stage è un ottimo strumento che mi ha consentito di semplificare la programmazione e l'uso del microcontrollore. L'Ethernet Starter Kit [PESK], è una scheda equipaggiata da componenti essenziali per lo sviluppo del software, assolutamente necessari al nostro scopo sono un connettore RJ45, un USB per il bebug da PC, una serie di pin che forniscano tutti gli input e output necessari e naturalmente il microcontrollore che coordini il tutto. Sono stai molto usati anche alcuni led programmabili presenti sul nostro Starter Kit, che ci hanno consentito di avere un responso veloce delle operazioni effettuate. L'Ethernet Starter Kit, con lungimiranza, è stato pensato ulteriormente espandibile, infatti può essere perfettamente abbinato ad un ulteriore scheda che descriverò a seguire.

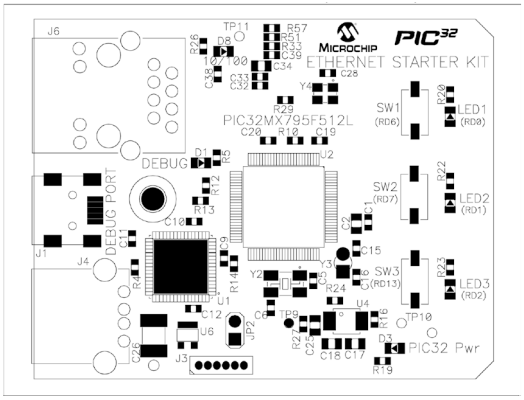


Figura 3 Ethernet Starter Kit front

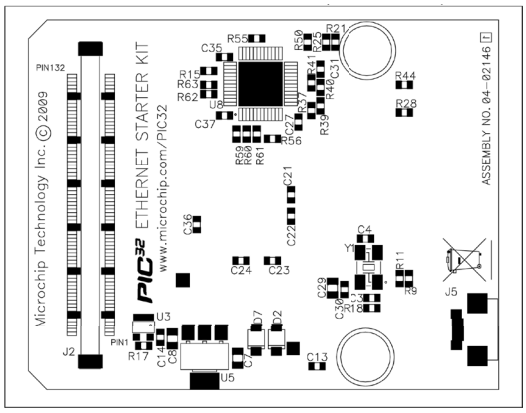


Figura 2 Ethernet Starter Kit back

2.4 Multimedia Expansion Board

Sulla parte posteriore della scheda è presente un connettore particolare a 132 pin, questo serve ad estendere ulteriormente l'ambiente hardware di sviluppo, consentendo l'uso di una Multimedia Expansion Board [MEB], che rende assolutamente completo l'intero sistema. Tra tutte le possibili estensioni a disposizione, quali un display touch screen, microfono, accelerometro, un termometro, uno slot microSD ed un modulo wireless 802.11, mi è stato molto utile, e in effetti ho molto usato, il connettore di espansione per l'I/O.

Questo connettore non fa altro che riportare all'esterno, e quindi rende comodamente accessibili, alcuni pin del microcontrollore, che è saldato sullo Starter Kit, cosa che quindi sicuramente non ne agevola l'accesso.

Quelli da me usati sono:

- Il pin 3, usato per la linea SCL del bus I^2C.
- Il pin 5, usato per la linea SDA del bus I^2C.
- Il pin 19 che mi è servito come ingresso per la lettura della tensione generata dal DAC.
- Il pin 28 che è il ground.

In altri termini questo connettore per l'espansione dell'I/O mi ha permesso di poter comandare tramite bus I^2C un DAC esterno, permettendo di generare una tensione cercata e di potere leggere questa tensione usando l'ADC integrato nel microcontrollore. Lo schema usato infatti prevede il collegamento del sistema Ethernet Starter Kit Multimedia Expansion Board con il DAC di Analog Devices, attraverso un bus I^2C. E quindi il collegamento del piedino di monitor del DAC, da cui siamo capaci di leggere il valore di tensione di un canale, con il blocco Ethernet Starter Kit -

Multimedia Expansion Board, così facendo permetteremo al modulo ADC presente all'interno del microcontrollore di poter verificare la tensione cercata.

La scelta di usare un bus di tipo I²C garantisce semplicità nell'elettronica del bus e si adatta perfettamente al tipo di comunicazione tra DAC e microcontrollore, infatti si riesce a inviare e ricevere piccole quantità di dati anche abbastanza velocemente, mantenendo l'affidabilità della comunicazione. Il tutto garantito dalla presenza di un forte standard, quello dell' I²C, di proprietà di Philips, ed ancora oggi in costante evoluzione.

Adesso passerò alla descrizione del bus I²C, quindi la parte che ci consente di dialogare con il DAC esterno prodotto da Analog Devices.

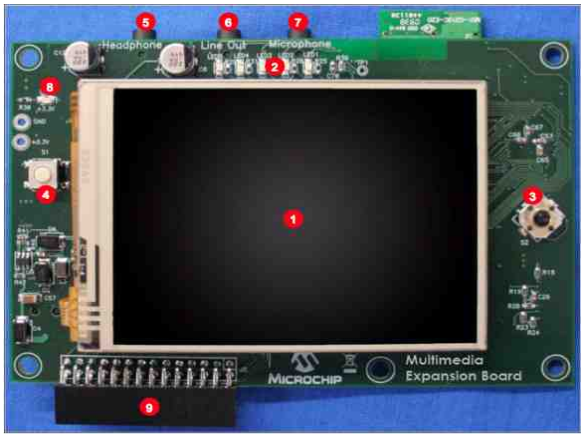


Figura 5 Multimedia Expansion Board front

Figura 4 Multimedia Expansion Board back

2.5 Il Bus I²C

Il tipo di collegamento scelto per la comunicazione tra microcontrollore e DAC è un bus di tipo I²C. Si tratta di un sistema di comunicazione seriale, caratterizzato dall'uso di due linee, SDA ed SCL. Lo standard I²C prevede un'architettura Master/Slave, permettendo anche situazioni con più di un master. Nel caso del mio progetto la figura dell'unico Master è intrapresa dal

microcontrollore, che quindi comunica con l'unico Slave, il DAC, attraverso il bus. Il Master è quindi capace di inviare dati allo Slave secondo una precisa codifica comprensibile a destinazione. Nella procedura da me create vengono previsti solo comandi di scrittura sul DAC, quindi possiamo pensare ad una comunicazione prevalentemente unidirezionale, escludendo i segnali di acknowledge che sono inviati dal DAC al microcontrollore. È quindi chiaro che il bus I²C serve prettamente per la comunicazione con un dispositivo, il DAC, capace solo di ascoltare, e quindi solamente di essere programmato per lo scopo richiesto.
Questo standard si è rivelato adatto al nostro scopo perché, anche non fornendo altissime velocità di trasmissione, consente di mantenere l'elettronica relativa al bus molto semplice e si ha sempre un meccanismo di verifica della comunicazione tramite ack, che ci assicura l'affidabilità della stessa. Lo standard I²C, inventato da Philips, ha delle particolari specifiche da rispettare, che ne garantiscono il funzionamento anche con hardware non prodotto da Philips. Un aspetto importante cui prestare attenzione sono le resistenze di Pull-Up, queste devono essere inserite su ambedue le linee di SDA e SCL, pena il malfunzionamento dell'intero bus.

