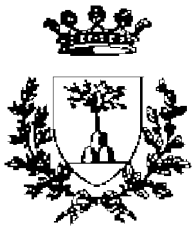
Università degli Studi di Ferrara

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica



REALIZZAZIONE DI UN'APPLICAZIONE PER L'ACQUISIZIONE DATI DA UN RIVELATORE A FIBRE SCINTILLANTI

Primo Relatore :

Dott. MIRCO ANDREOTTI

Secondo Relatore :

Dott. GIANLUIGI CIBINETTO

Laureando :

MATTEO MANZALI

ANNO ACCADEMICO 2007/2008

...alla mia famiglia, che mi è
sempre stata vicina...

Indice

Introduzione

In questa tesi presentiamo la realizzazione del sistema di acquisizione dati per lo studio sulle prestazioni di un nuovo rivelatore di particelle. In generale quando si utilizzano rivelatori di particelle si devono trattare segnali elettrici molto veloci, quindi inevitabilmente si devono utilizzare sistemi di elettronica in grado di manipolare tali segnali con frequenze molto alte. In conseguenza di questo fatto l'acquisizione, l'elaborazione e la memorizzazione dei dati corrispondenti deve essere affidata ad un calcolatore in grado di gestire con sufficiente rapidità tutte queste operazioni. Il calcolatore dovrà essere interfacciato con il sistema di rivelazione attraverso opportuni sistemi di interfaccia e dovrà quindi gestire l'acquisizione dei dati utilizzando un'applicazione software adeguatamente progettata. Nell'ambito del nostro esperimento il rilevatore di particelle è costituito da uno scintillatore plastico accoppiato con fibra ottica di tipo WLS e due fotorivelatori di tipo SiPM. In particolare si desidera quantificare l'efficienza di rilevazione dello scintillatore (la percentuale di particelle che riesce a rilevare) e la risoluzione temporale, cioè quantificare l'errore temporale da cui è affetta ogni misurazione. Le misurazioni vengono effettuate tramite un convertitore analogico-digitale (modulo ADC) e un digitalizzatore di tempo (modulo TDC), interfacciati ad un pc tramite un adattatore PCI. Tutto il lavoro di tesi è stato svolto nel laboratorio Test Rivelatori del Dipartimento di Fisica dell'Università degli Studi di Ferrara. La prima fase del lavoro di tesi si è concentrata sullo studio delle caratteristiche dei componenti del sistema di acquisizione e delle librerie fornite dalle case produttrici dell'hardware. Sucessivamente si è passati alla realizzazione, in linguaggio C, delle funzioni che permettono: la connessione e disconnessione dall'apparato, l'inizializzazione dei moduli e la lettura del *buffer di output* dei moduli. Queste funzioni sono poi state racchiuse in una libreria dinamica, per poter essere utilizzate da programmi realizzati in linguaggi differenti dal C. Infine si è sviluppata l'applicazione finale che utilizza la libreria precedentemente creata per interagire con l'apparato di acquisizione dati ed elabora i dati ricevuti, creando degli istogrammi. La presenza di questi istogrammi facilita lo studio dei dati acquisiti.

Inoltre i dati acquisiti tramite le funzioni racchiuse nella libreria, oltre ad essere restituiti dalle funzioni al programma chiamante, vengono inseriti in un file che può eventualmente essere utilizzato per studi più approfonditi. Infine per testare il corretto funzionamento dell'applicazione sviluppata si è effettuato un ciclo di acquisizione dati: lo studio dei dati raccolti ha fornito le caratteristiche dello scintillatore, che lo scopo dell'esperimento si proponeva di individuare.
Di seguito verranno descritti gli argomenti trattati nei vari capitoli.

Primo capitolo:

Viene descritta la composizione dell'apparato sperimentale e lo scopo per cui è stato costruito.

Secondo capitolo:

Si descrive l'hardware utilizzato per rendere possibile l'acquisizione dati dall'apparato sperimentale su un pc.

Terzo capitolo:

Vengono elencate e descritte le funzioni create per l'interfacciamento con i moduli di acquisizione dati.

Quarto capitolo:

Viene descritto il programma finale per l'acquisizione e analisi dei dati.

Quinto capitolo:

Vengono illustrati i risultati raccolti dopo un ciclo di acquisizione.

Capitolo 1

Apparato sperimentale

In questo capitolo verrà descritto l'apparato sperimentale per il quale si è sviluppato il sistema di acquisizione dati. Attraverso questo apparato, utilizzato nel laboratorio Test Rivelatori del Dipartimento di Fisica, si svolgono attività di studio sulle prestazioni di un nuovo rivelatore di particelle. L'apparato sperimentale è costituito da uno scintillatore plastico accoppiato con fibra ottica di tipo WLS e due fotorivelatori di tipo SiPM: elementi che descriviamo in dettaglio nel seguito.

1.1 Scintillatore

Lo scintillatore è un materiale capace di emettere impulsi di luce, in genere visibile o ultravioletta, quando viene attraversato da particelle cariche. Esistono diverse tipologie di scintillatori che si distinguono per tipo di materiale di cui sono composti, i tempi di risposta, le lunghezze d'onda emesse e l'efficienza di scintillazione (quanta energia viene convertita in luce). Lo scintillatore che viene usato in questo esperimento è di tipo plastico: è formato da una soluzione di materiale scintillante organico disciolto in un solvente successivamente polimerizzato, diventando così una soluzione solida. Gli scintillatori plastici offrono un segnale molto veloce con un tempo di risposta di circa 2 o 3 ns. Uno dei maggiori vantaggi degli scintillatori plastici è la flessibilità, che li rende facilmente manipolabili; il loro costo non eccessivo li rende particolarmente utili nel caso si necessiti di ampi volumi di scintillatori.

1.2 SiPM

Il SiPM o Silicon Photomultiplier è un nuovo tipo di fotomoltiplicatore al silicio. Diversamente dai fotomoltiplicatori tradizionali (PMT o Photomultiplier Tubes), costruiti con tubi a vuoto, i SiPM sono sostanzialmente dei diodi polarizzati inversamente, che trasformano la luce incidente in impulsi elettrici. Rispetto ai tradizionali PMT presentano numerosi vantaggi quali ad esempio la bassa tensione di funzionamento (da 30 a 80 V a seconda del modello e del costruttore) mentre i PMT richiedono tensioni di lavoro di alcuni kV; le dimensioni, che nei SiPM sono dell'ordine di 1 mm^2 mentre nei PMT di qualche centimetro, e la facilità di integrazione: i SiPM infatti possono essere facilmente inseriti a decine o centinaia in schede di silicio (formando vettori o matrici di foto moltiplicatori).
I SiPM sono stati prodotti per la prima volta in Italia nel 2005 dall'IRST-ITC di Trento nell'ambito di un programma di ricerca tecnologica svolto insieme all'Istituto Nazionale di Fisica Nucleare (INFN). I parametri che caratterizzano un photo detector sono i seguenti:

guadagno : ampiezza del segnale elettrico per ogni singolo fotone;

efficienza quantica : la probabilità di rivelare un singolo fotone incidente;

noise o Dark Counting Rate : probabilità che il fotorivelatore veda un evento in una stanza priva di luce, è il parametro principale per applicazioni con una bassa frequenza di eventi;

linearità : indica la linearità dell'aumento del guadagno all'aumentare del numero di fotoni catturati;

Nell'ambito del nostro apparato, i SiPM sono incorporati in un circuito che ne regola l'alimentazione, e può integrare un discriminatore per trasformare l'impulso elettrico in un'onda quadra (come illustrato in figura 1.1).

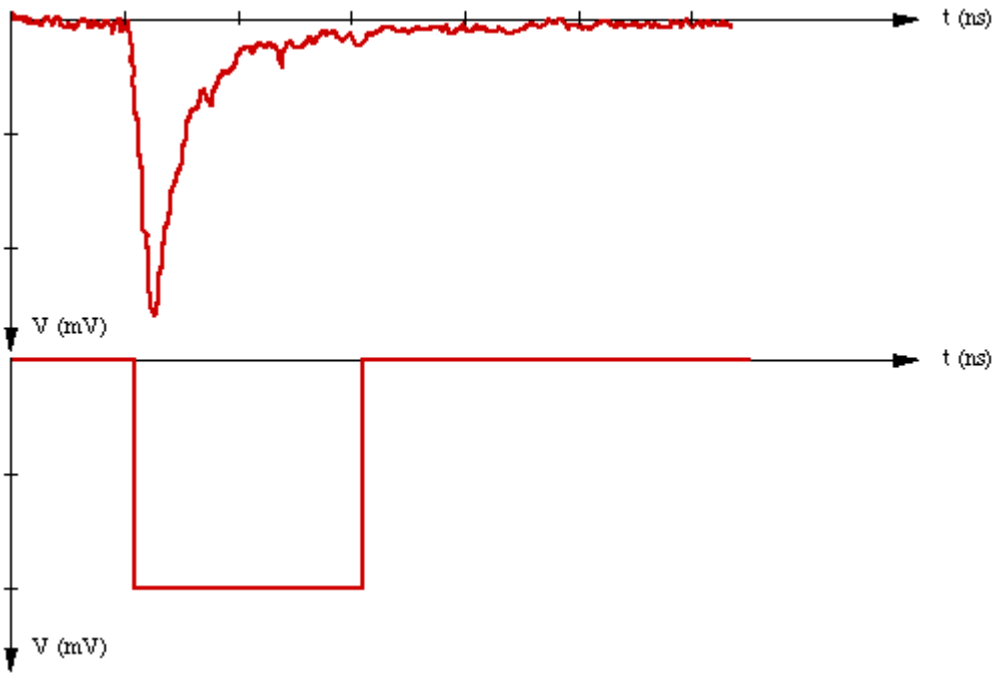


Figura 1.1: Rappresentazione grafica del funzionamento di un discriminatore: il primo grafico rappresenta l'output del SiPM, il secondo grafico rappresenta invece l'output del discriminatore

1.3 Fibra ottica

Le fibre ottiche sono filamenti di materiali vetrosi o polimerici, realizzati in modo da poter condurre la luce. Sono normalmente disponibili sotto forma di cavi (vedi figura 1.2). Sono flessibili, immuni ai disturbi elettrici ed alle condizioni atmosferiche più estreme, e poco sensibili a variazioni di temperatura. La tipologia da noi impiegata è la WLS (WaveLength Shifter): un materiale fotofluorescente che assorbe fotoni che hanno una frequenza elevata ed emette fotoni aventi frequenza più basse. La fibra attraversa lo scintillatore ed è il mezzo attraverso il quale la luce prodotta dallo scintillatore arriva al SiPM.

Figura 1.2: Esempio di fibre ottiche

1.4 Descrizione dell'apparato

Lo scintillatore principale è posizionato orizzontalmente tra altri due scintillatori di dimensioni molto ridotte (chiamati scintillatori di trigger), che si trovano in posizione speculare rispetto allo scintillatore principale. La fibra ottica passa attraverso lo scintillatore ed arriva a due fotodiodi (SiPM) posizionati alle estremità, sui quali verranno poi effettuate le rilevazioni. Per produrre una reazione negli scintillatori si utilizzano i raggi cosmici, che sono una sorgente naturale di particelle cariche provenienti dallo spazio.
Infine vi è un sistema di trigger che serve per assicurare la presenza di un evento da rilevare. Il sistema di trigger, formato da due scintillatori di trigger, due SiPM e due discriminatori, viene descritto in maniera dettagliata nel seguito. In figura 1.3 è presente lo schema rappresentante il circuito appena descritto.

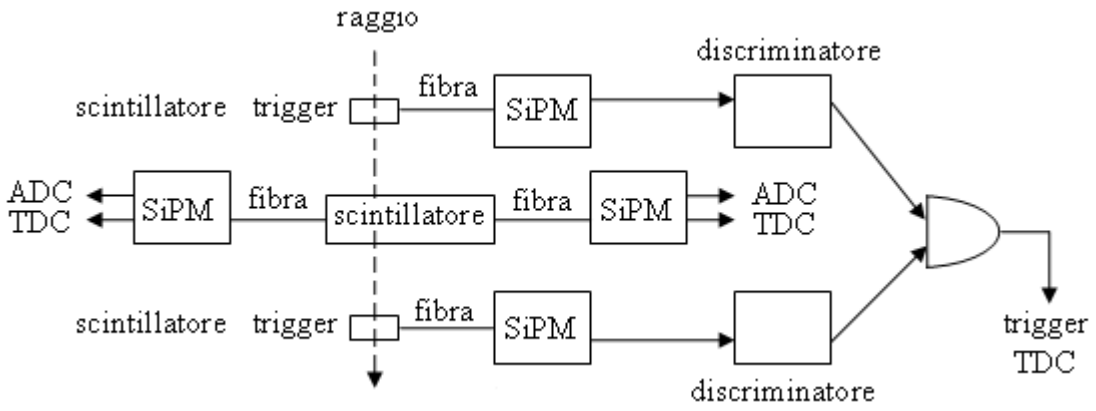


Figura 1.3: Schema dell'apparato

1.5 Sistema di trigger

Gli scintillatori di trigger vengono utilizzati per far si che le misure effettuate sullo scintillatore principale avvengano solamente per le particelle che attraversano un determinato punto (quello coperto dagli scintillatori di trigger). Idealmente tutti i tempi misurati devono essere identici, in quanto il punto colpito dalle particelle è sempre lo stesso; nella realtà invece le misure hanno sempre un errore, quindi si ha un'imprecisione nell'identificare la posizione in cui sono passate le particelle. Uno degli scopi di questo esperimento è proprio quello di individuare questo errore per poter poi sapere, senza l'utilizzo di scintillatori trigger, l'intervallo spaziale in cui ogni particella è passata.
Ad ogni scintillatore di trigger è collegato un SiPM con un discriminatore incorporato che trasforma l'impulso elettrico in un segnale impulsivo ad onda quadra. Le uscite di questi due SiPM vengono combinate in un circuito che implementa un AND logico. L'output del sistema di trigger produrrà un segnale solo quando entrambi gli scintillatori trigger saranno attraversati da una particella.

1.6 Scopo e misure dell'esperimento

Il primo obbiettivo dell'esperimento consiste nel quantificare l'efficienza di rilevazione dello scintillatore principale, cioè la percentuale di particelle che riesce a rilevare. Per ottenere questo valore si calcola il rapporto tra le particelle individuate dal sistema di trigger e il numero di volte che lo scintillatore principale ha rilevato tali particelle. Le misure effettuate sono affette da errore, e raccogliendo in un istogramma i valori dei tempi acquisiti si ottiene una curva a campana.

