



# **UNIVERSITÀ DEGLI STUDI DI FERRARA**

Dipartimento di Matematica e Informatica  
Corso di Laurea in Informatica

## **SVILUPPO IN C++ DI UN SISTEMA DI CONTROLLO E DI ACQUISIZIONE DATI PER DIGITIZER CAEN**

Relatore:  
Dott. Michele Gambetti

Laureando:  
Daniele Berto

Correlatore:  
Dott. Mirco Andreotti

Anno Accademico 2014 - 2015







# Introduzione

Il lavoro presentato in questa tesi si inquadra nelle attività dell’Istituto Nazionale di Fisica Nucleare nell’ambito del consorzio Eurogammas composto da INFN, Università di Roma ’La Sapienza’, CNRS (Francia) e vari partner industriali in ambito europeo. Il consorzio ha lo scopo di fornire la progettazione, realizzazione e messa in opera di una sorgente di fascio gamma dell’infrastruttura di ricerca ELI-NP (Extreme Ligth Infrastructure - Nuclear Physics) presso ’Horia Hulubei National Institute for Physics and Nuclear Engineering’ a Magurele, Bucharest, Romania. Questa infrastruttura sarà utilizzata per la ricerca nei campi della fisica nucleare, astrofisica, fisica fondamentale, e applicazioni nei campi di materiali nucleari, lavorazioni di scorie nucleari, scienza dei materiali e scienze della vita. In particolare questo elaborato è frutto di un lavoro svolto in collaborazione con il Dipartimento di Fisica e Scienze della Terra dell’Università degli Studi di Ferrara e le sezioni INFN di Ferrara, Firenze e Catania, le quali sono responsabili delle attività del Work Package 09 (WP09) di Eurogammas. Il WP09, coordinato dalla sezione INFN-Ferrara, prevede la realizzazione del sistema di collimazione e caratterizzazione dei fasci di raggi gamma che saranno prodotti presso l’infrastruttura ELI-NP. Il sistema di caratterizzazione è un insieme di rivelatori progettato per misurare le caratteristiche dei fasci di raggi gamma. Le caratteristiche temporali dei fasci (32 impulsi di fotoni distanti 16 ns ripetuti con una frequenza di 100 Hz) e le informazioni che se ne vogliono ricavare richiedono che l’elettronica di *front end* dei rivelatori abbia delle prestazioni temporali di campionamento molto elevate con risoluzioni dell’ordine del centinaio di picosecondi. La scelta della collaborazione WP09 è ricaduta sui prodotti commerciali dell’azienda CAEN, la quale mette a disposizione digitalizzatori (Digitizer) ad alte prestazioni ampiamente collaudati.

I requisiti richiesti dal sistema di acquisizione dati per il sistema di caratterizzazione hanno reso necessario lo sviluppo del lavoro presentato in questa tesi.

La progettazione e realizzazione del sistema di acquisizione dati da Digitizer CAEN, presentato in questa tesi, ha seguito l'obiettivo di massimizzare e ottimizzare tutte le prestazioni del sistema al fine di ottenere un prodotto in grado di soddisfare non solo le esigenze del WP09 di Eurogammas, ma anche in grado di essere utilizzato in altre applicazioni, per le quali siano richieste le massime prestazioni. Il sistema è stato inoltre realizzato con struttura modulare, in modo da poter essere facilmente modificato e riadattato alle esigenze di eventuali altre applicazioni. Tutto il sistema di acquisizione è realizzato in C/C++ ed è stato sviluppato in ambiente Linux al fine di essere il più distribuibile possibile. Il sistema realizzato è stato inoltre testato durante un test beam presso il CERN svolto dal gruppo INFN di Catania del WP09 nell'ambito di Eurogammas, fornendo ottimi risultati di funzionamento.

Nel primo capitolo della tesi si introdurrà il problema e il contesto operativo, nel secondo capitolo si analizzeranno alcune problematiche relative ad un utilizzo efficiente dei digitizer CAEN utilizzando le librerie C messe a disposizione dall'azienda, nel terzo capitolo si tratterà dello sviluppo del nuovo sistema di controllo e di acquisizione dati ObjectDump e nel quarto capitolo verrà presentata la prima applicazione sperimentale del sistema presso il Super Proton Syncrotron (SPS) del CERN di Ginevra effettuato dal gruppo di Catania del WP09 di Eurogammas.