2.5.1 Costruzione del Bus I²C

Per la realizzazione del collegamento tra il blocco di elaborazione, composto dalle dev. board del microcontrollore, e il blocco di conversione con la dev. board del DAC, è stato necessario l'attento studio di tre datasheet. Il primo naturalmente è quello di Philips [NFP], che definisce rigorosamente tutte le specifiche del bus, ed i modi corretti per l'implementazione. Il secondo e il terzo riguardano l'hardware, rispettivamente la dev. board del DAC [EB40C] e la

Multimedia Expansion Board [MEB], infatti è stato fondamentale individuare correttamente i piedini adibiti al tipo di comunicazione cercata, quindi distinguere tra i vari tipi di bus supportati dai devices (ad esempio SPI, UART, I^2C), e soprattutto, data la possibilità di avere più canali della stessa tipologia di bus, è stato importante escludere quelli che generano conflitti con le altre interfacce usate, come l'UART oppure quella ethernet.

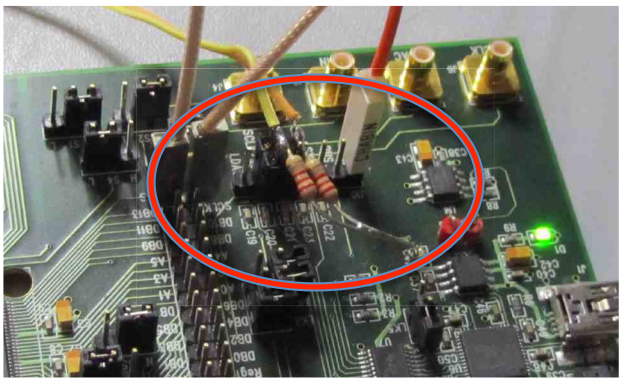


Figura 6 Resistenze di Pull-Up inserite sul bus I2C

Una volta individuati i punti per la connessione, ed inserite le due linee necessarie, (SDA e SCL), l'ultima specifica hardware da rispettare è quella dell'inserimento di due resistenze così dette di pull-up. Queste sono inserite rispettivamente a cavallo tra SDA ed SCL con il pin di VDD, il loro scopo è di garantire i livelli logici previsti anche se i dispositivi esterni sono scollegati. Come si vede in figura l'implementazione sulla scheda relativa al DAC.

2.6 Il Convertitore Digitale Analogico AD5380-5

Durante tutto il paragrafo precedente si è fatto spesso riferimento ad un convertitore digitale/analogico, adesso vedremo le sue principali caratteristiche, cercando di descrivere chiaramente gli aspetti che mi sono stati utili durante la fase di sviluppo.

Il modello di DAC usato è l'AD5380 prodotto da Analog Devices, un'altra società americana specializzata in semiconduttori e

soprattutto in conversione dati e condizionamento di segnali. Questo convertitore è caratterizzato da ben 14 Bit di risoluzione capace di fornire un'approssimazione di 0,3 mV, con la tensione che va da 0 a +5 V, per ognuno dei 40 canali disponibili. Anche questo chip è un 100 pin con un package LQFP, è fornito di svariate interfacce per la comunicazione, ma l'unica che ho usato è quella I^2C. Tra le funzioni disponibili c'è quella di Monitor, questa mi permette di impostare il piedino/canale di output n° 39 come monitor, cioè, posso usare questo pin come ripetizione di un dato canale, facendo ciò posso compiere delle misurazioni senza perturbare l'output principale. Questa funzione è essenziale per l'uso dell'ADC, perché è proprio dal pin 39 che esso legge il valore di tensione di un dato canale, permettendo la verifica, o la lettura di specifiche informazioni. Una delle potenzialità di questo chip è proprio quella che in un unico integrato si ha la possibilità di gestire tante uscite (canali). Questa peculiarità è perfetta ai fini di DMNR infatti, alla fine ogni canale sarà collegato ad un discriminatore, per impostare il proprio livello di soglia, quindi pensando che per ogni fascio di fibre ottiche scintillanti serviranno 2 discriminatori impostati alla stessa soglia, si capisce immediatamente l'alto livello di semplificazione della circuiteria. Dopo avere descritto il chip con le sue caratteristiche, passerò alla trattazione della dev. board su cui è attaccato. Come detto prima, questa è solo uno strumento utile ai fini dello sviluppo, infatti nella mainboard finale del prototipo, saranno presenti solamente gli integrati e si dovrà provvedere al corretto inserimento del resto dei componenti, già presenti sulle dev. board.

2.7 DAC Evaluation Board

Come per il microcontrollore in precedenza, anche il DAC in fase di sviluppo è stato usato corredato da un'Evaluation Board, questa mi ha permesso di semplificare alcuni settaggi, ha concesso di accedere più comodamente ai vari canali di uscita ed ha favorito la connessione del bus I²C, fornendo direttamente i pin per la connessione.

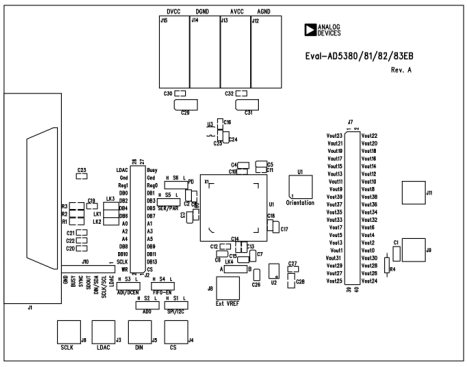


Figura 7 Evaluation Board DAC

Usando solamente l'interfaccia I²C è possibile agire su tutte le configurazioni del convertitore.

- Si può settare il valore dell'output.
- Si possono eseguire delle Special Function come:
  - Soft Reset, per il reset di tutti i registri interni.
  - Channel Monitor, per impostare il monitor di un dato canale.
  - Controll Register Write, per settare il Controll Register, una specie di pannello di controllo del convertitore.

E sono anche previsti diversi protocolli di scrittura per ottimizzare l'invio dei dati sul bus. Nel mio codice sono state prese in considerazione tutte queste alternative, in modo da ottenere un sistema il più possibile affidabile, per garantire stabilità partendo dal basso.