Più la σ di questa curva è piccola, più è piccolo l'errore da cui i dati sono affetti. Il secondo obbiettivo delle misurazioni è di calcolare la σ, per poter individuare l'intervallo spaziale in cui la particella ha attraversato lo scintillatore.
Le misure vengono effettuate sui SiPM posti ai due estremi dello scintillatore principale. I valori misurati per ogni fotorivelatore sono la quantità di carica di un solo impulso e il tempo di arrivo dello stesso. Questi valori sono in relazione: se un raggio attraversa lo scintillatore vicino all'estremità destra arriverà per primo l'impulso dal diodo posizionato sulla destra ed avrà una carica maggiore (avendo percorso meno distanza).
Un'ulteriore misura da effettuare è quella del tempo t_0 in cui il ciruito di trigger rileva il passaggio di una particella carica, è necessaria perché i tempi t_1 e t_2 letti dai due SiPM dello scintillatore non sono in relazione col trigger: solo calcolando $t_1 - t_0$ e $t_2 - t_0$ otteniamo i tempi di arrivo alle due estremità dello scintillatore (come mostrato in figura 1.4).

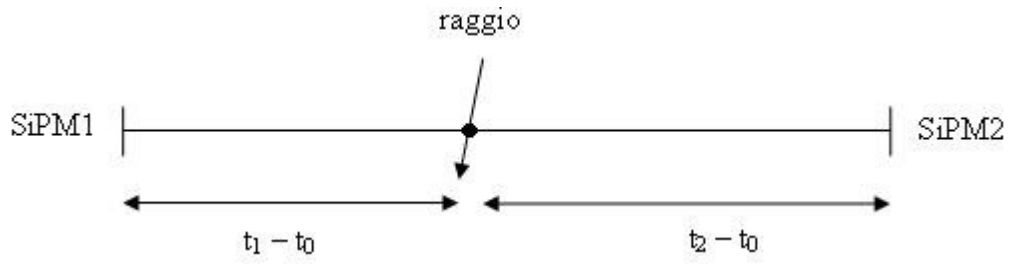


Figura 1.4: Schema dei tempi

Capitolo 2

Sistema di acquisizione

In questo capitolo si descriverà l'hardware utilizzato come interfaccia tra l'apparato e il calcolatore (PC). L'hardware in questione è composto da un *crate* in grado di ospitare diversi moduli per la conversione analogico digitale e per la sincronizzazione temporale. La tecnologia del bus dei moduli deve coincidere con il tipo di crate utilizzato: per la fase iniziale di test è stato utilizzato un crate CAMAC, successivamente però abbandonato in favore di un crate VME. Questa scelta è stata forzata, in quanto si sono cercate maggiori possibilità di integrazione con altre catene di acquisizione, e ormai il bus VME rappresenta l'attuale standard di mercato, la tecnologia CAMAC invece è datata e viene utilizzata in un ristretto numero di ambiti. Inoltre i moduli VME offrono caratteristiche più evolute rispetto ai moduli CAMAC, come per esempio maggior numero di canali e migliore risoluzione nelle letture.

2.1 Il bus VME

VMEbus (Versa Module Europa) è una tipologia di bus che fa uso dello standard Eurocard (è uno standard per i circuiti elettrici stampati) ed è stato introdotto da Motorola, Philips, Thompson e Mostek nel 1981. VMEbus è stato concepito per essere un ambiente flessibile capace di soddisfare una varietà di intense richieste di calcolo, ed è diventato un protocollo molto popolare nell'industria elettronica. Un esempio del *crate* utilizzato nell'esperimento è presente in figura 2.1.

Un trasferimento tipico di dati consiste in un ciclo di arbitraggio (per ottenere il controllo del bus), un ciclo di indirizzo (per selezionare il modulo) e il ciclo di dati. Sono supportati letture, scritture, modifiche e trasferimenti di blocchi di dati. Il VME è un bus multi-processore che permette l'uso di CPU a 8, 16, 32, 64

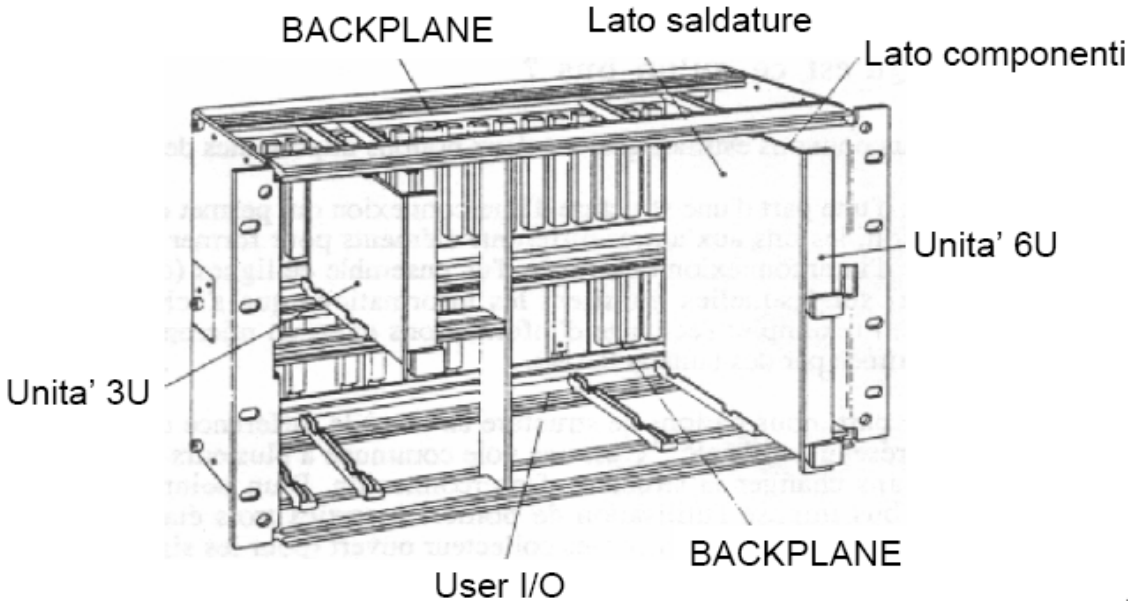


Figura 2.1: Esempio di crate VME

bit. La struttura del BUS è master-slave (vedi fig. 2.2): il master gestisce il ciclo e lo slave che ha decodificato il proprio indirizzo risponde alla transazione. Il bus permette trasferimenti dati asincroni: lo slave termina il ciclo con un *acknowledge* quando ha finito di processare la transazione. Esiste un ciclo indivisibile *Read Modify Write*, che permette l'implementazione di semafori, al fine di gestire risorse comuni in un sistema multi-master o multi-processore. Sono disponibili 7 linee di interrupts per richiedere l'attenzione del o dei master. Le specifiche (IEEE 1014-1987) definiscono il protocollo per il trasferimento dati e la gestione del bus stesso, i livelli elettrici utilizzati e tutta la meccanica(formato delle schede e connettori).
Le linee del bus possono essere classificate in 4 categorie:

Arbitration bus : il modulo che controlla il bus metterà la *bus busy line* (BBSY) bassa per mostrare che è in uso. Quando questa linea non è bassa l'*arbiter module* controllerà le linee di richiesta del bus (BR0-BR3) alla ricerca di una richiesta pendente. Le richieste su BR3 hanno la massima priorità.

Data transfer bus : è usato per leggere e scrivere dati tra i moduli. Il *data bus* (D00-D31) contiene i dati attuali durante il trasferimento. L'indirizzo del registro a cui si sta accedendo è presente nel address bus (A01-A31). Le linee *address midifier* (AM00-AM05) indicano la lunghezza dell'indirizzo, il tipo di

dati e l'ID del modulo master. L'indirizzo *strobe* (AS) viene usato per segnalare la presenza di un indirizzo valido. I dati strobe (DS0-DS1) sono usati dal modulo che controlla il trasferimento (modulo master) per segnalare la presenza e l'accettazione di dati validi sul bus, insieme a informazioni sulla lunghezza della word da trasferire. La linea WRITE viene usata per distinguere tra operazioni di lettura e scrittura. Il *data transfert acknowledgement* (DTACK) è usato del modulo a cui viene effettuato l'accesso (slave) per il completamento del trasferimento. Errori in questo trasferimento sono segnalati utilizzando la *bus error line* (BERR).

Priority interupt bus : contiene le linee di richiesta dell'interrupt (IRQ1-IRQ7). IRQ7 ha la priorità più alta. In risposta ad un interrupt viene generato un ciclo di indirizzo, dove l'indirizzo indica la richiesta che dev'essere accettata. L'*interrupt acknowledgement* (IACK) diventa arbiter, ciò significa la che richiesta viene accettata, e segue un ciclo di dati in cui il modulo che ha richiesto l'interrupt comunica il suo stato e ID.

Utility bus : supporta un clock indipendente di una frequenza di 16 MHz (SYSCLK). Permette l'inizializzazione attraverso la linea *system reset* (SYSRESET). Contiene due linee (SYSFAIL e ACFAIL) per la segnalazione di problemi generici, un'ulteriore linea permette un trasferimento addizionale di dati attraverso la linea *serial data* (SERDAT).

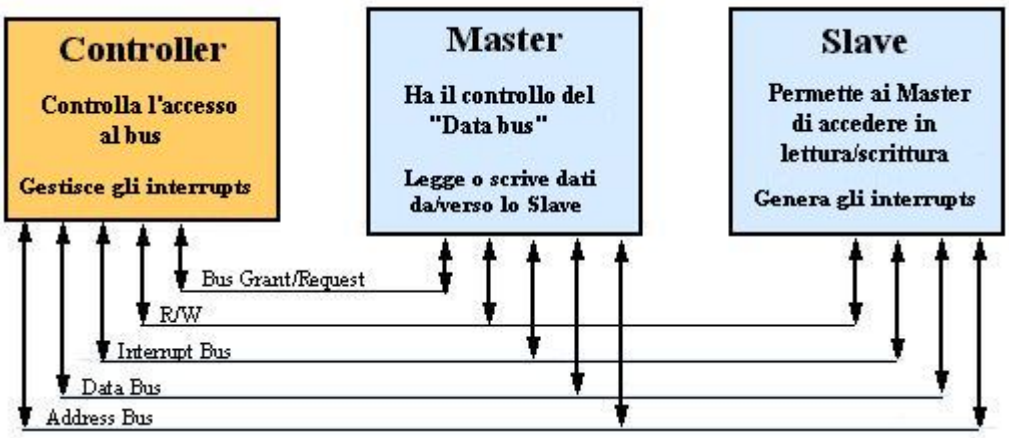


Figura 2.2: Struttura del bus VME

2.2 Modulo ADC L1182

Il modello L1182 (fornito dalla LeCroy) è un *Analogic to Digital Converter* (ADC), cioè un convertitore analogico-digitale. Il modulo ha otto canali e funziona come digitalizzatore della carica elettrica ricevuta su ognuno degli otto ingressi, la corrente applicata ad un ingresso viene integrata durante tutto il periodo in cui un segnale di gate è applicato al modulo (il segnale di *gate* è comune a tutti gli ingressi). Gli otto integrali degli inputs vengono digitalizzati e i risultati sono salvati nella memoria del modulo. Possono essere conservati fino a sedici eventi nel buffer di memoria dell'ADC.

2.2.1 Connettori, controlli e comandi

L'intera configurazione del L1182 viene effettuata attraverso il bus VME. Tutti i bit di controllo e le locazioni di memoria sono mappate nello spazio indirizzi VME. Il pannello frontale del 1182 supporta tredici connettori: otto sono per i segnali di input, due connettori per il Gate, due connettori per il Clear ed infine l'ultimo connettore è un output per il segnale di Conversion-in-Progress (CIP).
Visibili sul pannello frontale del modulo vi sono otto led che permettono di controllare lo stato degli switch che definiscono gli otto bit più significativi dell'indirizzo del modulo. Tre altri led rossi forniscono invece indicazioni su: la validità dell'accesso al modulo via VME, CIP e power.
Tutti i comandi del L1182 sono implementati come bit di controllo nello spazio indirizzi VME. Stato e bit di controllo sono localizzati all'indirizzo 0. I successivi sette indirizzi permettono di accedere ai puntatori per operazioni sulla memoria dati.

2.2.2 Istruzioni

L'ADC 1182 è controllato tramite il bus VME. Utilizza lo standard A24/D16: A24 indica che l'indirizzamento è a 24 bit, mentre D16 significa che le *word* sono di 16 bit. poiché in VME gli indirizzi si riferiscono a un byte specifico (8 bit), ogni *word* è identificata da due indirizzi. Nelle operazioni con *word* a 16 bit, normalmente si indirizza l'*upper byte* (D16-D8) e il *lower byte* viene implicitamente incluso nella word letta o scritta.

Address Structure	
A16 D A23	Base address (switches)
A9 D A15	Invalid addresses
A1 D A8	CSR and RAM space
A0	Undefined in VME
0x0000, 0x0001	CSR 0
0x0002, 0x000F	Invalid addressing
0x0100, 0x0101	Channel 0 Event 0
0x0102, 0x0103	Channel 1 Event 0
0x0104, 0x0105	Channel 2 Event 0
0x0106, 0x0107	Channel 3 Event 0
0x0108, 0x0109	Channel 4 Event 0
0x010A, 0x010B	Channel 5 Event 0
0x010C, 0x010D	Channel 6 Event 0
0x010E, 0x010F	Channel 7 Event 0
0x011_	Channel 0 Event 1
0x012_	Channel 0 Event 2
.	
.	
.	
0x01F_	Channel 0 Event 15
0x0200, 0xFFFF	Invalid addressing

Figura 2.3: Struttura degli indirizzi nel modulo ADC 1182

Come illustrato in figura 2.3 il *Control and Status Register* (CSR0) è il primo indirizzo e fornisce l'accesso allo stato del 1182. L'accesso a questo registro è identico all'accesso a qualsiasi altro indirizzo VME. La particolarità di questo registro consiste nell'essere sempre accessibile e di rispecchiare lo stato corrente del 1182. La lista seguente identifica le funzioni controllate dai bit di CSR0:

$CSR0 < 0 >$ indica che la conversione è completa e i dati sono disponibili per la lettura, questo bit ($CONV\_COMP$) diventa true dopo che CIP ha terminato. Questo è un bit di sola lettura.

$CSR0 < 1 >$ indica che vi è una conversione in corso. Questo significa che i dati immagazzinati durante l'intervallo di tempo in cui il *gate* era attivo ora vengono digitalizzati e salvati in memoria. Quando il processo è terminato, CIP viene azzerato e $CONV\_COMP$ viene impostato a uno. Questo è un bit di sola lettura.

$CSR0 < 2 >$ seleziona il *gate* dal pannello frontale o posteriore. Quando il bit è impostato a uno, viene utilizzato il *gate* del pannello frontale.

$CSR0 < 3 >$ indica che il buffer degli eventi è pieno (in logica negata, *Event Buffer not full*); quando il bit è a zero, il contatore degli eventi sarà a zero indicando che sedici eventi sono stati registrati nella memoria. Il bit viene chiamato $EVENT\_FULL^*$.