# Indice

<b>Introduzione</b>	<b>i</b>
<b>1 Contesto operativo e introduzione al problema</b>	<b>1</b>
1.1 Il digitizer . . . . .	2
1.2 Software a disposizione . . . . .	3
1.3 Documentazione a disposizione . . . . .	5
1.4 Hardware a disposizione . . . . .	5
1.5 Ambiente di sviluppo . . . . .	5
1.6 Principali limitazioni del panorama attuale . . . . .	6
<b>2 Considerazioni preliminari allo sviluppo di ObjectDump</b>	<b>7</b>
2.1 Controllare il digitizer . . . . .	7
2.2 Un software di readout di base . . . . .	8
2.2.1 Inizializzazione e configurazione del digitizer . . . . .	9
2.2.2 Inizio dell'acquisizione . . . . .	11
2.2.3 Trattamento dei dati acquisiti . . . . .	11
2.2.4 Termine dell'acquisizione . . . . .	12
2.3 La decodifica dei dati acquisiti . . . . .	14
2.4 La velocità di acquisizione . . . . .	15
2.4.1 Versione 1 . . . . .	18
2.4.2 Versione 2 . . . . .	20
2.4.3 Versione 3 . . . . .	21
2.5 Decodificare offline i dati . . . . .	22
2.6 Problema relativo alla compilazione dei driver . . . . .	26
<b>3 Sviluppo di ObjectDump</b>	<b>27</b>
3.1 Sviluppo di un prototipo . . . . .	28

3.1.1	Aspetti emersi dallo sviluppo del prototipo . . . . .	28
3.2	Caratteristiche esterne del nuovo sistema . . . . .	29
3.2.1	Interfaccia utente . . . . .	29
3.2.2	Aiuti all'utente . . . . .	29
3.2.3	File di configurazione . . . . .	29
3.2.4	Affidabilità . . . . .	30
3.2.5	Controllo completo sul digitizer . . . . .	30
3.2.6	Supporto TCP/IP . . . . .	30
3.2.7	Documentazione con Doxygen . . . . .	30
3.3	Caratteristiche interne del nuovo sistema . . . . .	30
3.3.1	Acquisizione multithreaded . . . . .	31
3.3.2	Manutenibilità del codice e defensive programming . .	31
3.3.3	Paradigma a oggetti . . . . .	31
3.4	Sviluppo del lato server del sistema . . . . .	31
3.4.1	Creazione dello scanner per leggere il file di configura- zione . . . . .	33
3.4.2	Salvataggio delle informazioni ottenute dal file di con- figurazione . . . . .	33
3.4.3	Wrapper attorno alle funzioni della libreria CAENDi- gitizer . . . . .	34
3.4.4	Gestione dei codici di errore . . . . .	34
3.4.5	La macchina a stati . . . . .	35
3.4.6	Organizzazione di un modello producer-consumer . .	35
3.4.7	Strumenti per gestire l'input via tastiera e via TCP/IP	37
3.4.8	L'oggetto FlowControl . . . . .	38
3.4.9	La funzione getopt . . . . .	38
3.4.10	L'oggetto OutputModule . . . . .	40

3.4.11 Il file di log . . . . .	41
3.5 Il lato client ed esempio di implementazione di un comando . .	41
3.5.1 Funzione main del lato client . . . . .	43
3.5.2 Funzione ricevitore . . . . .	46
3.5.3 Funzione command_parser . . . . .	48
3.5.4 Funzione reg_matches . . . . .	49
3.5.5 Aggiungere un nuovo comando al sistema . . . . .	49
3.6 Il programma di decodifica offline . . . . .	53
3.7 Organizzazione del codice sorgente del sistema . . . . .	54
3.8 Installazione del sistema . . . . .	55
3.9 Avvio dei programmi . . . . .	55
3.9.1 Avvio del lato server . . . . .	56
3.9.2 Avvio del lato client . . . . .	56
3.9.3 Avvio del programma di decoding . . . . .	56
3.10 Comandi eseguibili dal sistema . . . . .	57
3.11 Contenuto del file di configurazione del sistema . . . . .	59
3.11.1 Impostazioni comuni a tutti i canali . . . . .	59
3.11.2 Impostazioni per singolo canale o gruppo . . . . .	63
3.12 Compilazione di ObjectDump . . . . .	64
<b>4 Test del sistema</b>	<b>65</b>
<b>Conclusioni</b>	<b>69</b>
<b>A Risultati dei test di data transfer rate</b>	<b>71</b>
<b>B Datasheet</b>	<b>79</b>
<b>Bibliografia</b>	<b>83</b>



# Capitolo 1

## Contesto operativo e introduzione al problema

L’azienda CAEN - Costruzioni Apparecchiature Elettroniche Nucleari S.p.A. - è attiva nel mercato dei prodotti hardware e software per la ricerca attinente la fisica nucleare e la fisica delle particelle. Tra i prodotti da essa offerti ci sono i digitizer, un nome con cui si racchiude un insieme di strumenti più o meno complessi con cui è possibile compiere attività di digitalizzazione e acquisizione di un segnale analogico.

L’azienda offre un insieme di librerie scritte in codice C e Labview con cui è possibile interfacciarsi con i digitizer e mette a disposizione un software dimostrativo *open source*, Wavedump, scritto in linguaggio C che utilizza tali librerie.

I digitizer CAEN, le librerie di interfacciamento e Wavedump sono strumenti di uso comune nel campo dell’acquisizione dati prodromiche alle ricerche di fisica nucleare e non solo.

In questo documento si tratterà dello sviluppo di ObjectDump, un nuovo sistema di controllo e di acquisizione dati per digitizer CAEN.