Per funzionare correttamente l'evaluation board ha bisogno di particolari settaggi. Per cominciare sulla parte superiore della scheda ci sono i connettori banana per l'alimentazione principale, che deve essere fornita, con un valore di 5 V, sia all'ingresso digitale sia a quello analogico. Durante lo sviluppo ho usato un alimentatore di precisione fornito dal laboratorio, perché naturalmente il valore

della tensione di reference è basilare ai fini di una buona conversione, infatti un alimentazione instabile causerebbe la generazione di una tensione altrettanto instabile. Successivamente ho dovuto provvedere al corretto posizionamento di una serie di jumper, il settaggio da me usato prevede: il pin LK3 impostato su ON, così facendo s'imposta la conversione automatica ogniqualvolta vengo aggiornati correttamente i registri del DAC. Il pin LK4 è stato impostato su B, in modo da impostare la tensione di reference a 2,5 V, che sarebbe quella della scheda. Il pin S1 impostato su H, permette l'uso dell'interfaccia I^2C piuttosto che quella SPI. Invece S2 e S3 permettono di impostare l'Address Bit; come già discusso il bus I^2C si basa su una trasmissione master/slave, in questo DAC quindi, è stata prevista la possibilità di inserire sullo stesso bus 4 convertitori digitale analogico, semplicemente modulando i 2 bit a disposizione dell'Address Bit, quindi caratterizzando ognuno dei 4 DAC con un indirizzo univoco. L'ultimo jumper importante è l'S5 serve ad impostare la comunicazione o in seriale oppure in parallelo, naturalmente usando I^2C deve essere tenuto su H, che abilita la comunicazione seriale. La connessione col computer non è stata per nulla utilizzata, poiché la casa produttrice fornisce soltanto un piccolo programmino che consente un numero limitato d'impostazioni.

Per cui, la parte hardware riguardante la creazione di una tensione si esaurisce qui, adesso mi occuperò della descrizione del sistema usato per la verifica e la lettura di questa tensione usando un altro strumento, un convertitore analogico digitale (ADC).

2.8 Il Convertitore Analogico Digitale ADC

Una volta impostato un valore di tensione per un certo canale, il problema potrebbe essere quello di monitorare le soglie o verificare il valore scritto. Proprio per questo motivo si è scelto di inserire all'interno del sistema un ADC capace, anche con una certa approssimazione, di fornirci uno strumento aggiuntivo per migliorare l'affidabilità del risultato. Proprio in quest'ottica era stato previsto un modulo ADC esterno, prodotto da Analog Devices, con una risoluzione di tutto rispetto, 12 bit, connesso tramite I^2C al microcontrollore, ma ragionando in un'ottica di diminuzione dei costi del prodotto finito, si è optato per l'uso del convertitore analogico digitale interno al pic32, questo ha una risoluzione minore, 10 Bit, ma comunque è capace di soddisfare il nostro scopo e ci consente di semplificare notevolmente la circuiteria. Nelle prossime righe descriverò i due componenti da me usati per la conversione analogico digitale, ma teniamo sempre presente che nella versione finale sarà usato l'ADC interno al microcontrollore.

2.8.1 ADC a 10 Bit interno al microcontrollore

Ormai è un dato di fatto che il microcontrollore da noi usato è veramente un ottimo strumento. Grazie proprio al convertitore analogico digitale presente al suo interno è stato possibile semplificare pesantemente l'intero sistema infatti, è bastato collegare ad uno degli ingressi analogici della Multimedia Expansion Board il pin di monitor del convertitore digitale analogico, e col giusto software a corredo, si è ottenuto il valore di tensione cercato. Come accennato prima, questo convertitore ha a disposizione 16 ingressi analogici a bordo del micro, e può restituire 8 diversi

formati di conversione, queste ad altre funzioni lo rendono davvero malleabile, pensando che si tratta anche di un modulo già presente a bordo del controllore e che basta una sola connessione per il suo funzionamento, è palese il vantaggio nell'usarlo. L'unico aspetto a sfavore è quello della risoluzione, infatti, trattandosi di un convertitore a 10 Bit per una tensione che va da 0 V a +3.3 V posiamo ottenere al massimo una risoluzione di 3 mV, che confrontata con la risoluzione del DAC è nettamente maggiore, quindi l'ADC fornisce una lettura più grossolana rispetto al valore che è capace di generare il DAC, ma questa situazione non crea disagi, ed è un approssimazione che siamo disposti a tollerare.
In una fase preliminare dello sviluppo avevo anche considerato l'uso di un ADC esterno, di seguito discuterò delle sue caratteristiche e i motivi che ci hanno spinto a tralasciarlo.

2.8.2 Il Convertitore Analogico Digitale AD7992-1

Il convertitore analogico digitale AD7992 di Analog Devices, è un componente capace di racchiudere in un package MSOP a 10 pin un convertitore con 2 canali, 12 bit di risoluzione e un'interfaccia I²C da cui scrivere i comandi e leggere i risultati. Siccome il chip non prevedeva una board per lo sviluppo,

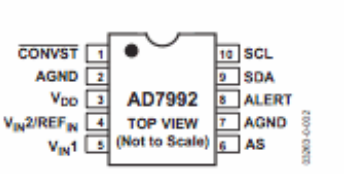


Figure 3. AD7992 Pin Configuration

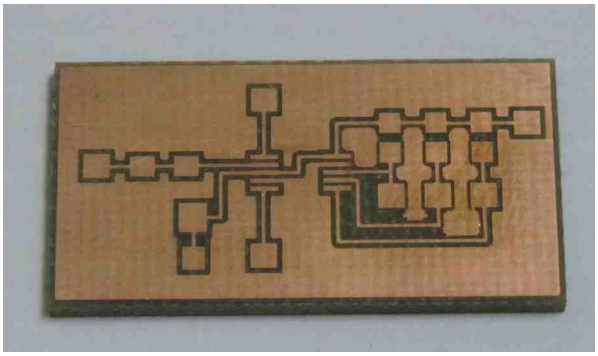


Figura 8 Scheda ADC spoglia

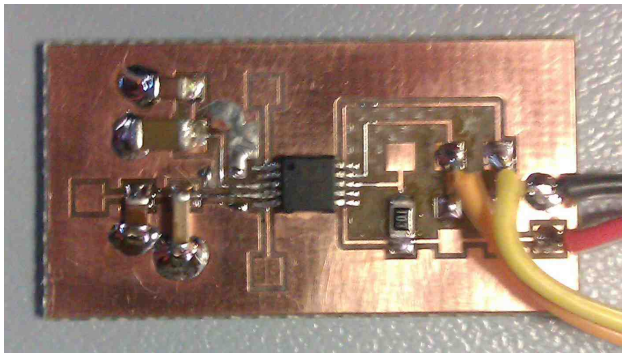


Figura 9 Scheda ADC completa

ne è stata creata una in laboratorio, usando una fresa numerica quindi è stata plasmata una basetta su cui montare i vari componenti necessari: l'ADC, condensatori e resistenze di corredo, il bus I^2C, l'alimentazione e un filo per prelevare la tensione da leggere. Tramite il bus I^2C sono riuscito ad ottenere risultati ottimi dal punto di vista della conversione, anche grazie alla versatilità del chip usato. L'uso di questo sistema è stato lasciato come possibilità fruibile, nel caso in cui in un futuro servirà maggiore accuratezza nella conversione.

Nella fase di sviluppo ho adottato una configurazione particolare, in pratica il DAC e l'ADC esterno sono stati inseriti sullo stesso bus I^2C che parte dal microcontrollore (il master) e usa i due convertitori come slave. Questo è stato possibile impostando correttamente l'indirizzo dei due devices, tutto il resto è stato affidato al software.