$CSR0 < 7 : 4 >$ indica il contatore degli eventi (0-15); il bit 4 è il meno significativo (LSB).

$CSR0 < 8 >$ *Clear Module*; quando viene scritto un uno, $CONV\_COMP$, $EVENT\_FULL^*$ e il contatore degli eventi vengono azzerati. Infine questo bit si azzera da solo.

$CSR0 < 9 >$ *Internal Test*; la scrittura a uno di questo bit causa la generazione di un *test-gate* di approssimativamente 650 ns e un *test-pulse* (un impulso per poter testare il corretto funzionamento dell'ADC) di circa tre quarti rispetto al fondoscala. Infine il bit si azzera automaticamente.

$CSR0 < 10 : 15 >$ non sono implementati.

2.3 Modulo TDC V1190A

Il modello V1190A è un Time to Digital Converter, cioè modulo che converte il tempo in valori digitali, con 128 ingressi. Il convertitore permette di catturare il tempo di arrivo dei segnali applicati agli ingressi. L'unità ospita quattro chips High Performance TDC, sviluppati dalla CERN/ECP-MIC Division. La risoluzione della misura può essere impostata a 100 ps, 200 ps o 800 ps. Il CERN/ECP-MIC HPTDC è un convertitore TDC multiuso, con 32 canali per chip. I chips possono essere abilitati per la rilevazione del fronte di salita e/o di discesa e della larghezza del segnale. L'acquisizione dati può essere programmata in "eventi" (il "trigger matching mode" con una finestra temporale programmabile chiamata match window) oppure in "continuos storage mode" (cioè in acquisizione continua). La programmazione del TDC è effettuata attraverso un microcontrollore che implementa un'interfaccia ad alto livello nei confronti dell'utente, in modo tale da mascherare l'hardware del TDC. La peculiarità di questo modello di TDC è la notevole versatilità, che però richiede la configurazione di un numero elevato di parametri per un corretto funzionamento, di seguito verranno descritte le funzionalità utilizzate nell'ambito dell'esperimento.

2.3.1 Trigger matching mode

Il *trigger matching mode* si basa sull'appartenenza (matching) di uno o più hits alla finestra temporale del trigger, chiamata *search window*. Un hit rappresenta la rilevazione di un evento, se vengono rilevati più eventi si hanno altrettanti hit (si è in presenza di un *multihit*). Il controllo di *matching* del trigger è gestito attraverso l'uso di quattro parametri programmabili (individuabili in figura 2.4):

a) match window,

b) window offset,

c) extra search margin,

d) reject margin.

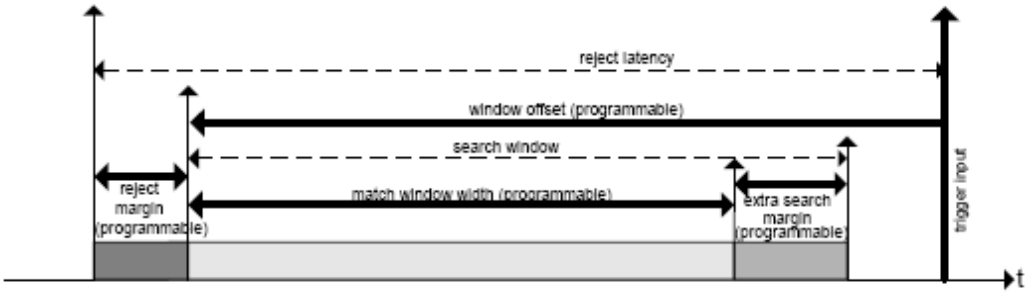


Figura 2.4: Diagramma temporale del Trigger Matching Mode

Una volta ricevuto un trigger input, vengono considerati hits validi tutti quegli hits che sono all'interno della *search window*, possono esserci più hit per ogni canale: in questo caso vengono assegnati all'evento nello stesso ordine di arrivo (*first come first served*). Il tempo registrato per ogni hit può essere un tempo assoluto (che fa riferimento ad un contatore interno al TDC) oppure riferirsi all'inizio della *match window*. In base ai valori assegnati a *window width* e *window offset* la finestra temporale può precedere l'*input trigger* oppure quest'ultimo può cadere dentro o precedere la finestra temporale. L'*extra search margin* è di default impostato a otto cicli di clock, ma può essere aumentato per quelle applicazioni che lavorano con impulsi molto lunghi, nelle quali può capitare che il fronte di salita dell'impulso cada nella *match window*, ma non il fronte di discesa. L'*extra search window* serve quindi ad assicurare che l'intero impulso, e non solo un suo fronte, venga rilevato dalla *search window*. Ogni volta che si verifica un segnale di trigger, gli hits accumulati dai canali vengono caricati nell'*Output Buffer* dove vengono organizzati in eventi. Ogni evento consiste di:

Il Global Header (32 bit) : che contiene gli indirizzi geografici e il contatore degli eventi (fino a 22 bit), vedi fig. 2.5.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	EVENT COUNT																						GEO				

Figura 2.5: Output Buffer: il *Global Header*

Gli eventi : accumulati dal TDC, ognuno dei quali composto come segue:

a) un TDC Header (se abilitato), vedi fig. 2.6;

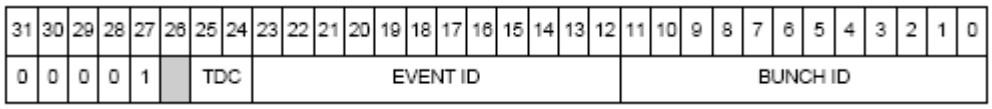


Figura 2.6: Output Buffer: il *TDC Header*

b) i tempi acquisiti dal TDC (catturati all'interno della *trigger matching window*), vedi fig. 2.7;

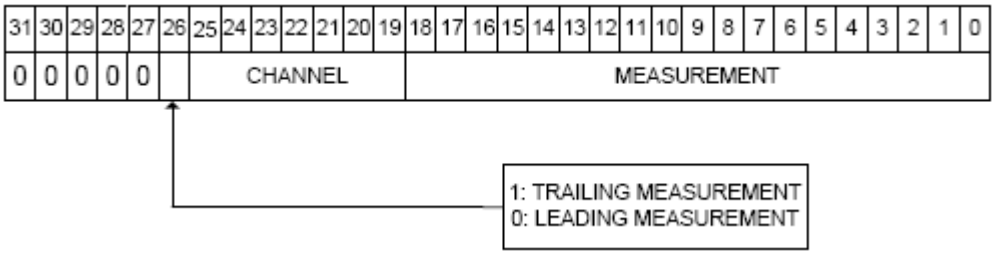


Figura 2.7: Output Buffer: le *TDC Measurement*

c) gli eventuali errori del TDC (se abilitati), vedi fig. 2.8;

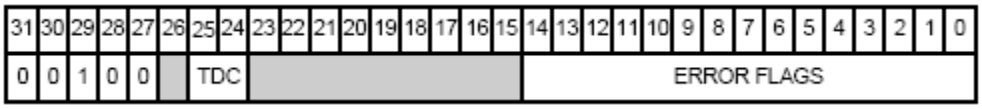


Figura 2.8: Output Buffer: il *TDC Error*

d) il TDC Trailer (se abilitato), vedi fig. 2.9;

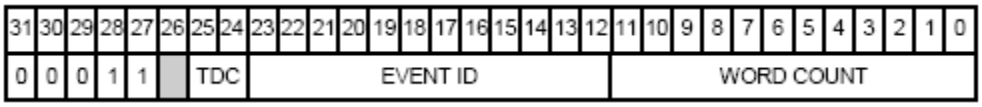


Figura 2.9: Output Buffer: il *TDC Trailer*

L'extended Trigger Time Tag (opzionale) : contiene il tempo di arrivo del trigger, relativo al *Count Reset* (LSB = 800 ns), vedi fig. 2.10.

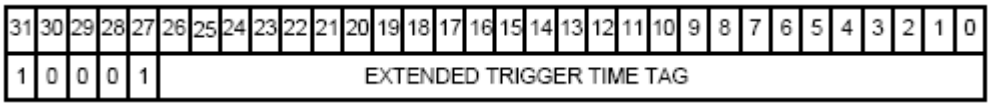


Figura 2.10: Output Buffer: l'*Extended Trigger Time Tag*

Il Trailer : che contiene gli indirizzi geografici, le *words* del contatore degli eventi (16 bit) e uno "status" (3 bit), vedi fig. 2.11.

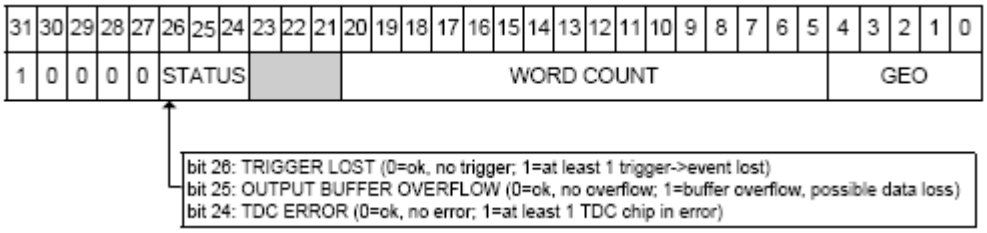


Figura 2.11: Output Buffer: il *Trailer*

2.3.2 Software clear

Il software clear è un particolare tipo di reset che non cancella la programmazione del TDC (un vero e proprio reset globale del TDC infatti pulisce tutti i buffer del TDC, inizializza tutti gli stati interni della macchina e i contatori ai loro stati iniziali e cancella tutti gli errori rilevati). Per effettuare un software clear è necessario effettuare un'accesso in scrittura all'indirizzo (Base Address + 0x1016), dove Base Address rappresenta l'indirizzo del modulo TDC.

2.4 Adattatore da VMEbus a PCI

L'hardware fondamentale per la comunicazione tra il crate VME e il pc è l'adattatore VME to PCI modello 618-3 della SBS Technologies (vedi figura 2.12). Esso è composto da una scheda VME collegata ad una scheda PCI da un cavo in fibra ottica, tramite le quali connette il bus VME con il bus PCI creando un bus virtuale che permette ai due sistemi di operare come se fossero un sistema unico. Si è scelto di utilizzare questo tipo di adattatore in virtù della comunicazione tramite fibra ottica, poiché si necessita di immunità al rumore e alte prestazioni. Dal lato

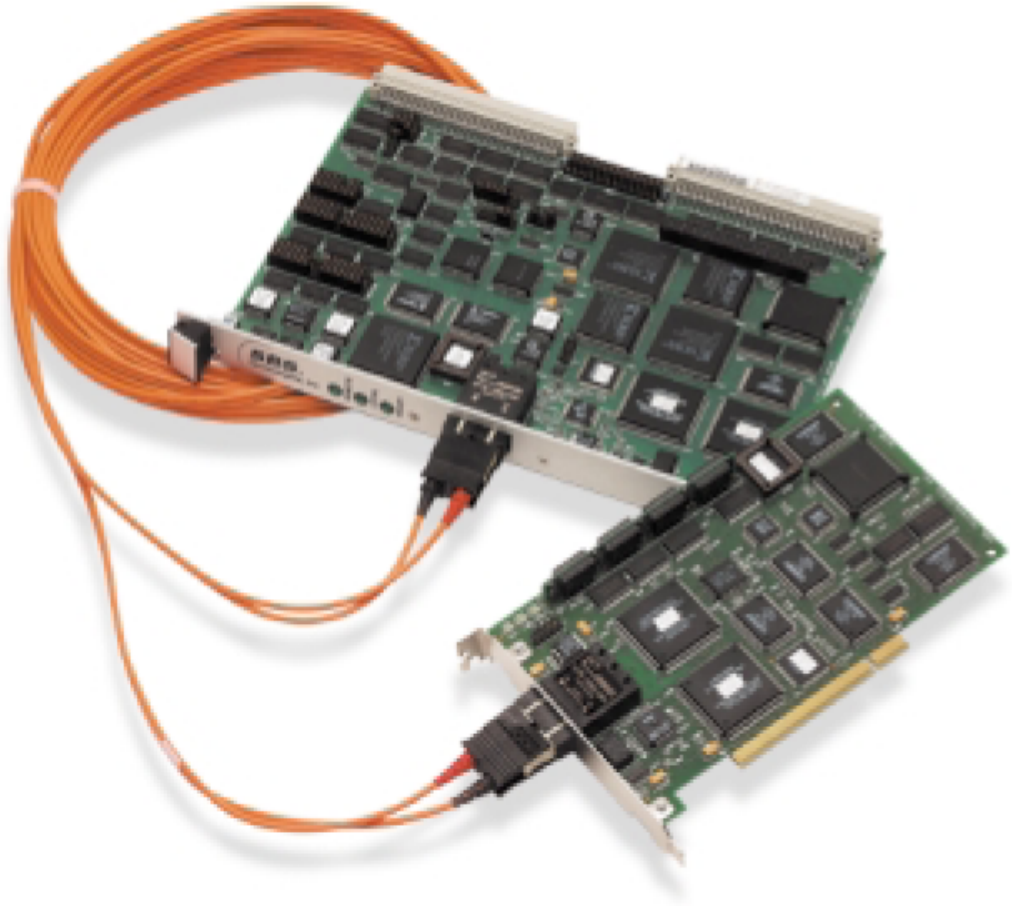


Figura 2.12: Modello 618-3 della SBS Technologies, adattatore VME - PCI

VME l'adattatore viene visto come un processore sul bus VME, mentre in realtà il processore è rappresentato dal PC su cui è installato l'adattatore PCI. Un PC standard con questo adattatore può quindi sostituire una scheda VME con processore dedicato, permettendo all'utente di avvalersi dei vantaggi di un software

pronto all'uso e dalla semplicità di utilizzo propria del personal computer.
L'adattatore supporta due metodi di comunicazione tra PCI e VME: Memory & I/O mapping e Direct Memory Access (DMA).
Nel primo metodo lo spazio degli indirizzi di destinazione è visto come uno spazio indirizzi addizionale del bus dell'host ottenuto attraverso mapping della memoria o di I/O. Il mapping prende un range definito di indirizzi inutilizzati della memoria nel bus dell'host e lo traspone nello spazio di indirizzi selezionato della memoria del bus di destinazione.
DMA, il secondo metodo di comunicazione, permette di trasferire ad alte velocità grosse quantità di dati tra i due sistemi con un piccolo sovraccarico del lavoro del processore. Il DMA engine legge i dati da un bus e li scrive sull'altro bus. Quando il trasferimento è completo, l'adattatore interrompe il processore. Sono supportati due modalità DMA: il controller mode e lo slave mode. Il primo usa il controller DMA dell'adattatore per abilitare i trasferimenti ad alta velocità da un sistema di memoria direttamente nell'altro sistema di memoria. Il processore può iniziare il trasferimento in entrambe le direzioni. Il controller DMA permette il trasferimento di dati tra la memoria PCI e la dual port RAM nel adattore VME. La modalità slave è il processo attraverso il quale un dispositivo VME usa l'adattatore per trasferire un blocco di dati direttamente nella memoria dell'host, ad una velocità che arriva a 13 MByte/s. Come la modalità controller DMA, slave DMA usa una piccolissima parte della potenza del processore durante il trasferimento. Può essere utilizzato un interrupt per avvertire il processo host quando il trasferimento in modalità slave è stato completato.
Si è scelto di utilizzare il metodo di mapping della memoria e dell'I/O poiché si ha una limitata quantità di dati da trasferire, inoltre non occorrono funzionalità "real-time" grazie al basso rate di eventi generati e soprattutto per la presenza di buffer multievento sui moduli di raccolta dati. Nel seguito verrà illustrata la caratteristica di questo metodo.

2.4.1 Mapping Registers

L'adattatore PCI è equipaggiato con i registri di Mapping, questo gli permette di far corrispondere ad una grande sezione di memoria locale contigua tante piccole sezioni di memoria remota non contigua. Questa caratteristica è molto importante poiché la maggior parte delle architetture moderne utilizzano come sistemi di gestione della memoria la memoria virtuale e la paginazione della memoria. Queste tecniche di gestione spesso soddisfano una richiesta di memoria da parte

di un'applicazione con piccole sezioni di memoria che sono sparse per tutto lo spazio di memoria. Il registro di mapping permette di rimappare queste aree non contigue di memoria in aree contigue nel bus remoto. Questo processo di mapping funziona sia per accessi da PCI a VME che per accessi da VME a PCI.
Verrà ora descritto il procedimento di inizializzazione dell'adattatore.

2.4.2 Inizializzazione

Prima che le funzioni dell'adattatore possano essere usate, è necessario eseguire i seguenti passi per inizializzare l'adattatore:

1. leggere il registro di stato locale (Local Status Register) per essere sicuri che il bit 0 sia a zero. Questo indica che il crate VME è acceso e che il cavo è connesso. Se Una di queste due condizioni si rivelasse falsa, ogni tentativo di accedere a risorse remote ritornerebbe un errore di interfaccia.

2. leggere il registro di stato remoto (Remote Status Register) e scartare il risultato. Questa operazione pulisce il cavo da errori causati dalla transizione in "power on", cioè da eventuali errori formatisi all'accensione del crate.

3. settare il bit 7 del registro di comando locale (Local Command Register) per pulirlo da eventuali errori registrati durante l'accensione del crate.

4. leggere il registro di stato locale (Local Status Register) per essere sicuri che nessun bit di errore sia alto.

Con la conclusione di questo capitolo si termina la descrizione della parte *hardware* dell'esperimento. Dal capitolo successivo si comincerà a trattare gli aspetti riguardanti il *software*.

Capitolo 3

Libreria per acquisizione dati

In questo capitolo si descriveranno le funzioni necessarie alla comunicazione con il crate VME e relativi moduli. Queste funzioni sono state ottenute a partire da metodi forniti dalle case produttrici dell'hardware e da un programma scritto in ambiente Visual Studio che effettua una lettura di n eventi dall'ADC e dal TDC e stampa i risultati su terminale. Questi metodi, sono poi stati inseriti in una libreria dinamica chiamata "VMElib.dll", utilizzata dagli strumenti virtuali (VI) creati nell'ambiente di programmazione LabVIEW (descritto nella sezione 4.1) per interfacciarsi con l'hardware.
L'applicazione principale non può essere a conoscenza del contenuto della libreria, quindi utilizza le funzioni di connessione, lettura e pulizia senza sapere come esse siano in realtà implementate. Se in futuro ci sarà necessità di sostituire l'hardware con una tecnologia diversa, basterà sostituire la libreria dinamica attuale con un'altra che contenga funzioni con la stessa firma, senza dover modificare il livello "più alto" dell'applicazione.

3.1 Scelte implementative

Dovendo utilizzare la libreria in un ambiente esterno, si è dovuto trovare compromesso per quanto riguarda il passaggio di parametri da LabVIEW alle funzioni di libreria, e viceversa. Infatti originariamente i metodi di lettura dei moduli ritornavano una matrice di interi, ma LabVIEW ammette come valori di ritorno di funzioni solo interi, puntatori ad interi, o puntatori a stringhe. Si è scelto perciò di far restituire dalle funzioni di lettura della libreria i valori letti formattandoli in un'unica stringa. In LabVIEW si effettuerà poi il *parsing* della stringa per estrapolare i dati.

Alcune funzioni richiedono un tipo di dato non primitivo tra i parametri. Questo tipo di dato, bt_desc_t, è un puntatore ad una struttura ed identifica una connessione al crate VME: viene restituito dal metodo Connection_and_Setup_ADC_TDC, e successivamente passato come argomento agli altri metodi implementati nella libreria, poiché è indispensabile identificare la connessione sulla quale bisogna effettuare le procedure. bt_desc_t è però definito all'interno della libreria bit3_984 fornita dal costruttore della scheda PCI che si interfaccia con il crate VME. Ciò può diventare un problema nel caso le funzioni vengano richiamate da un'applicazione che, come nel nostro caso, non conosce la struttura di bt_desc_t: si è deciso comunque di non modificare la firma di tali funzioni, poiché questo tipo di dato non viene mai letto dall'applicazione principale, ma solamente trasmesso come argomento. Verrà trattato come un intero senza segno a 32 bit, essendo un indirizzo.

3.2 Libreria a collegamento dinamico

In informatica, una dynamic-link-library (termine inglese, tradotto in italiano libreria a collegamento dinamico) è una libreria software che non viene collegata staticamente ad un eseguibile in fase di compilazione, ma che viene caricata dinamicamente in fase di esecuzione. Queste librerie sono note con l'acronimo DLL, che è l'estensione del file che hanno nel sistema operativo Microsoft Windows, o anche con il termine librerie condivise (da shared library, usato nella letteratura dei sistemi Unix). Nei sistemi che usano ELF come formato dei file eseguibili, come ad esempio Solaris o Linux, sono anche note come ".so", abbreviazione di Shared Object.

3.2.1 Vantaggi e svantaggi

La separazione del codice in librerie a collegamento dinamico permette di suddividere i programmi in parti concettualmente separate, che verranno caricate solo se effettivamente necessarie. Inoltre, una singola libreria può essere caricata in memoria una sola volta e utilizzata da più programmi, il che permette di risparmiare le risorse del sistema. Un altro vantaggio è la possibilità di aggiornare un programma modificando solo le DLL: inserendo una versione diversa della DLL, che contiene ad esempio dei *bug fix*, tutti i programmi che la usano saranno automaticamente "aggiornati" senza bisogno di essere ricompilati.
Il principale svantaggio è legato al fatto che una nuova versione di una DLL

potrebbe effettuare dei cosidetti *breaking changes* in modo volontario o a causa di bug. Un *breaking change* è una modifica del comportamento di una funzione che la rende non più compatibile con le convenzioni in uso (ad esempio, una funzione che prima restituiva NULL in caso di errore, nella versione aggiornata restituisce un valore non nullo).

3.2.2 Creazione della libreria

Per ottenere il file .dll è stato creato un progetto DLL in ambiente Visual Studio, includendo tutti i file forniti da SBS Tecnologies per la comunicazione tramite bus VME e i files che contengono funzioni specifiche per ogni modulo VME utilizzato. Infine sono state implementate nel file principale del progetto le funzioni che verranno descritte più avanti, anteponendo alla loro firma: __declspec(dllexport).

3.3 Funzioni fornite dalla DLL

In questa sezione vengono elencate le funzioni disponibili utilizzando la libreria dinamica, con le relative firme e descrizioni. Come si può notare da figura 3.1, tali funzioni sono: Connection_and_Setup_ADC_TDC, read_Event_ADC, my_read_One_Event_TDC, clear_TDC, clear_ADC, Close. In figura 3.1 vengono illustrate assieme ai parzmetri in ingresso e al valore restituito.

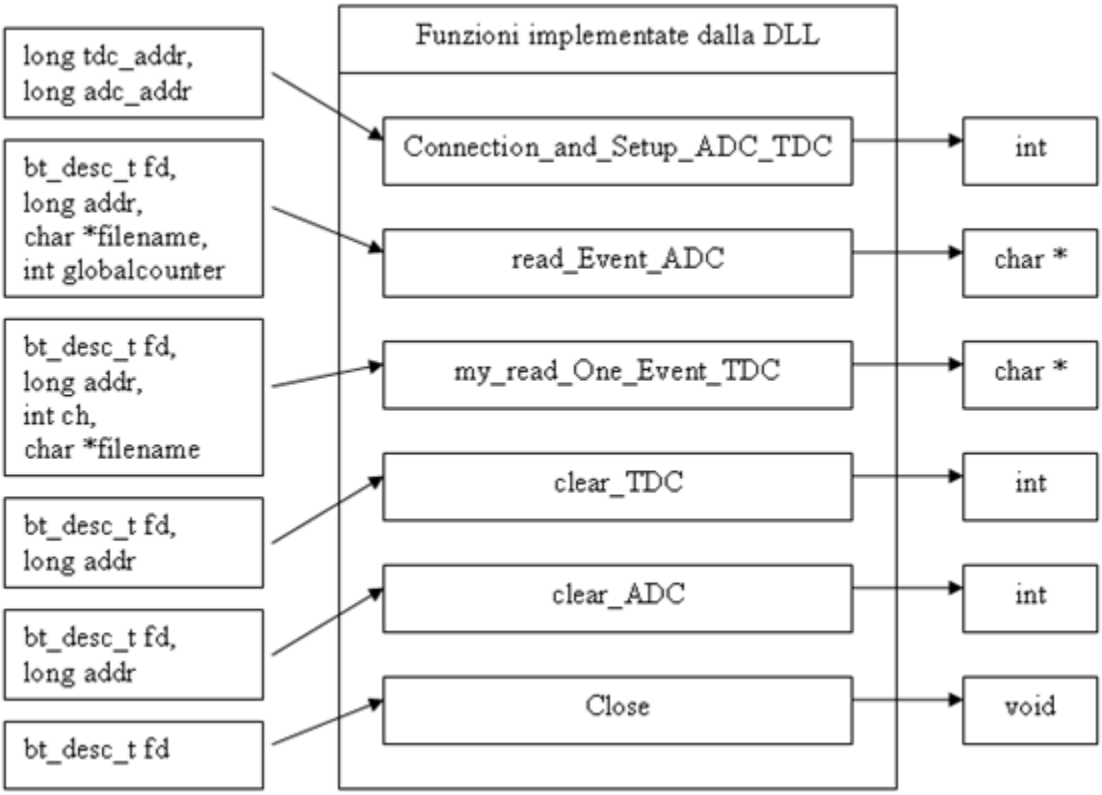


Figura 3.1: Schema che rappresenta le funzioni fornite dalla libreria

3.3.1 Connection_and_Setup_ADC_TDC

Firma della funzione:

int Connection_and_Setup_ADC_TDC(long tdc_addr, long adc_addr);

Richiede come parametri:

- l'indirizzo del modulo TDC;
- l'indirizzo del modulo ADC;

Ritorna:

L'identificatore della connessione in caso di successo, "-1" se c'è stato un errore durante la connessione al crate VME, "-2" se si è verificato stato un errore durante il setup del TDC o "-3" se c'è stato un errore durante il setup dell'ADC.

Procedura:

Effettua una connessione al crate VME tramite la funzione Open_Vme_A32, passando per indirizzo fd, un oggetto di tipo bt_desc_t. Durante la connessione viene impostata la modalità di accesso A32 (cioè l'utilizzo di indirizzi a 32 bit) e viene anche inizializzato l'adattatore PCI to VME (vedi 2.4.2). Se la connessione va a buon fine, fd punterà ad una struttura che identifica la connessione. Successivamente configura il modulo TDC attraverso la funzione Setup_TDC, passando come argomenti fd e l'indirizzo del modulo TDC.
Il setup viene effettuato eseguendo i seguenti passi:

1. reset software del modulo;
2. il modulo viene impostato in Trigger Matching mode;
3. viene impostata la window width;
4. viene impostato il valore di offset;
5. viene disabilitato il trigger time subtraction;
6. vengono abilitati tutti i canali;
7. l'edge detection viene impostata come leading + trailing;
8. il tempo di risoluzione viene impostato a 100ps;

9. vengono abilitati l'header e il trailer;
10. il massimo numero di hit viene impostato a NO LIMIT;
11. viene abilitata l'error mask TDC;

Successivemente a questo passo viene configurato il modulo ADC attraverso la funzione Setup_ADC, gli argomenti passati sono fd e l'indirizzo del modulo ADC. Contrariamente a quanto avvenuto per il TDC, il setup dell'ADC è composto semplicemente da un clear, effettuato chiamando il metodo clear_ADC.

3.3.2 read_Event_ADC

Firma della funzione:

char* read_Event_ADC(bt_desc_t fd, long addr, char *filename, int global-counter);

Richiede come parametri:

- l'identificatore della connessione;
- l'indirizzo del modulo ADC;
- il path assoluto che identifica il file di testo su cui stampare le letture degli eventi;
- il valore del contatore degli eventi;

Ritorna:

Una stringa contenente, in caso di successo, nella prima riga il numero di eventi letti (possono anche essere zero) e nelle otto successive i valori letti dagli otto canali. In caso di errore invece la stringa conterrà "-1".

Procedura:

Inizialmente viene inizializzata una matrice 16x8 che conterrà l'output buffer del modulo ADC. Le 16 righe corrispondono al numero massimo di eventi che il modulo può memorizzare; per ogni riga sono presenti 8 elementi che rappresentano gli otto canali dell'ADC. Attraverso il metodo L1182_read viene letto l'output buffer del modulo e riempita la matrice passata come argomento. Come argomento viene anche passato un intero che conterrà, al ritorno del metodo

L1182_read, il numero degli eventi letti. Se non viene letto nessun evento la funzione termina immediatamente restituendo "0", altrimenti il numero degli eventi letti viene inserito nella stringa str_ret. Infine i dati precedentemente acquisiti vengono letti dalla matrice e stampati nel file di testo identificato dalla stringa filename e in str_ret. Per quanto riguarda il file sono state aggiunte delle descrizioni ad ogni dato per rendere il contenuto più leggibile, poiché è destinato ad uso utente. Nella stringa invece vengono stampati solamente i valori letti dalla matrice, uno per ogni riga.