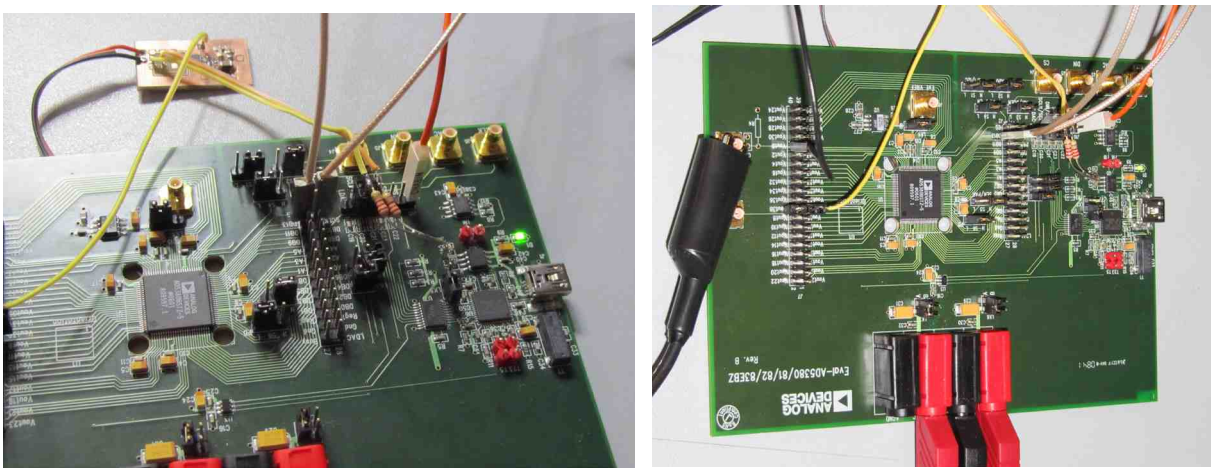


Figura 10 Esempio del sistema DAC ADC usato durante lo sviluppo

3. Architettura software

Terminata la descrizione dei componenti hardware utilizzati, possiamo concentrarci sulla parte di software implementata per fare funzionare il tutto. Per cominciare descriverò l'ambiente di sviluppo usato, per poi passare alle funzioni appositamente create per il progetto, anticipo che durante tutta la fase di scrittura del codice ho fatto largo uso delle librerie fornite dalla casa produttrice, ciò mi ha consentito di semplificare la stesura del codice, pur mantenendo alti livelli prestazionali, garantiti dalle ottimizzazioni sul codice di libreria, proveniente dalla casa produttrice. Per il reto ho cercato di mantenere un occhio di riguardo alla manutenibilità del codice, ed al riutilizzo, infatti grazie alle funzioni di trasferimento parametriche nel caso in cui si cambi dispositivo, e quindi caratteristiche hardware ad esempio: bit di risoluzione oppure tensione di reference, si potrà facilmente riusare il codice con le nuove caratteristiche senza riscriverlo per intero.

3.1 L'ambiente di sviluppo MPLAB IDE

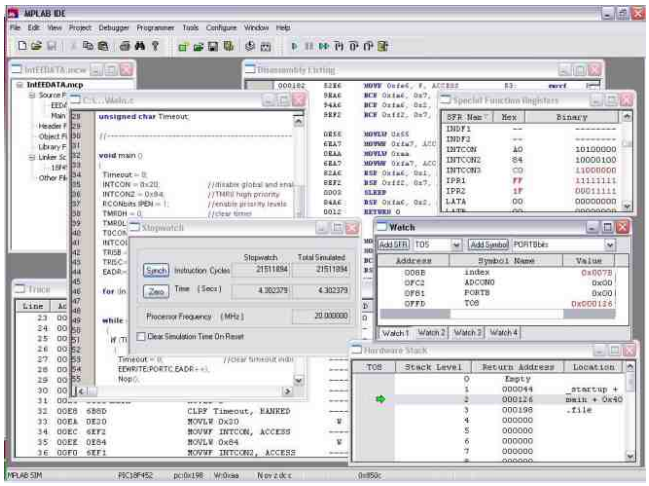
La suite di sviluppo usata per programmare il microcontrollore è quella fornita ufficialmente dalla Microchip, e si chiama MPLAB IDE. Questo "Integrated Development Environment" (IDE) è composto da diverse porzioni ognuna dedicata a un aspetto specifico del componente da programmare, c'è la parte per il monitoraggio della memoria, quella per il caricamento dell'eseguibile, quella per il debug e tutte le funzioni necessarie per l'interazione con lo Starter Kit. Questa suite, che prevede anche una versione commerciale a pagamento, ha avuto, com'è facile immaginare, un ruolo chiave nella velocità di stesura del codice, ho avuto modo di usare sia la versione

dedicata al sistema operativo Windows che quella per Mac OsX, apprezzando le differenze tra MPLAB usato su Windows e MPLAB X usato su Mac OsX. Alla fine comunque l'intero lavoro è stato portato a termine con la suite per Windows, che si è dimostrata più stabile. Per fare funzionare l'IDE basta collegare, con un cavo mini-usb, lo Starter Kit al computer dove è stata preventivamente installata la suite di sviluppo, automaticamente sarà lanciata l'installazione dei driver relativi al device, e con pochi settaggi del debugger si potrà procedere alla scrittura, compilazione e caricamento del nostro sorgente. Una volta caricato il codice all'interno della memoria del micro, lo si potrà avviare, stoppare o resettare e sarà molto semplice ed intuitivo trovare punti di deadlock e portare avanti un efficace debug.

Un altro aspetto fondamentale per l'uso di MPLAB è quello del corretto settaggio delle librerie. Infatti il microcontrollore è programmabile in Ansi-C, oppure direttamente a bassissimo livello usando funzioni Assembly MIPS (essendo un componente con un'architettura RISC). Tuttavia la casa produttrice fornisce una vasta gamma di librerie già implementate a basso livello, quindi il massimo grado di ottimizzazione, che forniscono al programmatore una comoda interfaccia usabile per evitare errori da parte di chi scrive il codice. Nel mio caso è stata utile la libreria [PPL] riguardante la comunicazione I^2C che mi ha sollevato dal carico di gestire tutte le tempistiche per i segnali su SCL e SDA, ed ha fornito semplici chiamate per avviare o stoppare una trasmissione gestendo alla perfezione i buffer e tutti i registri interni chiamati in causa. Un'altra libreria che mi è stata molto utile è quella per l'uso del modulo ADC, questa mi ha fornito dei moduli contenti i principali settaggi dei registri necessari a fare funzionare il convertitore. La soluzione adottata da Microchip, per configurare l'ADC, è quella di

prevedere delle parole chiave ordinate secondo un preciso schema, naturalmente a queste parole saranno associati precisi settaggi dei registi interni per attuare le impostazioni cercate. Il resto del codice è stato scritto usando le comuni librerie di Ansi-C, naturalmente in versione comprensibile al PIC32, con l'uso massiccio dei datasheet dei vari componenti, indispensabili per interfacciare e settare correttamente i vari chip e le schede usate. Altrettanto importanti in fase di sviluppo sono stati i sample di codice forniti da Microchip, infatti, la casa produttrice fornisce esempi praticamente per ogni tipo di applicazione del microcontrollore. Questa cosa semplifica notevolmente la vita agli sviluppatori, che possono facilmente prendere spunto dal codice fornito, e nel caso di particolari difficoltà, esiste una vasta community per fornire ogni tipo di supporto. Un altro aspetto che gioca a favore di MPLAB riguarda gli aggiornamenti, infatti la casa produttrice mantiene un frequente e costante flusso di aggiornamenti sia sull'ambiente di sviluppo, inserendo sempre nuove funzioni sull'IDE, che sulle librerie, correggendo bug delle vecchie versioni e creando nuovi esempi da cui prendere spunto.