3.3.3 my_read_One_Event_TDC

Firma della funzione:

char* my_read_One_Event_TDC(bt_desc_t fd, long addr, int ch, char *filename);

Richiede come parametri:

- l'identificatore della connessione;
- l'indirizzo del modulo TDC;
- il numero del canale di riferimento;
- il path assoluto che identifica il file di testo su cui stampare le letture degli eventi;

Ritorna:

Una stringa contenente, in caso di successo, nella prima riga il numero di eventi letti (possono anche essere zero) e nelle successive i valori letti dai canali (secondo una determinata formattazione spiegata nel seguito). In caso di errore invece la stringa conterrà "-1".

Procedura:

Con la configurazione attuale del modulo TDC, vengono utilizzati al massimo 16 canali (anche se il modulo può arrivare fino a 128 canali). Inoltre per la tipologia dei dati che ci interessa acquisire, ci aspettiamo raramente casi di multihit, cioè di più valori per un singolo evento. Quindi è stato pensato che il numero massimo di hit per ogni evento e per ogni canale sia di cinque, ovviamente un caso in cui si abbia effettivamente cinque hit per un solo evento sarebbe anomalo.

Questi due valori sono definiti come MAX_CH_TDC per quanto riguarda il numero di canali, e MAX_HIT_TDC per quanto riguarda il numero massimo di hit. Viene perciò utilizzata una matrice 16x5 per contenere i dati letti dal TDC, ed inizializzata a -1 in quanto i tempi letti dal TDC sono sempre positivi. Successivamente vi è un ciclo do-while che controlla DREADY, il primo bit del registro di stato, il quale indica la presenza di dati da leggere. I casi di uscita dal ciclo sono due: il primo è che il bit sia a 1, e il secondo è lo scadere di un timeout di 10 ms (EVENT_TIMEOUT). Il controllo del timeout è necessario in quanto se per errore, come per esempio il dimenticarsi di attaccare il cavetto al canale del modulo TDC, non si avessero valori in ingresso si creerebbe un ciclo infinito. Una volta certi che ci siano dati in ingresso si passa alla loro acquisizione attraverso un operazione di lettura a 32 bit (il registro di output è composto da parole di 32 bit) contenuta in un altro ciclo do-while che viene eseguito finchè non sono stati letti tutti i dati o è scaduto un timeout di 5 secondi (READ_TIMEOUT). Il valore letto viene assegnato ad un intero, essendo anch'esso di 32 bit. Ad ogni ciclo viene controllato il tipo di dato letto, attraverso un costrutto ”switch” e delle maschere applicate all'intero (i primi 5 bit identificano il tipo di word), infatti ad ogni evento oltre ai valori acquisiti sono associati anche vari header e trailer. Se il valore è di tipo TDC_DATA si va ad inserire il valore nella matrice. Non appena viene individuato il GLOBAL_TRAILER significa che il buffer di output è stato completamente letto. Infine i valori salvati nella matrice vengono stampati nel file di testo (identificato dal contenuto della stringa ”filename”) e su di una stringa temporanea. Bisogna tener conto che i tempi ottenuti dal TDC sono tempi assoluti, riferiti ad un contatore interno al modulo, mentre ai fini del nostro esperimento ci interessa avere la differenza tra i tempi dei vari canali e il tempo ottenuto dal canale di riferimento. Quindi i tempi contenuti nel file e nella stringa sono il risultato di questa differenza, e potrebbero essere anche negativi (se per esempio il tempo su un determinato canale è minore del tempo sul canale di riferimento). Per quanto riguarda il file identificato da ”filename”, sono state aggiunte delle descrizioni ad ogni dato per rendere il contenuto più leggibile, poiché è destinato ad uso utente.
Nella stringa temporanea invece vengono stampati in ordine il canale, il numero di hit e il tempo relativo per ogni evento, separati da uno spazio. Per ogni riga si trova un singolo evento, secondo la seguente formattazione:

CANALE NUMERO_HIT TEMPO_RELATIVO

Infine nella stringa str_ret viene inserito il numero degli eventi letti, seguito dal carattere "a capo" e dalla stringa temporanea precedentemente creata. Sarà str_ret a essere restituita al programma chiamante.

3.3.4 clear_TDC

Firma della funzione:

int clear_TDC(bt_desc_t fd, long addr);

Richiede come parametri:

- l'identificatore della connessione;
- l'indirizzo del modulo TDC;

Ritorna:

"0" in caso di successo, "-1" in caso di errore.

Procedura:

All'interno di questo metodo viene generato un "software clear" (vedi 2.3.2) sul modulo TDC identificato dall'indirizzo addr.

3.3.5 clear_ADC

Firma della funzione:

int clear_ADC(bt_desc_t fd, long addr);

Richiede come parametri:

- l'identificatore della connessione;
- l'indirizzo del modulo ADC;

Ritorna:

"0" in caso di successo, "-1" in caso di errore.

Procedura:

La funzione richiama il metodo L1182_clear. All'interno di questo metodo vengono effettuati due clear in sequenza. Queste operazioni di clear, successivamente alla pulizia del buffer di output, abilitano il gate di entrata dei dati. Effettuare due clear è necessario in quanto nell'ADC 1182 l'abilitazione del gate successiva al clear può generare un *glitch* fantasma, che verrà poi scambiato per un evento reale.
Infatti durante la lettura dell'ADC il gate di entrata dei dati è chiuso, successivamente con il primo clear viene aperto il gate e questo può causare il così detto *glitch* fantasma. Al secondo clear il gate è già aperto quindi l'unica operazione che viene effettivamente effettuata è la sola pulizia del buffer di output.
Questa caratteristica del gate è stata notata durante lo studio del modulo 1182 e non è documentata sui manuali reperibili in rete.

3.3.6 Close

Firma della funzione:

```
void Close(bt_desc_t fd);
```

Richiede come parametro:

- l'identificatore della connessione.

Procedura:

All'interno del metodo viene richiamata la funzione Close_Vme, che a sua volta termina la connessione identificata da fd.

Capitolo 4

Applicazione di acquisizione dati in LabVIEW

La prima parte di questo capitolo è dedicata alla descrizione dell'ambiente di programmazione utilizzato per lo sviluppo dell'applicazione finale, e i motivi per cui questo ambiente è stato preferito rispetto ad altri. Nella seconda parte verrà invece descritta l'applicazione finale, in che modo è stata creata e quali servizi offre.

4.1 LabVIEW

Come ambiente di programmazione è stato scelto LabVIEW, nel seguito verrà descritto brevemente e verranno esposte le motivazioni per cui è stato scelto.

4.1.1 Programmazione G

Il linguaggio di programmazione usato in LabVIEW si distingue dai linguaggi tradizionali perché grafico, e per questa ragione battezzato G-Language (Graphic Language). Un programma o sottoprogramma G, denominato VI (Virtual Instrument), non esiste sotto forma di testo, ma può essere salvato solo come un file binario, visualizzabile e compilabile solo da LabVIEW.
La definizione di strutture dati ed algoritmi avviene con icone e altri oggetti grafici, ognuno dei quali incapsula funzioni diverse, uniti da linee di collegamento (wire), in modo da formare una sorta di diagramma di flusso. Tale linguaggio viene definito dataflow (flusso di dati) in quanto la sequenza di esecuzione è definita e rappresentata dal flusso dei dati stessi attraverso i fili monodirezionali

che collegano i blocchi funzionali. poiché i dati possono anche scorrere in parallelo attraverso blocchi e fili non consecutivi, il linguaggio realizza spontaneamente il multithreading senza bisogno di esplicita gestione da parte del programmatore.

4.1.2 Dettagli dei VI

Nell'ambiente di sviluppo, i VI constano di tre componenti principali:

il pannello frontale: interfaccia utente del VI. Si realizza con controlli e indicatori, che costituiscono i terminali interattivi d'ingresso e d'uscita, rispettivamente. I controlli sono matrici, manopole, potenziometri, pulsanti, quadranti e molti altri; simulano i dispositivi d'ingresso degli strumenti e forniscono dati allo schema a blocchi del VI. Gli indicatori sono grafici, tabelle, LED, termometri e molti altri; simulano i dispositivi d'uscita degli strumenti e visualizzano i dati che lo schema a blocchi acquisisce o genera.

lo schema a blocchi: è il diagramma di flusso che rappresenta il codice sorgente in formato grafico. Gli oggetti del pannello frontale appaiono come terminali di ingresso o uscita nello schema a blocchi. Gli oggetti dello schema a blocchi comprendono:

- terminali
- funzioni
- costanti
- strutture
- chiamate ad altri VI (subVI)
- fili di collegamento
- commenti testuali

il riquadro connettori: ogni VI puó essere a sua volta utilizzato come subVI (o sottoVI) e comparire all'interno dello schema a blocchi di altri VI, proprio come una qualsiasi funzione, e come tale puó avere ingressi e uscite a cui collegare le linee di flusso. Il riquadro connettori serve appunto a definire qual'è l'aspetto del VI quando appare come subVI in uno schema a blocchi: che facciata ha l'icona, ma soprattutto come e dove vanno collegate le linee per permettere il passaggio dei dati. In generale con pochi passaggi ogni controllo puó essere associato a un ingresso e ogni indicatore puó essere associato a un'uscita.

4.1.3 perché LabVIEW?

La semplicità di programmazione (abbastanza intuitiva in quanto modellata su un diagramma di flusso), la semplicità di utilizzo (l'utente finale dispone di uno strumento virtuale disegnato sullo schermo del computer) e la grande versatilità, hanno reso LabVIEW molto impiegato e diffuso nell'ambito dell'acquisizione dei dati e nel loro controllo nei processi industriali, nonchè nel campo della ricerca scientifica. Le caratteristiche appena descritte rappresentano il motivo principale per cui la nostra scelta è ricaduta su questo linguaggio.

4.2 Utilizzo della DLL in Labview

In questa sezione viene illustrato il procedimento tramite il quale è stato possibile creare gli strumenti virtuali utilizzabili in LabVIEW, che implementano le funzioni fornite dalla DLL.

4.2.1 Call Library Function Node

La Call Library Function Node (visibile in figura 4.1) è uno strumento virtuale messo a disposizione dall'ambiente Labview, viene utilizzato per richiamare una DLL o una libreria condivisa.

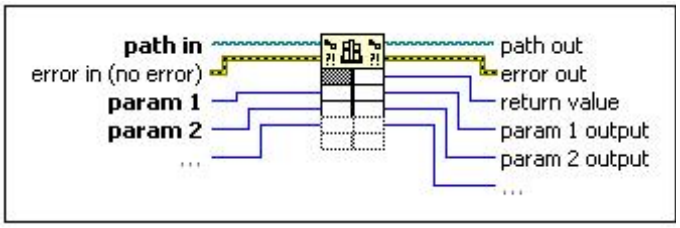


Figura 4.1: Schema dello strumento virtuale Call Library Function Node

La Call Library Funciton Node riconosce la maggior parte degli standard per la creazione di DLL. Ogni istanza di Call Library Function Node identifica una funzione della libreria, inoltre vanno specificati i tipi degli argomenti assegnati come parametri e il tipo dell'oggetto restituito, in modo tale da ricreare la firma della funzione specificata. Un esempio è illustrato in figura 4.2.

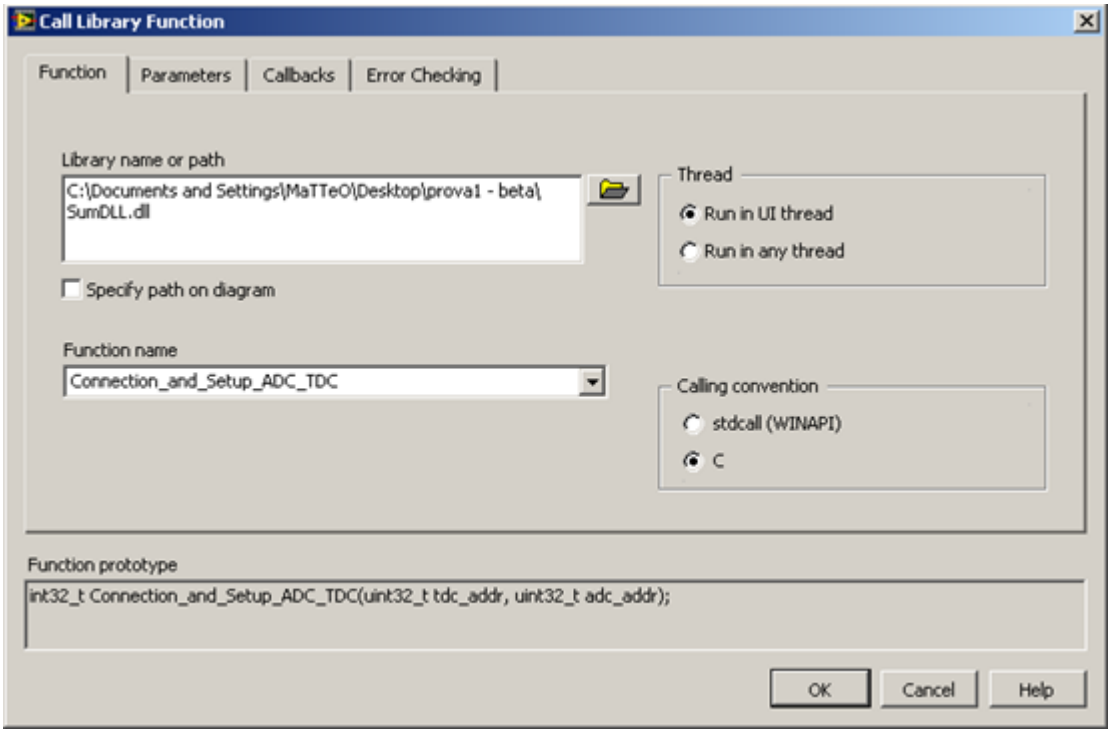


Figura 4.2: Finestra di configurazione della Call Library Function Node

4.2.2 Relizzazione strumenti virtuali

poiché, come già spiegato, un oggetto Call Library Function Node puó implementare una sola funzione, si è deciso di creare uno strumento virtuale per ogni funzione e infine aggiungere questi strumenti nel progetto finale. Ogni strumento virtuale contiene un'istanza di Call Library Function Node che implementa una funzione della DLL, ed inoltre vanno aggiunti gli oggetti Numeric Control o String Control (in base al tipo) ad ogni parametro che richiede la funzione implementata: se per esempio richiede come parametro una stringa, verrà collegato un oggetto String Control. Questo permette di poter impostare i valori dei parametri dello strumento virtuale, altrimenti verrebbe richiamata la funzione senza peró la possibilità di indicare alcun parametro. Infine va aggiunto un oggetto Numeric Indicator e String Indicator se la funzione restituisce un valore, in questo modo noi possiamo leggere quel valore all'esterno dello strumento virtuale che implementa la funzione. Un esempio chiarificatore è presente in figura 4.3.

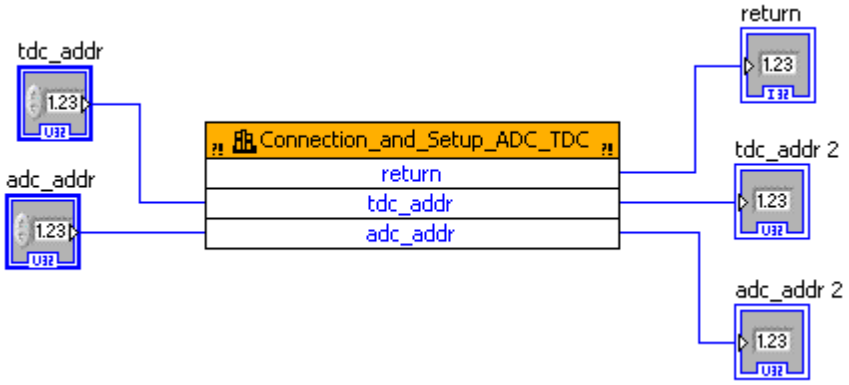


Figura 4.3: Schema circuitale del VI "Connection & Setup"

Come si puó notare da figura 4.3, sulla sinistra vi sono tdc_addr e adc_addr, due Numeric Control che permettono di assegnare rispettivamente l'indirizzo del modulo TDC e l'indirizzo del modulo ADC. Sulla destra vi sono invece tre oggetti: il primo è return, un Numeric Indicator che permette di leggere il valore restituito dalla funzione, gli altri due Numeric Indicator permettono di leggere i due parametri assegnati alla funzione dopo che quest'ultima è terminata. Ció risulta utile nel caso questi valori vengano modificati all'interno della funzione (non devono essere "passati" per valore ma per indirizzo), ed è un modo per poter permettere alla funzione di restituire piú valori di ritorno.

Nel seguito verrà descritta l'applicazione finale, che è composta da tre parti: l'inizializzazione, l'acquisizione dati e l'elaborazione grafici (come illustrato in figura 4.4).

Figura 4.4: Rappresentazione schematica delle tre macro sequenza dell'applicazione

4.3 Inizializzazione

All'avvio dell'applicazione vengono settate tutte le variabili, e viene eseguito un ciclo while che termina alla pressione del pulsante START. Ció permette all'utente di scegliere i canali desiderati e eventualmente di modificare valori già settati di default. Una volta premuto il pulsante START vengono letti i valori delle due stringhe rappresentanti gli indirizzi del modulo ADC e del modulo TDC, interpretati come stringhe esadecimali e convertiti in interi. A questo punto viene effettuata la connessione e il setup dei moduli utilizzando il VI "connect & setup", successivamente utilizzando i VI "clear ADC" e "clear TDC" vengono inizializzati i moduli. Nel caso in cui almeno uno di questi tre VI restituisca un errore, viene stampato il tipo di errore nel campo Finestra STATUS e l'applicazione termina. In caso invece di esito positivo nel campo Finestra STATUS viene stampata la stringa "connessione stabilita" e si da inizio all'acquisizione. In figura 4.5 è presente il frammento di diagramma a blocchi rappresentante l'inizializzazione e configurazione dell'applicazione.

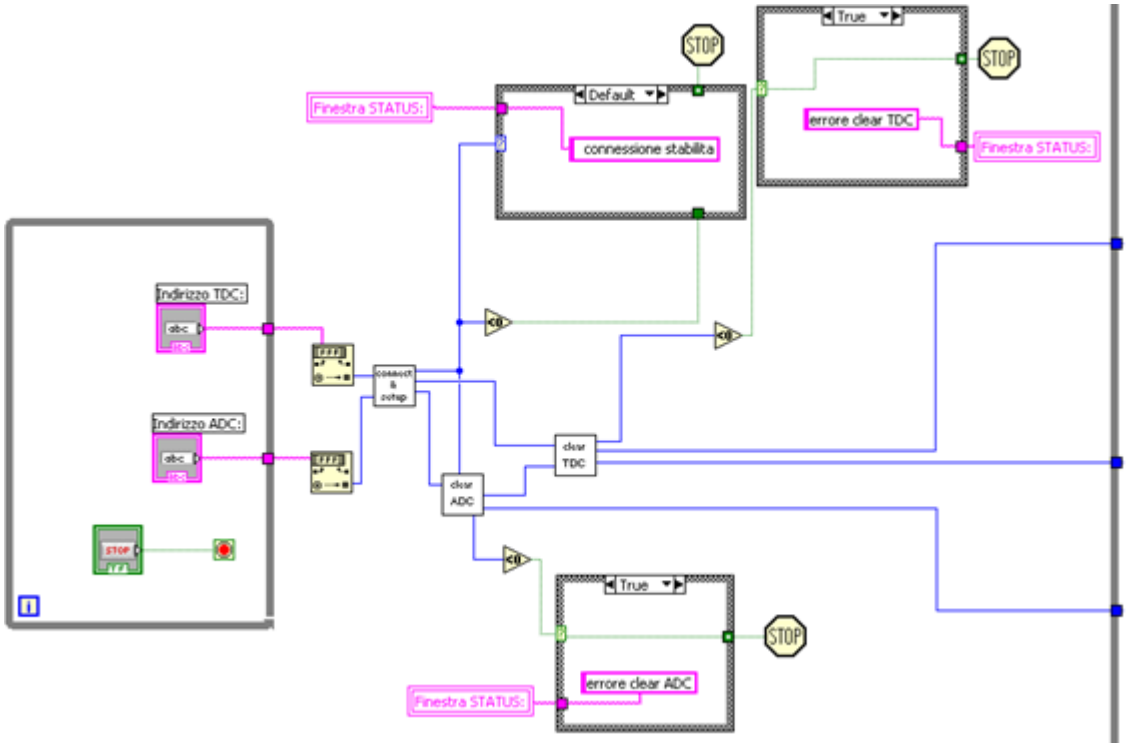


Figura 4.5: Frammento di diagramma a blocchi rappresentante l'inizializzazione e configurazione dell'applicazione

4.4 Acquisizione dati

L'acquisizione viene fatta all'interno di un ciclo while, la cui condizione è che il numero di eventi acquisiti sia minore o uguale al numero massimo di eventi da acquisire (il contenuto del ciclo while è rappresentato in figura 4.6). La prima operazione che viene eseguita ad ogni ciclo è l'esecuzione del VI "read ADC", che restituisce una stringa contenente il numero di eventi e i valori letti. Si effettua l'analisi della stringa estraendone il numero degli eventi, il resto della stringa viene passato alla parte "elaborazione grafici". Se il numero degli eventi è superiore a zero allora si incrementa il contatore degli eventi, nel caso invece in cui sia maggiore di uno si è in presenza di un multihit e viene incrementato il contatore "multihit ADC". Un multihit rappresenta un caso anomalo nell'esperimento che si sta conducendo, la probabilità che se ne ottenga uno è molto piccola ma bisogna comunque gestire questa situazione: si è deciso di rappresentare sugli istogrammi solamente il primo valore dell'ADC e i tempi letti dal TDC in riferimento al primo valore letto dall'ADC, poiché si suppone che gli eventi successivi al primo siano dati non di interesse ai nostri scopi. Se invece il numero degli eventi letti è -1 significa che all'interno della funzione richiamata si è verificato un errore, si gestisce la situazione come se non fossero stati letti eventi. Successivamente alla lettura dell'ADC viene effettuato un ciclo di letture del TDC (tante letture quanti sono gli eventi letti dal modulo ADC) utilizzando il VI "read TDC", all'esterno di questo ciclo viene poi restituito un *array* di stringhe, di cui viene tenuta solo la prima e passata alla parte "elaborazione grafici". Infine vengono svuotati i buffer di output dei moduli richiamando i VI "clear ADC" e "clear TDC". In figura 4.6 è presente il frammento di diagramma a blocchi rappresentante l'acquisizione dati dai moduli ADC e TDC.

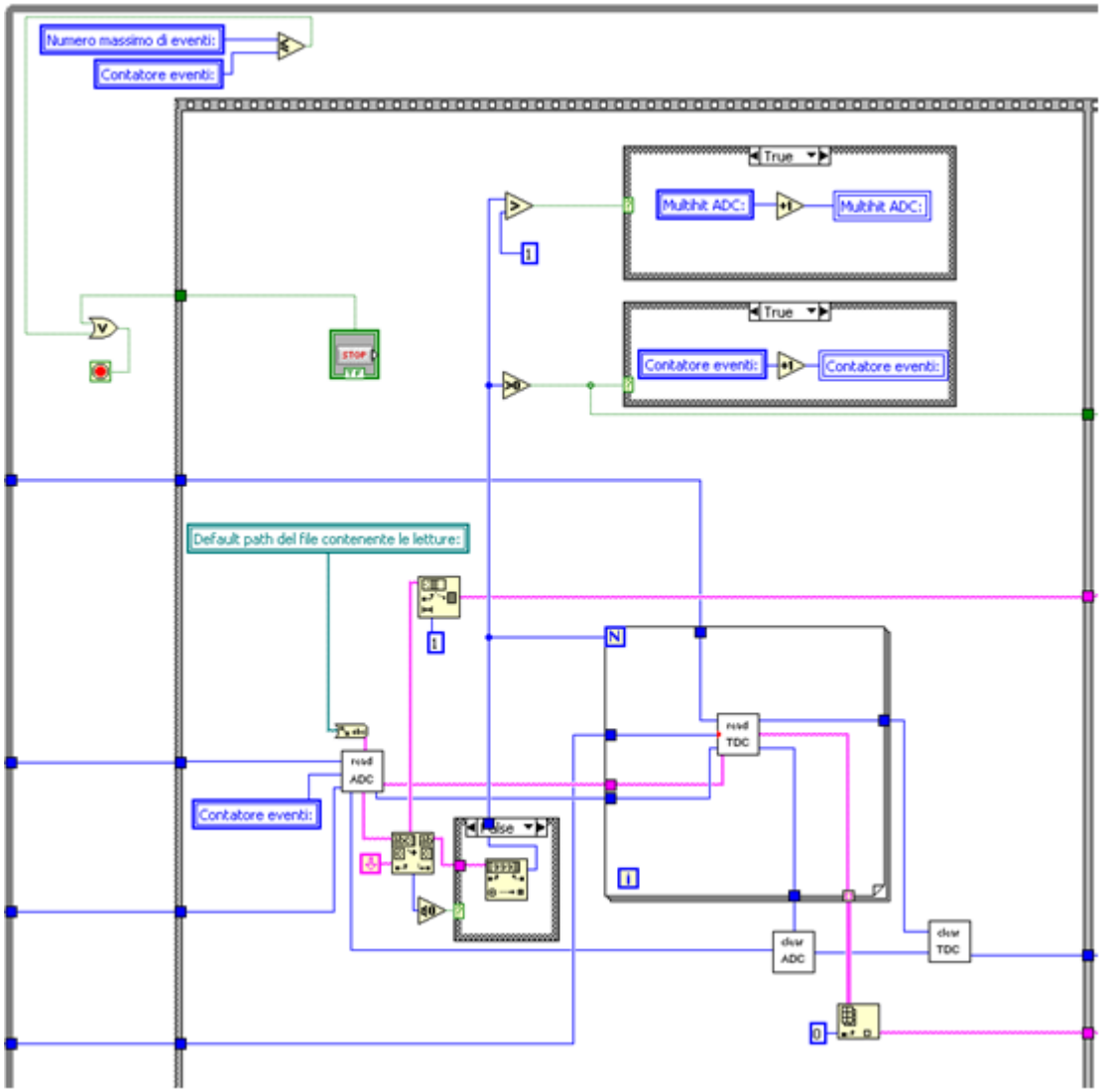


Figura 4.6: Frammento di diagramma a blocchi rappresentante l'acquisizione dati dai moduli ADC e TDC

4.5 Elaborazione grafici

Terminata la fase di acquisizione, inizia la seconda parte del ciclo while esterno: la lettura dei valori dai file temporanei creati dalle funzioni read_Event_ADC e my_read_One_Event_TDC. Questa fase si divide in due parti: la creazione dei due grafici relativi ai valori letti dal modulo ADC e la creazione dei due grafici relativi ai tempi letti dal modulo TDC.

4.5.1 Grafici dei valori acquisiti dal modulo ADC

Viene ricevuta dalla parte di "acquisizione" la sottostringa contenente i valori letti dal modulo ADC e ha inizio un for di otto cicli, essendo otto i canali del modulo ADC, ed ad ogni ciclo viene letta una riga del file. In base alla formattazione della stringa si ha un valore per ogni riga: la prima riga contiene il valore del primo canale, la seconda riga il valore del secondo canale, e cosí via. Se il canale letto è uno dei due selezionati nella fase di inizializzazione allora viene inserito il valore nel grafico. Nel caso di un multihit, ci saranno otto valori (quindi otto righe) per ogni evento, ma come spiegato nella sezione 4.4 consideriamo solo il primo evento. In figura 4.7 è presente il frammento di diagramma a blocchi rappresentante la realizzazione dei due grafici rappresentanti i valori acquisiti dal modulo ADC.

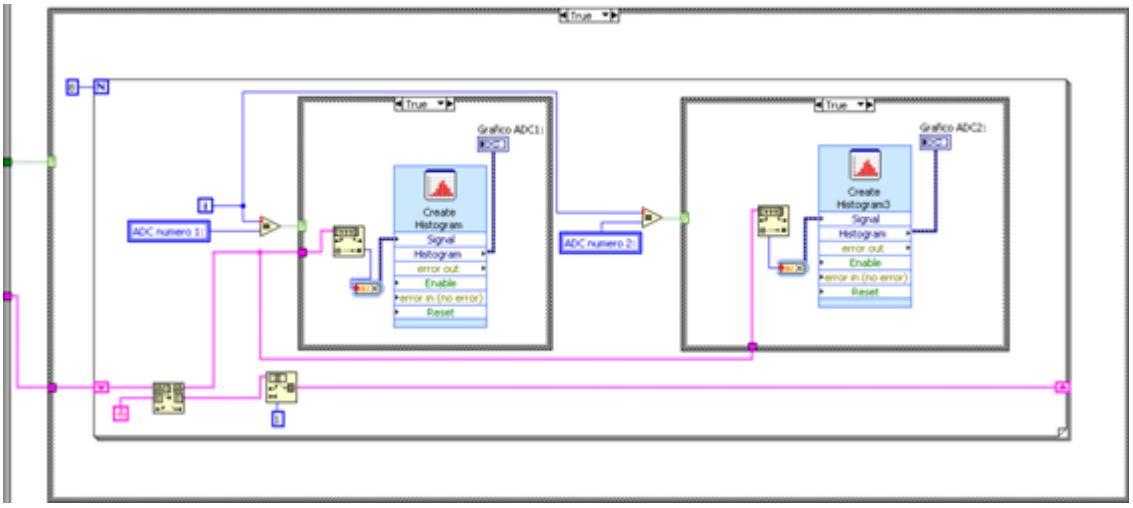


Figura 4.7: Frammento di diagramma a blocchi rappresentante la realizzazione dei due grafici rappresentanti i valori acquisiti dal modulo ADC

4.5.2 Grafici dei tempi acquisiti dal modulo TDC

Viene ricevuta dalla parte di "acquisizione" la sottostringa contenente i valori letti dal modulo TDC e ha inizio un ciclo for di n cicli, dove n è il numero di eventi catturati dal TDC (questo valore viene restituito da my_read_One_Event_TDC). Ad ogni ciclo viene letta una riga e ne viene effettuato il *parsing*, che consiste in CANALE NUMERO_HIT TEMPO_RELATIVO (vedi funzione nella sezione 3.3.3). Successivamente vi è una procedura parallela per entrambi i grafici: viene controllato se il canale indicato nella riga coincide con il canale selezionato per il grafico, in caso affermativo viene inserito il tempo nel grafico. Visto che per quanto rigurda il modulo TDC vogliamo considerare anche gli hit multipli, è stato deciso di traslare sull'asse x del grafico di $1000 * (NUMERO\_HIT - 1)$ per i tempi positivi e di $-1000 * (NUMERO\_HIT - 1)$ per i tempi negativi. Questo meccanismo permette di individuare a colpo d'occhio se un gruppo di valori corrispondono al primo hit, o al secondo e cosí via. Come già spiegato nella descrizione della funzione my_read_One_Event_TDC, si considerano al massimo cinque hit. In figura 4.8 è presente il frammento di diagramma a blocchi rappresentante la realizzazione dei due grafici rappresentanti i tempi acquisiti dal modulo TDC.