Dopo l'introduzione dell'ambiente di sviluppo software usato, passo alla descrizione vera a propria delle funzioni create per fare funzionare l'intero sistema.



3.2 Le Funzioni Custom

Una volta assimilate le specifiche dello standard I²C e compreso il funzionamento del microcontrollore, il problema da risolvere era quello di creare un codice capace di impostare un valore su un dato canale del DAC e dopo leggerlo tramite ADC. Per questo scopo ho messo appunto una serie di funzioni che saranno descritte di seguito. Lo scheletro del progetto prevede un main dove come prima cosa saranno effettuati tutti i settaggi standard necessari allo start-up del microcontrollore, e al suo interno potranno essere richiamate in base all'uso, le seguenti funzioni:

- BOOL SetThreshold (UINT canale, float tensione)
- float ReadThreshold (UINT canale)
- BOOL reset ()
- void sveglia I2C ()

Oltre a quello del main c'è anche un file, HardwareProfile.h, dove sono stati inseriti tutti i parametri del sistema, in modo da raccoglierli in un solo posto per essere consultati e modificati facilmente. Si va dal settaggio della frequenza di clock del micro, alla velocità del bus I²C, poi a seguire l'indirizzo dello slave I²C, e ancora i valori dei parametri della funzione di trasferimento del DAC e quelli relativi all'ADC. Questa impostazione è stata pensata per consentire in un futuro il riuso del codice, infatti, basterà cambiare i valori attuali con quelli del nuovo hardware per fare funzionare il nuovo sistema senza grossi cambiamenti.

```
// Clock Constants
#define SYS_CLOCK (80000000L)

#define GetSystemClock()            (SYS_CLOCK)
#define GetPeripheralClock()        (SYS_CLOCK/2)
#define GetInstructionClock()       (SYS_CLOCK)
#define I2C_CLOCK_FREQ              5000

// DAC Constants
#define DAC_I2C_BUS              I2C2
#define DAC_ADDRESS              0x55        // 0b1010101 Serial DAC
address

// ADC Constants
#define ADC_ADDRESS              0b0100100   // 0b0100100 Serial ADC
address

// Valori Funz Trasf DAC
#define GAIN_COEFFICIENT         16382       // 0x3FFE oppure
0b11111111111110
#define OFFSET_COEFFICIENT       8192        // 0x2000 oppure
0b10000000000000
#define DAC_RESOLUTION           14          // Numero di bit DAC
#define REFERENCE_VOLTAGE_DAC    2.5F        // Valore della Reference

// Valori Funz Trasf ADC ESTERNO
#define ADC_RESOLUTION           12          // Numero di bit DAC
#define REFERENCE_VOLTAGE_ADC    5           // Valore della Reference
#define ANALOG_GROUND_ADC        0           // Valore del Ground
```

Sopra si vede l'esempio dell'implementazione del file HardwareProfile.h, con tutti i parametri usati.

Per quanto riguarda la comunicazione I²C, ho fatto largo uso delle librerie messe a disposizione, e degli esempi di supporto per la programmazione, disponibili sul sito ufficiale, perciò ho impostato una riorganizzazione dei sorgenti, mantenendo tutte le parti di controllo della comunicazione con i vari warning, ma rendendo il tutto trasparente al resto del codice. Così ho ottenuto un alleggerimento della struttura e una migliore leggibilità.

Adesso passo alla descrizione del sorgente che consente la conversione tra i valori inseriti dall'utente in valori comprensibili al convertitore e viceversa. Infatti la conversione digitale analogica richiede un certo formato di dato per funzionare, ed io non ho fatto altro che automatizzare la conversione tra il valore in Volt che viene inserito e le varie sequenze di byte che vanno inviate tramite bus I²C al DAC. La fase successiva è stata quella di convertire il valore di

tensione letto dall'ADC, che si presentava in formato binario diviso su 2 byte, in un numero corrispondente al voltaggio misurato.
Qui di seguito l'intero codice.

```
#include <HardwareProfile.h>
#include <math.h>

UINT8 Calc_DacDataFormat(BOOL mostSB, float tensione){

	int risultato, cont, i;
	double pot, pot1;
	UINT8 msb, lsb;

	pot = pow(2, DAC_RESOLUTION);
	pot1 = pow(2, DAC_RESOLUTION - 1);

	risultato = (float)(tensione * (pot / (2 * REFERENCE_VOLTAGE_DAC)) *
(pot / (GAIN_COEFFICIENT + 2))) - ((OFFSET_COEFFICIENT - pot1) * (2 *
REFERENCE_VOLTAGE_DAC) / pot);

	msb = 0b0;
	lsb = 0b0;
	cont = DAC_RESOLUTION - 1;

	for(i = DAC_RESOLUTION; i > 0; i--){
	 if( risultato >= pow(2, cont) ){
		if( pow(2, cont) > 128 ){
			msb = msb + pow(2, cont - 8);
		}else{
			lsb = lsb + pow(2, cont);
		}
		risultato = risultato - pow(2, cont);
		cont--;
	 }else{
		cont--;
	 }
	}
	if(mostSB){
	 return msb;
	}else{
	 return lsb;
	}
}

float Decodifica_ADC_Esterno(UINT8 i2cbyteMsb, UINT8  i2cbyteLsb){

	UINT16 valBinarioRiunito;
	float valTensione;
	int cont, i;

	valBinarioRiunito = 0b0;
	cont = ADC_RESOLUTION - 1;

	for(i = ADC_RESOLUTION; i > 0; i--){
```

Continua ...

```
if( i2cbyteMsb >= pow(2, cont) ){
              valBinarioRiunito = valBinarioRiunito + pow(2, cont + 8);
              i2cbyteMsb = i2cbyteMsb - pow(2, cont);
              cont--;
       }else{
              cont--;
       }
      }

      valBinarioRiunito = valBinarioRiunito + i2cbyteLsb;

      valTensione = (float)(2048 * ANALOG_GROUND_ADC + REFERENCE_VOLTAGE_ADC
/ 2 + (4095 * REFERENCE_VOLTAGE_ADC * valBinarioRiunito / 4096)) / 4096;

      return valTensione;
}

float Decodifica_ADC_Micro(UINT daConv){

      float valTensione;

      valTensione = (float)(REFERENCE_VOLTAGE_ADC_MICRO * daConv) / 1023;

      return valTensione;
}
```

Durante lo sviluppo è sorto un problema nel trattamento dei dati binari, infatti succedeva che nella divisione in 2 byte di una catena binaria di 16 bit, necessaria al settaggio del DAC, venivano persi i pesi della seconda porzione. Allora posso spiegare il funzionamento di questo codice secondo la teoria della conversione decimale binaria. In pratica si esegue la divisione per una potenza del due, crescente, e si attribuisce il giusto valore di “0” oppure di “1” al peso corretto. Usando questo metodo sono riuscito a conservare l’intera sequenza necessaria. Infatti, si presentava un problema nel caso di numero binario in cui i numeri più a destra erano zero, che venivano automaticamente non considerati dal compilatore. In questo modo il sistema ha dimostrato di funzionare correttamente con qualsiasi range d’input.

L’ultimo file appartenente al progetto è anche quello più importante, si chiama MacroFunzioni.h e contiene tutte le implementazioni delle funzioni elencate all’inizio del paragrafo. Queste si occupano essenzialmente di creare la connessione ed inviare i dati prodotti con le funzioni descritte in precedenza e leggere i valori misurati.

L'intero algoritmo è stato testato e corretto nei punti critici, si è presentata anche la necessità di inserire funzioni dedicate esclusivamente all'ottimizzazione del processo di conversione.

```
#include <plib.h>
#include <HardwareProfile.h>
#include <I2C_Function.h>
#include <DAC_Function.h>

UINT8               i2cData[10];
I2C_7_BIT_ADDRESS   SlaveAddress;
int                 DataSz;
BOOL                Success = TRUE;
UINT                channel4;    // conversion result as read from result
buffer

void svegliaI2C () {

// Initialize the data buffer
     I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
     i2cData[0] = SlaveAddress.byte;          // Address 1010101
     i2cData[1] = 0b00000000;                 // Pointer byte A5 to A0
     i2cData[2] = 0b00000000;                 // Data MSB
     i2cData[3] = 0b00000000;                 // Data LSB
// Reset di tutti i canali
     i2cData[4] = 0b00000000;                 // Pointer byte A5 to A0
     i2cData[5] = 0b00000000;                 // Data MSB
     i2cData[6] = 0b00000000;                 // Data LSB

     DataSz = 7;

     My_Init();
     My_StartTransfer();
     Success = My_TransmitOneByte(Success, DataSz, i2cData);
     StopTransfer();
}

BOOL reset () {

// Initialize the data buffer
     I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
     i2cData[0] = SlaveAddress.byte;          // Address 1010101
     i2cData[1] = 0b00001100;                 // Pointer byte A5 to A0
     i2cData[2] = 0b00110110;                 // Data MSB
     i2cData[3] = 0b00000000;                 // Data LSB
// Reset di tutti i canali
     i2cData[4] = 0b00000010;                 // Pointer byte A5 to A0
     i2cData[5] = 0b00000000;                 // Data MSB
     i2cData[6] = 0b00000000;                 // Data LSB

     DataSz = 7;

     My_Init();
     My_StartTransfer();
     Success = My_TransmitOneByte(Success, DataSz, i2cData);
     StopTransfer();
```

Continua ...

```
if(Success){
       return TRUE;
      } else {
       return FALSE;
      }
}

BOOL SetThreshold (UINT canale, float tensione) {

// Initialize the data buffer
      I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
      i2cData[0] = SlaveAddress.byte;          // Address 1010101
      i2cData[1] = 0b00001100;                 // Pointer byte A5 to A0
      i2cData[2] = 0b00110110;                 // Data MSB
      i2cData[3] = 0b00000000;                 // Data LSB

// Imposto tensione canale
      i2cData[4] = canale;                     // Pointer byte A5 to A0
      i2cData[5] = Calc_DacDataFormat(1, tensione) + 0b11000000; // Data MSB
      i2cData[6] = Calc_DacDataFormat(0, tensione);              // Data LSB

      DataSz = 7;

      My_Init();
      My_StartTransfer();
      Success = My_TransmitOneByte(Success, DataSz, i2cData);
      StopTransfer();

      if(Success){
       return TRUE;
      } else {
       return FALSE;
      }
}

float ReadThreshold (UINT canale) {

// Initialize the data buffer
      I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, DAC_ADDRESS, I2C_WRITE);
//Scrivo il Controll Register
      i2cData[0] = SlaveAddress.byte;          // Address 1010101
      i2cData[1] = 0b00001100;                 // Pointer byte A5 to A0
      i2cData[2] = 0b00110110;                 // Data MSB
      i2cData[3] = 0b00000000;                 // Data LSB
// Settaggio monitor per il canale 0
      i2cData[4] = 0b00001010;                 // Pointer byte A5 to A0
      i2cData[5] = canale;                     // Data MSB
      i2cData[6] = 0b00000000;                 // Data LSB

      DataSz = 7;
```

Continua ...

```
My_Init();
     My_StartTransfer();
     Success = My_TransmitOneByte(Success, DataSz, i2cData);
     StopTransfer();

     int i;
     for(i = 10000; i > 0; i-- ){}       //NOP

     // configure and enable the ADC
     CloseADC10(); // ensure the ADC is off before setting the
configuration
//ouput in integer | trigger mode auto
     #define PARAM1  ADC_FORMAT_INTG | ADC_CLK_AUTO
//ADC ref internal | disable offset test | disable scan mode | use single
buffers
     #define PARAM2  ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE |
ADC_SCAN_OFF | ADC_ALT_INPUT_OFF
//use ADC internal clock | set sample time
     #define PARAM3  ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_15
//set AN9 as analog inputs
     #define PARAM4 ENABLE_AN9_ANA
// do not assign channels to scan
     #define PARAM5 SKIP_SCAN_ALL
// use ground as neg ref for A | use AN9 for input A
// configure to sample AN9
     SetChanADC10( ADC_CH0_NEG_SAMPLEA_NVREF | ADC_CH0_POS_SAMPLEA_AN9 );
// configure ADC using the parameters defined above
     OpenADC10( PARAM1, PARAM2, PARAM3, PARAM4, PARAM5 );
     EnableADC10();      // Enable the ADC
     AcquireADC10();
     while ( ! mAD1GetIntFlag() ) {
// wait for the first conversion to complete so there will be vaild data in
ADC result registers
     }

     channel4 = ReadADC10(ADC1BUF9);

     return Decodifica_ADC_Micro(channel4);

     mAD1ClearIntFlag();
     CloseADC10();
}
```

In ordine di stesura, passo alla descrizione delle funzioni implementate.