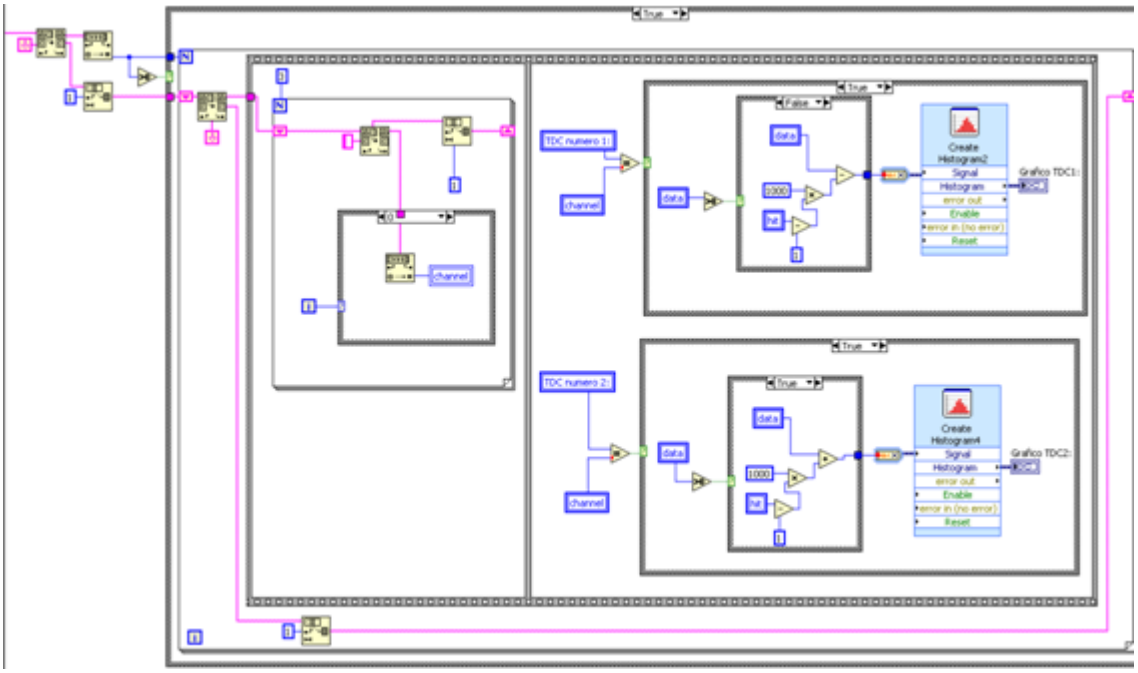


Figura 4.8: Frammento di diagramma a blocchi rappresentante la realizzazione dei due grafici rappresentanti i tempi acquisiti dal modulo TDC

4.6 Applicazione finale

Unendo le tre parti descritte in precedenza si ottiene l'applicazione finale, la cui interfaccia utente è rappresentata in figura 4.9.

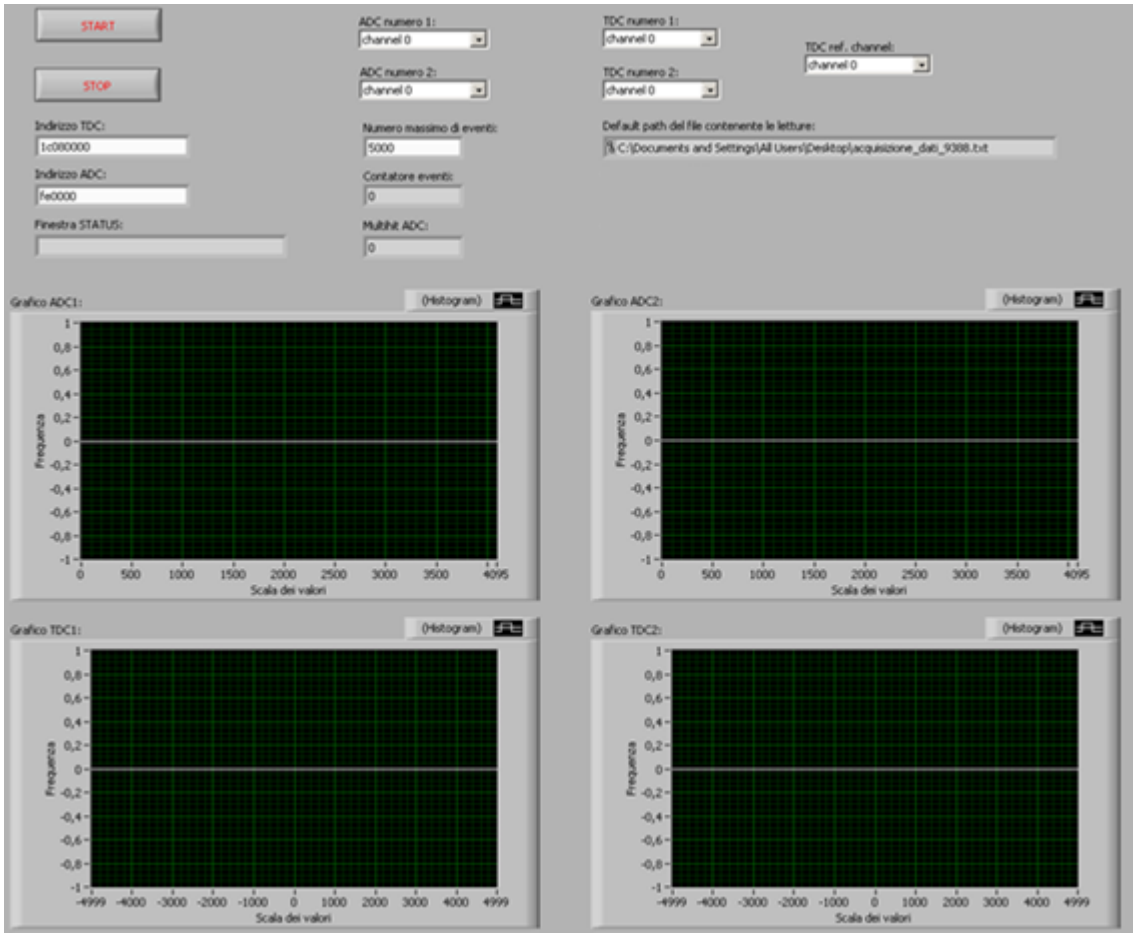


Figura 4.9: Pannello frontale dell'applicazione finale

4.6.1 Guida all'uso

Non appena viene eseguita l'applicazione è necessario inserire: due canali del modulo ADC, due canali del modulo TDC e il canale di riferimento del modulo TDC. C'è la possibilità di modificare l'indirizzo del modulo ADC e del modulo TDC, che vengono inizialmente settati dall'applicazione rispettivamente a 1c080000 e fe0000. Un'altro campo che viene settato dall'applicazione è il numero massimo di eventi da acquisire, che è impostato a 5000: a piacere puó essere mod-

ificato, anche in tempo reale durante l'acquisizione. L'ultimo campo impostato dall'applicazione è il path del file contenente tutti i dati acquisiti, che non puó essee modificato dall'utente.
Una volta che l'utente ha terminato di impostare i campi necessari puó dare inizio all'acquisizione premendo il pulsante START. L'applicazione a questo punto tenterà di connettersi e di configurare il setup dei moduli ADC e TDC, se non si riscontrano errori viene stampata la stringa "connessione stabilita" nel campo Finestra STATUS, in caso di errore invece verà stampato il tipo di errore. Una volta completata la connessione e il setup dei moduli ha inizio l'acquisizione e i grafici prenderanno forma in tempo reale: i due superiori rappresentano i valori letti dai due canali dell'ADC selezionati, mentre i due grafici inferiori rappresentano i tempi relativi dei due canali del TDC selezionati. Il termine dell'acquisizione puó avvenire per il ragiungimento del numero massimo di eventi acquisiti o per la pressione del pulsante STOP. In entrambi i casi compare una finestra nella quale è possibile indicare il path in cui si desidera salvare il file contenente tutti i dati acquisiti; nel caso si esca da tale finestra senza aver effettuato il salvataggio si puó comunque recuperare i dati dal file creato di defualt sul desktop (nel caso in cui si effettui invece il salvataggio richiesto, il file di default viene eliminato in quanto sarebbe un duplicato).

Capitolo 5

Test, acquisizione finale e conclusioni

In questo capitolo verranno descritti i test effettuati per valutare le prestazioni dell'applicazione realizzata, e i valori ottenuti da un ciclo di acquisizione dati (spiegando il significato dell'andamento dei grafici ottenuti).

5.1 Test delle prestazioni

Ai fini del nostro esperimento non si necessita di un applicazione che lavori a frequenze superiori a qualche decina di Herz, ma per completezza di analisi si è deciso di testare il programma di acquisizione dati con frequenze sempre più alte e vederne il comportamento. A questo scopo si è simulato il passaggio di raggi cosmici attraverso lo scintillatore con un diodo luminoso (led) di cui è possibile variare la frequenza di lavoro. In particolare è interessante individuare fino a che frequenza l'applicazione acquisisce tutti gli eventi generati, e succesivamente con quale andamento le prestazioni calano all'aumentare della frequenza. I test fanno riferimento ad acquisizione di 5000 eventi. Nel seguito vengono presentati due grafici contenenti rispettivamente l'andamento dei tempi di acquisizione e il calo di prestazioni all'aumentare della frequenza.

5.1.1 Curva dei tempi di acquisizione

Tutti i tempi sono stati calcolati sull'acquisizione di 5000 eventi: si è notato che fino ad una frequenza di circa 35 Hz l'applicazione non perde eventi ed il tempo che impiega ad effettuare l'acquisizione coincide col tempo impiegato dal pulser per generare i 5000 eventi. Aumentando la frequenza però il tempo di acquisizione reale aumenta rispetto al tempo di acquisizione teorico, segno che il programma non riesce ad acquisire tutti gli eventi generati. In figura 5.1 è presente il grafico che rappresenta l'andamento dei tempi calcolati.

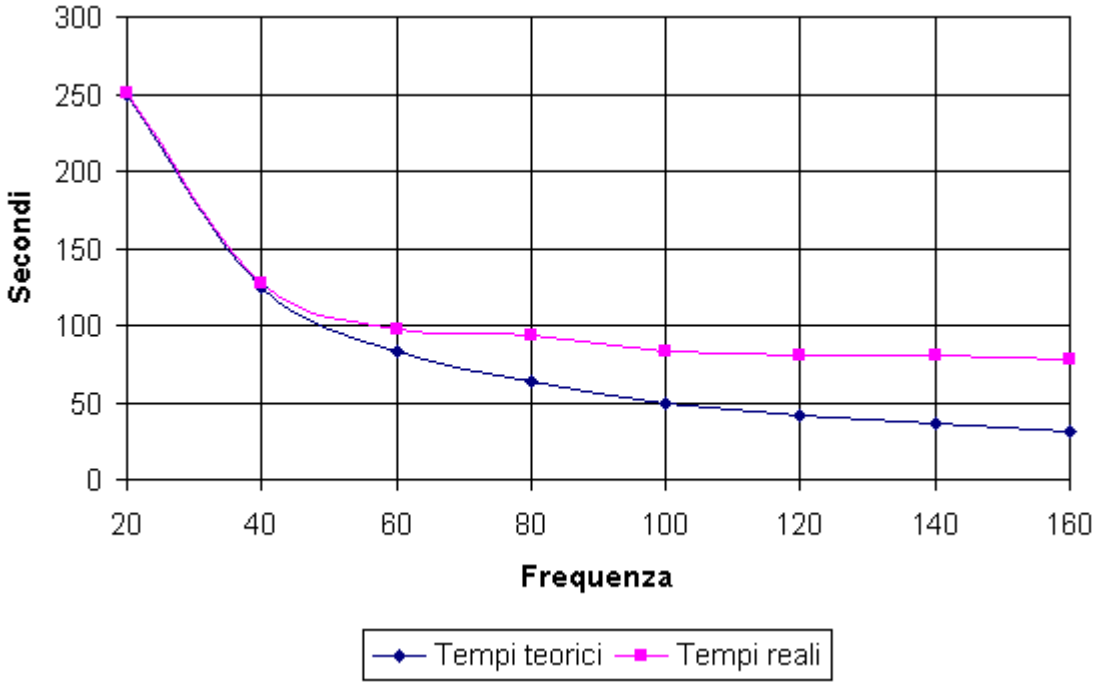


Figura 5.1: Grafico rappresentante i tempi di lettura teorici e reali al variare della frequenza

5.1.2 Curva della percentuale degli eventi acquisiti

Nel grafico in figura 5.2 viene visualizzata la percentuale degli eventi acquisiti rispetto al totale degli eventi generati. Come anticipato nella sezione 5.1.1 fino ad una frequenza di circa 35 Hz non si perdono eventi, successivamente all'aumentare della frequenza si acquisisce una percentuale di eventi sempre minore fino ad arrivare ad una frequenza di 160 Hz a cui corrisponde una percentuale di eventi acquisiti del 40%.

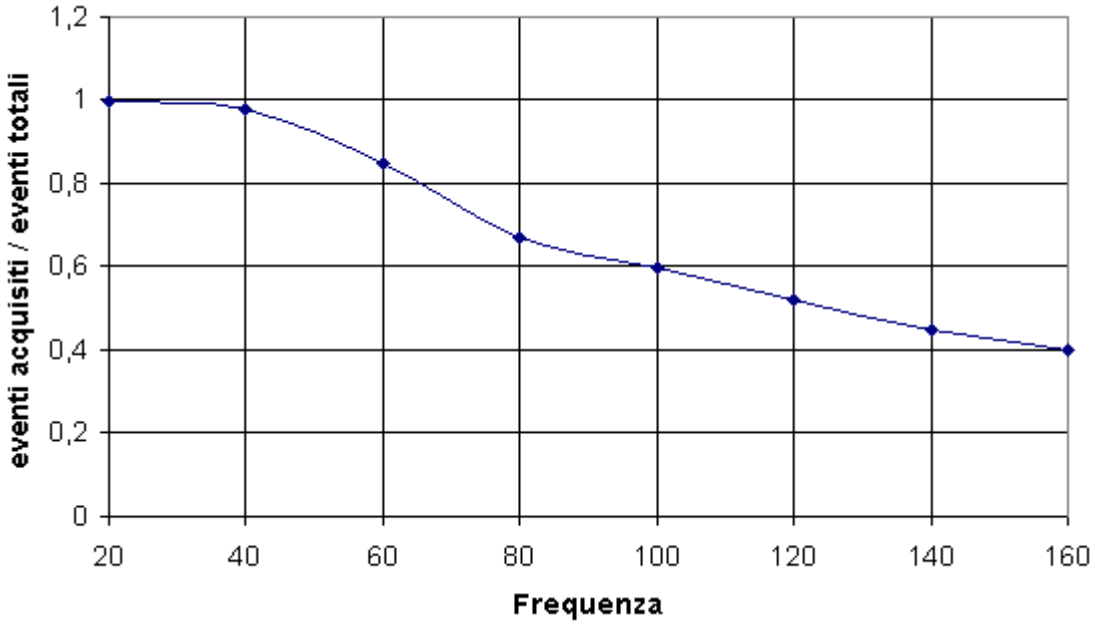


Figura 5.2: Grafico rappresentante l'efficienza di lettura al variare della frequenza

Nel seguito verranno mostrati la determinazione del piedistallo e i risultati ottenuti effettuando un ciclo di acquisizioni.

5.2 Determinazione del piedistallo

Come già descritto in precedenza nel paragrafo riguardante i SiPM (1.2), vi è una certa probabilità che il foto moltiplicatore rilevi degli eventi in realtà inesistenti: questa probabilità è chiamata "Dark Counting Rate". Ciò comporta un errore di cui bisogna tener conto nell'effettuare le misurazioni, per tale motivo si effettua un ciclo di acquisizione dati senza collegare la fibra ottica ai SiPM, in modo tale che gli unici eventi rilevati siano questi falsi eventi dovuti all'elettronica. Il picco che si ottiene nel grafico è chiamato piedistallo. Per essere sicuri che i dati ottenuti dalle misurazioni non siano affetti dall'errore dovuto a questi falsi eventi, si prendono in considerazione solo i valori superiori all'intervallo del piedistallo. Nel nostro caso un buon valore di taglio può essere 140.

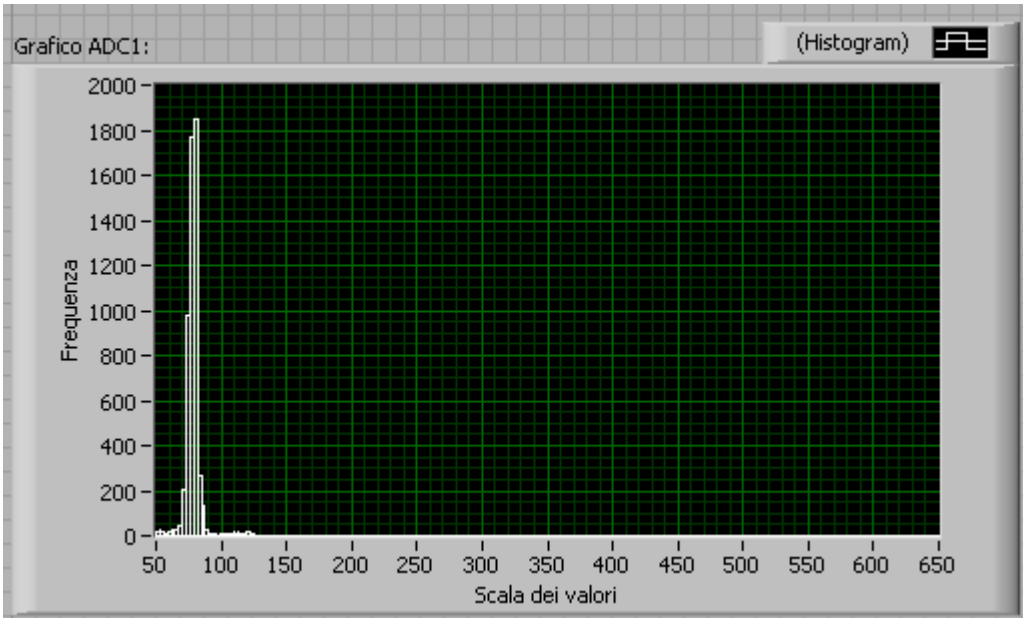


Figura 5.3: Grafico che rappresenta l'intervallo di valori del piedistallo

5.3 Analisi dei risultati ottenuti

L'applicazione è stata impostata per acquisire i dati da un solo SiPM, in modo tale da poter visualizzare le stesse misurazioni su i due grafici ADC e vederli contemporaneamente su due scale diverse. In particolare il grafico ADC1 permette di osservare la curva gaussiana nel dettaglio, il grafico ADC2 invece permette di osservare come non siano stati acquisiti valori superiori al limite della coda destra della curva. Stesso discorso per quanto riguarda i grafici dei tempi: il grafico TDC1 visualizza la curva nel dettaglio, mentre TDC2 permette di vedere come non ci siano praticamente stati casi di multihit.

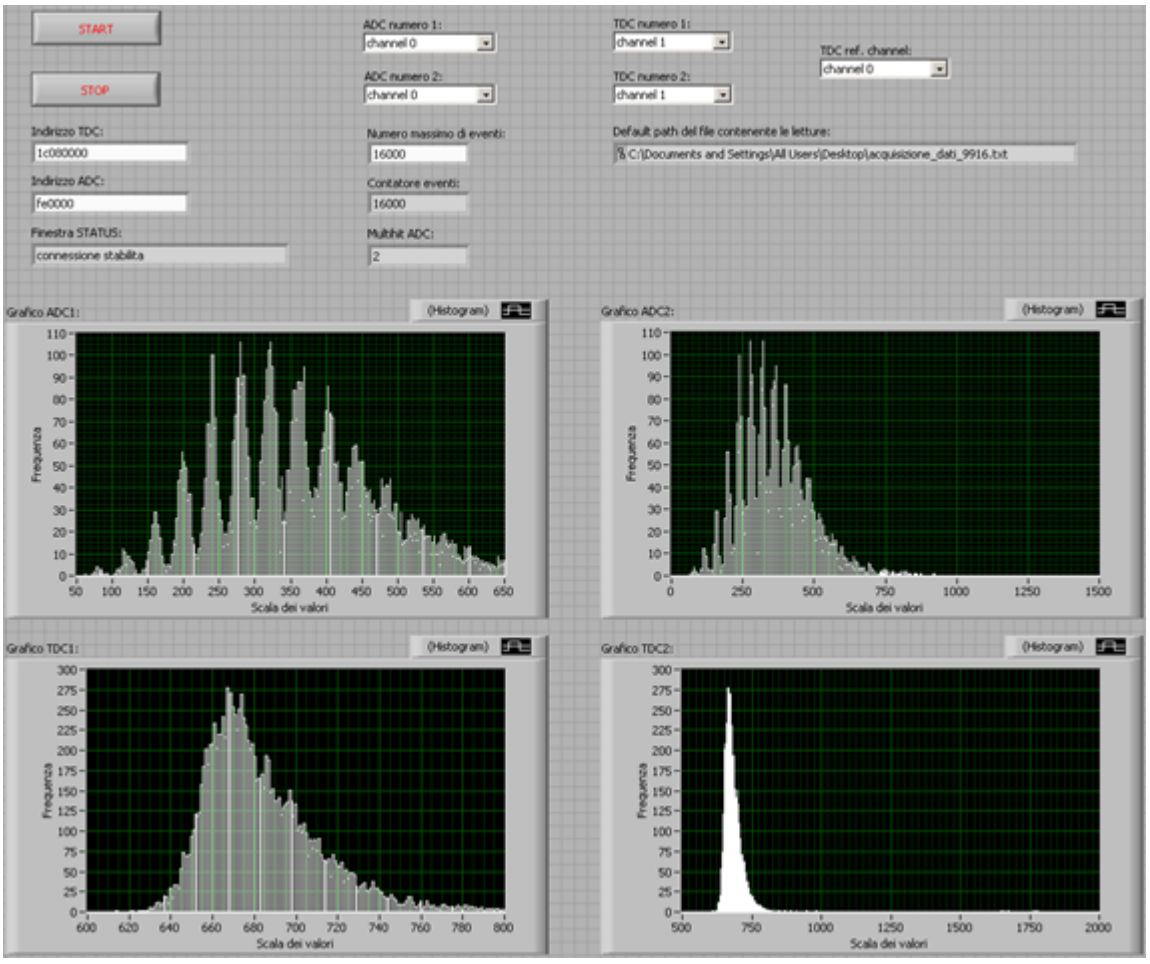


Figura 5.4: Pannello frontale dell'applicazione una volta terminata l'acquisizione dati

5.3.1 Analisi grafico ADC

Osservando il grafico si notano dei picchi, questi picchi corrispondono al numero di fotoelettroni rilasciati dallo scintillatore quando colpito da particelle cariche. Il primo picco, che si conclude intorno a 100, rappresenta il piedistallo; il secondo picco individua le misurazioni in cui è stato rilevato un solo fotoelettrone, e così via. Il valore di taglio, che è stato determinato come 140, ci fa scartare i primi due picchi. Si nota inoltre che, eseguendo un fit gaussiano sul grafico, il valore medio della distribuzione risulta essere intorno a 370; quindi quando una particella carica attraversa lo scintillatore, viene rilasciata in media una quantità di carica pari a sette fotoelettroni, essendo 370 il valore medio del picco corrispondente a sette fotoelettroni. Dato che il fondo scala di 4096 canali del modulo ADC corrisponde a una carica di 204,8 pCoulomb, effettuando una proporzione si calcola che lo scintillatore emette una carica media di 17,5 pCoulomb.
Per quanto riguarda l'efficienza dello scintillatore si ottiene calcolando il rapporto tra l'area totale del grafico e l'area del grafico che esclude il piedistallo, questo valore risulta essere del 98%.

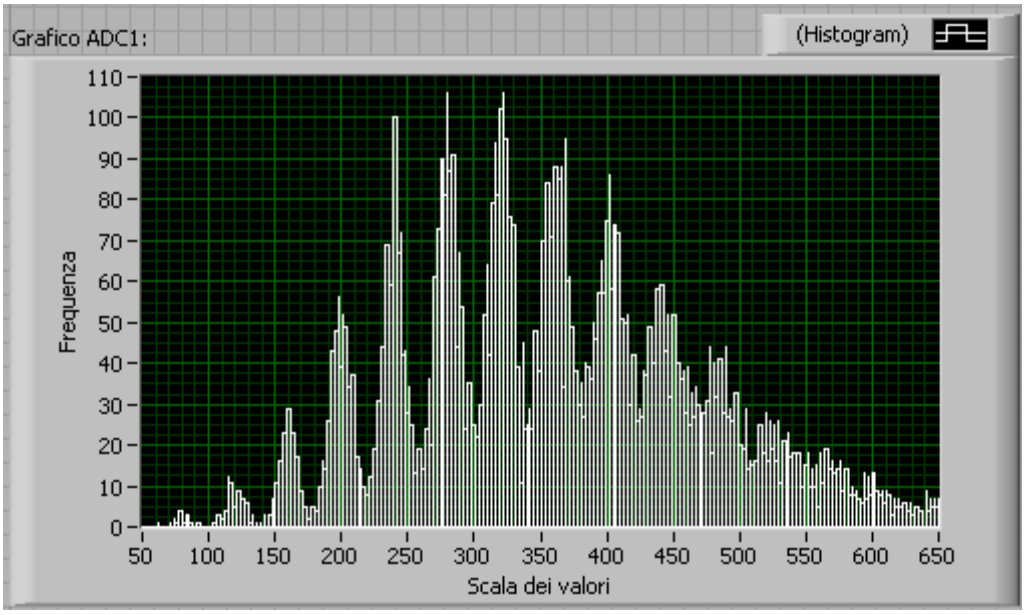


Figura 5.5: Grafico dei valori letti dal modulo ADC

5.3.2 Analisi grafico TDC

La sensibilità data dalle specifiche tecniche è di 0,1 ns, ed indica un'unità sull'asse delle ascisse nel grafico dei tempi. Per determinare la risoluzione temporale si calcola σ della curva gaussiana presente nel grafico TDC1 o TDC2 in figura 5.3, che corrisponde a 10, la si moltiplica per la sensibilità e si ottiene una risoluzione temporale di 1 ns. Ciò significa che in ogni misurazione di tempi si ha un errore di $\pm$ 1 ns.

5.4 Conclusioni

I valori letti rispecchiano le aspettative iniziali, l'applicazione ha risposto in maniera positiva alle esigenze che si sono riscontrate nel procedere delle misurazioni e non sono stati riscontrati malfunzionamenti o errori di progettazione.

Elenco delle figure

Bibliografia

[1] *Datasheet of Kuraray WLS fibers, Mod. T11*.

[2] *Techniques for nuclear and particle physics*, William R. Leo. Springer-Verlag, 1993.

[3] *Mod. V1190-VX1190 A/B, 128/64 Ch Multihit TDC*. Settima revisione, 09/05/2006. Manuale per informazioni tecniche riguardante il modulo TDC utilizzato nell'esperimento.

[4] *LeCroy Model 1182 VME Charge-to-Digital Converter*. Indirizzo web: http://www.lecroy.com/lrs/dsheets/1182.htm. Manuale per informazioni tecniche riguardante il modulo ADC utilizzato nell'esperimento.

[5] *Model 618-3, 618-9U & 620-3 Adapters Hardware Manual*. Seconda revisione, 07/03/2005. Manuale per informazioni tecniche riguardante l'adattatore PCI to VME utilizzato nell'esperimento.

[6] Manuale software fornito da LabVIEW. Aggiornato ad agosto 2007.