- void svegliaI2C () {...} Questa funzione è stata creata esclusivamente per ovviare a un bug del sistema, infatti durante lo sviluppo ho notato delle incongruenze nella comunicazione I²C, che avvenivano solamente al primo avvio del dispositivo. Sostanzialmente la prima comunicazione dopo l'avvio del dispositivo non dava gli effetti voluti, per questo ho creato questa chiamata che non fa altro che inviare dati inconsistenti allo slave, in modo da instaurare la prima

connessione, e consentire, alle successive di andare a buon fine con gli effetti cercati. Come si vede la procedura d'invio sul bus I²C è molto rigorosa, infatti l'ordine di chiamata dei comandi e la costruzione delle cose da mandare sono in pratica sempre le stesse, naturalmente quello che deve cambiare, ed in modo corretto, è il contenuto dell'informazione.

- BOOL reset () {…} La seconda funzione invece è quella di reset. Questa usando un paradigma simile alla precedente, è capace di resettare tutti i canali del convertitore digitale analogico, impostandoli tutti a zero Volt. Sono state usate le codifiche descritte nel datasheet del DAC con i vari byte necessari alla configurazione. In oltre ho implementato anche un sistema con un flag booleano che indica la corretta esecuzione del reset.

- BOOL SetThreshold (UINT canale, float tensione) {…} Questa terza procedura consente di impostare un valore di tensione su uno specifico canale del DAC. Anche qui notiamo la sequenza per la comunicazione sul bus I²C e il meccanismo che consente di verificarne la correttezza. Ma a differenza dei casi precedenti, qui è stata necessaria una correzione nei byte da inviare, infatti notiamo in i2cData[4], i2cData[5] e i2cData[6] la possibilità variare in base alla necessità, impostando di volta in volta il canale necessario, ed i 2 byte necessari per il settaggio del valore di tensione cercato.

- float ReadThreshold (UINT canale) {…} L'ultima macro funzione creata riguarda la lettura, tramite l'ADC, della tensione. Questa restituisce naturalmente un valore decimale che rappresenta il risultato della conversione. In ingresso

basta passare il canale da cui si vuole leggere il valore. La procedura in questo caso è più complessa infatti potremmo suddividerla in 2 stadi, il primo in cui si comunica col DAC e si imposta correttamente per fornire sul piedino di monitor il valore del canale cercato. Il secondo stadio invece consiste nel settaggio dell'ADC interno del microcontrollore e nella lettura del valore misurato. Tra questi due stadi ho preferito inserire una NotOPeration per consentire al DAC di impostare il monitor correttamente, infatti questo strumento ha una certa latenza nelle operazioni, usando la NOP, allora si garantisce la corretta lettura e in un certo qual modo si sincronizzano i due dispositivi. Per tutta la parte che riguarda i settaggi dell'ADC ho fatto largo uso delle librerie, scegliendo i parametri che garantivano il minimo onere per il processore, in modo da non consumare eccessive risorse.

Questa era l'ultima funzione da me creata e considerando le nostre specifiche, il codice è sufficiente a soddisfare tutti gli usi del sistema DAC/ADC. Ho avuto un occhio di riguardo sia per la correttezza delle conversioni che per l'impiego delle risorse del microcontrollore, poiché l'uso di DAC e ADC è solo uno dei tanti aspetti del sistema finale, per questo il micro deve essere capace di rispondere ad altre richieste. A seguire descriverò la fase di test e debug, visto il massiccio uso di questi strumenti.

4. Test e Debug

La fase conclusiva del processo di sviluppo dell'applicazione mi ha impegnato dal punto di vista del testing e debugging di tutte le funzioni create. Il mio obiettivo era quello di creare su un canale del DAC una tensione, il più possibile fedele a quella cercata. Per eseguire questa verifica ho usato un multimetro collegato al pin dedicato al canale voluto, con questo strumento è stato possibile verificare all'istante se il valore di tensione veniva aggiornato correttamente, e il livello di accuratezza che si otteneva. Naturalmente il processo di conversione genera una piccola incertezza, causata dall'approssimazione intrinseca dello strumento, proporzionale al numero di bit e al valore della VRef che rappresenta il fondo scala dello stesso. Perciò ho verificato che tutte le misure rientrassero in questa caratteristica. Ho notato anche un altro problema noto dei convertitori in genere, quello del Gain Error, questo è causato dagli amplificatori presenti all'interno del componente, e comporta una piccola deriva della funzione di trasferimento ideale, per valori vicini alla VRef. Il DAC da me usato però prevede un sistema per correggere questo scostamento, e quindi consente la completa affidabilità della misura.

La parte del debug invece è stata portata avanti passo passo, durante tutta la stesura del codice, usando i servizi forniti dalla suite di sviluppo. Tutte le procedure sono state verificate usando la chiamata DBPRINTF(), fornita dalle librerie proprietarie della Microchip, che mostra a schermo messaggi e valori richiesti. In alcuni casi l'uso di questa chiamata si è rivelato troppo oneroso per il microcontrollore, portando alla luce piccoli bug[6] del sistema Starter Kit, quindi abbiamo superato il problema usando i led programmabili presenti

[6] Ogni volta che si usa la chiamata DBPRINTF() viene disabilitata la UART1.

sulla scheda, questi all'accensione o allo spegnimento ci fornivano un referto sulle operazioni effettuate. Nella release finale della build, naturalmente tutta la parte di debug è stata eliminata, per fare eseguire al micro solo il codice utile al nostro obiettivo.

Un altro test importante è stato eseguito fondendo le funzioni sviluppate da me con il codice sviluppato da un collega, che riguarda un altra parte del progetto DMNR, quella del conteggio degli impulsi generati dai discriminatori. Il sistema di conteggio essenzialmente fa uso dell'interfaccia UART, e gestisce gli interupt del controllore in modo particolare. Anche questo test è andato a buon fine, dopo avere impostato correttamente le interfacce escludendo i conflitti.

Per quello che riguarda la capacita di resistenza agli errori delle mie funzioni, ho considerato un ambito applicativo in cui l'uso dei miei metodi viene effettuato automaticamente da altre routine, magari in modo sistematico e impostando a priori i range di utilizzo, ed i giusti parametri da passare. Per questo fornire una corretta descrizione delle funzioni da me create dovrebbe bastare ad un futuro sviluppatore a completare il progetto in modo consistente. Naturalmente il progetto DMNR non è fatto solo da DAC e ADC; a seguire descriverò un altro aspetto di cui mi sono interessato, quello dell'interfacciamento sulla rete di tutto il progetto.

Figura 11 Esempio delle configurazioni usate per lo sviluppo

5. Controllo remoto e distribuito

Tutto quello descritto fino adesso riguarda una piccola porzione di un sistema ben più grande, che è il progetto DMNR che oltre al trattamento dei dati misurati deve prevedere anche una qualche procedura che interfacci ogni singola postazione di misura con una console remota, che è proprio la caratteristica che permette di aumentare la sicurezza della gestione dei rifiuti nucleari eliminando l'intervento umano dedicato al monitoraggio.

Proprio per questo motivo, fin delle prime fasi di progettazione è stato fondamentale l'inserimento di un interfaccia ethernet come parte integrante del progetto. Proprio attraverso questa interfaccia, si è pensato di fare passare tutte le informazioni dal sistema, di misura al mondo esterno e dall'esterno al sistema di misura. Semplicemente l'interfaccia ethernet sarà l'unico componente ad avere l'onere di trasmettere e ricevere tutto quello riguardante il nostro sistema.

Essendo queste operazioni comunque non banali, si è correttamente scelto di affidarle al microcontrollore presente sulla mainboard. Appunto per questo la versione della dev. board dedicata al microcontrollore, lo starter kit, è stata scelta tra le tante, nella versione ethernet starter kit, che di base prevede la presenza sulla scheda di un connettore RJ45 cablato con i giusti pin del microcontrollore.

In questo modo si aprono tanti scenari di utilizzo, infatti pensando ad esempio a un pc collegato a tutti i rivelatori di un unico deposito, oppure ancora all'utilizzo un qualsiasi tipo di rete locale per consentire il monitoraggio anche a distanza. Sicuramente il tutto necessiterà dei giusti applicativi dal lato console e di un ottimo software a bordo del microcontrollore. Tutto questo per adesso è in

lavorazione, la creazione del software come l'assemblaggio dell'hardware.
Nei prossimi paragrafi descriverò per primo lo stack da me studiato per verificare l'effettiva fattibilità delle specifiche richieste ed a seguire una breve considerazione sul possibile sviluppo sulla rete, intesa come internet, dell'intero sistema.

5.1 Lo stack TCP/IP

Per soddisfare le premesse fatte prima è chiaro che, oltre all'applicazione che gira sulla console remota, ricopre un ruolo strategico il codice fatto girare all'interno del microcontrollore.
A proposito di ciò la Microchip fornisce uno strumento software davvero potentissimo. Oltre a fornire le normali librerie che consentono di alzare il livello della programmazione, propone un intero esempio dimostrativo, dove in un unico progetto, si integrano tutte le procedure necessarie alla comunicazione su rete. Grazie all'uso di questo stack sono riuscito a fare girare una basilare server web, in grado di rispondere ad alcune mie richieste, a rendere operativa la scrittura sul DAC e la lettura da ADC ed ho integrato il modulo software sviluppato da un collega che si occupa del conteggio degli impulsi. Il tutto in un'unica applicazione che gira sul microcontrollore, dimostrando in questo modo che un piccolo pic ha la potenza necessaria per rispettare le specifiche.
L'affinamento della parte di progetto relativa allo stack TCP/IP e tutt'ora in corso, perché l'obiettivo è quello di escludere dal server web tutte le parti inutili e mantenere solo quelle necessarie come la sezione per la comunicazione UDP che è quella scelta per la connessione ethernet, infatti si suppongono piccole sessioni di

trasmissione, non necessitando per cui di una connessione TCP indicata per sessioni di trasmissione ben più lunghe. Un serio problema è stato riscontrato col DHCP, che allo scopo di garantire un ip univoco ad ogni sistema di misura, veniva disabilitato, ma in questo caso il progetto si comportava in modo strano, scoprendo la presenza di malfunzionamenti nel software. In fondo comunque il problema è stato già circoscritto, quindi sicuramente troverà soluzione al più presto, magari anche grazie all'aggiornamento dello stack correggendo bug critici.
Nell'ultimo paragrafo del capitolo farò brevi considerazioni sull'espansione sulla rete del progetto.

5.2 Sviluppo sulla rete del progetto

In un'ottica futura sarebbe bello pensare ad una gestione dei rifiuti nucleari del tutto trasparente, magari fornendo in chiaro su internet i valori delle letture effettuate del progetto DMNR, escludendo naturalmente la parte relativa al controllo, in modo da assecondare l'aumentare della sensibilizzazione nei confronti di argomenti come l'inquinamento ambientale. Il tutto a vantaggio della società che applica questa politica, perché garantisce uno standard di sicurezza alto, da cui sarebbe perfetto trarre il motto aziendale.

6. Conclusioni

Allo stato attuale tutta la parte dedicata al funzionamento di DAC e ADC è perfettamente testata e funzionante, secondo i canoni delle specifiche e nei limiti evidenziati in questa relazione. È chiaro che alla fine il sistema richiederà diverse sessioni di test conclusive per affinare aspetti che si potranno rivelare critici. Al momento è evidente la robustezza del codice per la conversione da digitale ad analogico e da analogico a digitale. Per il resto l'esperienza dello stage è stata davvero interessante, perché ho avuto la possibilità di lavorare sulla linea di confine tra software e hardware, avendo modo di toccare con mano tutte le migliorie create di volta in volta. Tutti gli strumenti usati e l'ambiente di lavoro mi hanno dato un carico di esperienze davvero prezioso, e perfettamente pertinente al mio corso di studi.

Bibliografia

[IAEA10] Handbook on Nuclear Law – Implementing Legislation *International Atomic Energy Agency, 2010*

[DMNR] Strategic Project INFN Energy Detector Mesh for Nuclear Repositories http://www.lns.infn.it/link/dmnr

[ANPA26] La gestione dei rifiuti radioattivi Guida tecnica n° 26 Agenzia Nazionale Protezione Ambiente

[NFP] NXP Founded by Philips. *I2C-bus specification and user manual.*

[EB40C] Analog Devices. *Evaluation Board for 32-/40-Channel, 12-/14-Bit, Parallel and Serial Input, Voltage-Output DACs EVAL-AD5380/81/82/83EB.*

[PFDS] Microchip. PIC32MX5XX/6XX/7XX Family Data Sheet. High-Performance, USB, CAN and Ethernet 32-bit Flash Microcontrollers.

[PESK] Microchip. PIC32 Ethernet Starter Kit User's Guide.

[MEB] Microchip. Multimedia Expansion Board User's Guide.

[PPL] Microchip. PIC32 Peripheral Libraries for MPLAB C32 Compiler.

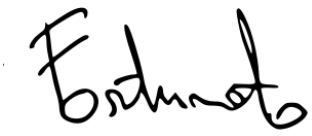
Ringrazio:

Con sentito affetto, i miei genitori e mio fratello, che mi hanno permesso di crescere e continuare a farlo.

Con riconoscenza per la professionalità ed il tempo dedicatomi, il Professore Michele Malgeri, che è stato sempre disponibile a chiarire qualsiasi dubbio.

Con un incommensurabile apprezzamento al Laboratorio Nazionale del Sud ed INFN, per circoscrivere i magnifici personaggi che lo rendono vivo. In particolare il Dott. Paolo Finocchiaro, sempre presente per risolvere qualsiasi incertezza e per citare un esclamazione di un noto Professore, “un Artista”. I responsabili del Dipartimento di Elettronica e Rivelatori: Claudio Calì, Pietro Litrico, Giuseppe Passaro e Salvatore Marino; che mi hanno ospitato nelle lunghe giornate di sperimentazione.

Per concludere ringrazio Sergio Scirè e Gianfranco Vecchio, che sono stati sempre disponibili ed hanno condiviso con me le loro esperienze.



Dedicato a Nonno Nato, Nonno Giovanni e Zio Franco.