In questo primo capitolo verranno introdotti gli strumenti con cui è stato realizzato il sistema: i digitizer CAEN e i prodotti software offerti dall’azienda. In particolare, verrà introdotto il software *open source* Wavedump offerto dall’azienda CAEN e si tratterà delle sue limitazioni rispetto alle esigenze WP09 di Eurogammas[1][2].

## 1.1 Il digitizer

Il digitizer è uno strumento complesso che permette, principalmente, di digitalizzare un segnale analogico. L'azienda CAEN offre una vasta gamma di modelli di digitizer divisi in famiglie[4].

Per lo svolgimento delle operazioni di test preliminari alla redazione di questo elaborato, si sono utilizzati due digitizer: il modello dt5751 della famiglia 751 e il modello v1742 della famiglia 742.

La digitalizzazione delle forme d'onda avviene con frequenze di campionamento molto elevate, che rendono il digitizer uno strumento potente ed adatto alla ricerca. In particolare, il modello v1742 può digitalizzare un segnale con una frequenza di campionamento fino a 5 GS/s.

Un'altra delle caratteristiche fondamentali di un digitizer è la risoluzione della conversione. Essa dipende dal modello di digitizer: quelli della famiglia 751 possono digitalizzare un segnale con una risoluzione di 10 bit, quelli della famiglia 742 con una risoluzione di 12 bit.

Il numero di canali in input varia anch'esso: il modello dt5751 ne ha 4, il modello v1742 ne ha 32.

Il range acquisibile da entrambi i modelli è di 1 Volt a cui è possibile applicare un offset di 0.5 Volt al modello dt5751 e di 1 Volt al modello v1742.

Il digitizer converte un segnale analogico entrante in uno digitale e, quando riceve un segnale di trigger, salva il risultato della conversione nel buffer di readout.

Il digitizer può ricevere segnali di trigger hardware, di trigger software e in ogni canale può essere impostata una soglia superata la quale scatta il trigger.

È possibile impostare le azioni che il digitizer può intraprendere una volta

ricevuto un segnale di trigger: esso può essere usato per l’acquisizione, può essere propagato sull’uscita trigger out o può essere ignorato. Per ulteriori dettagli tecnici, fare riferimento ai datasheet B.1 e B.2 tratti dai manuali scritti dall’azienda CAEN[6][7].

Come accennato nell’introduzione e come si vedrà in seguito, l’azienda CAEN offre i driver per comunicare con i digitizer e una gerarchia di librerie C e Labview per controllarli e acquisire dati. Viene altresì fornito il software C *open source* Wavedump.

## 1.2 Software a disposizione

Il software a disposizione consiste in:

- librerie C CAENVme
- librerie C CAENComm
- librerie C CAENDigitizer
- software Wavedump scritto in C
- Driver per VME
- Driver per PCI card modello A3818
- CentOS 7 a 64 bit e Linux kernel-3.10.0-327.10.1.el7.x86\_64

Tutto il software a disposizione può essere scaricato gratuitamente dal sito dell’azienda CAEN[8]. Lo schema delle librerie fornite dall’azienda CAEN è indicato nella figura 1.1: esse posano sui drivers e sono organizzate in layers.

### CAENVme

Libreria per il funzionamento del bridge VME. Le librerie di livello superiore richiedono l’installazione di questa libreria, anche se non c’è la necessità di usare un bridge VME[13].

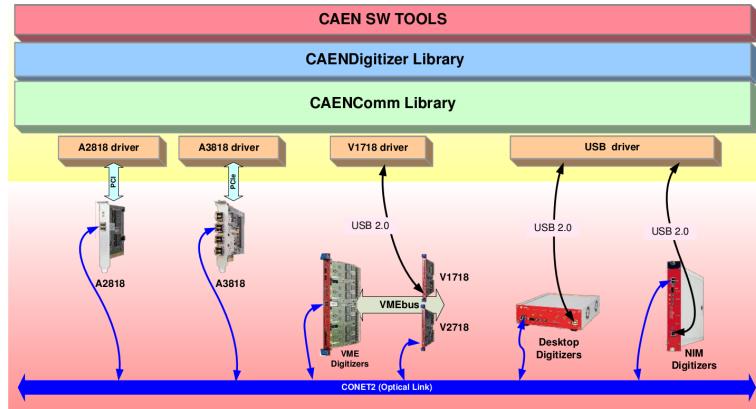


Figura 1.1: Livelli hardware e software dei prodotti CAEN[13]

## CAENComm

Libreria che astrae i dettagli della comunicazione col digitizer fornendo un’interfaccia comune alle implementazioni di livello superiore. I servizi offerti consentono di aprire e di chiudere la comunicazione con un digitizer e di scrivere e leggere dai registri[13].

## CAENDigitizer

Libreria che fornisce una vasta gamma di funzioni ad alto livello per controllare il digitizer, per acquisire dati e per decodificarli. Il software ObjectDump e il software Wavedump sono basati su questa libreria[13].

## Wavedump

Wavedump[14] è il software di controllo e di acquisizione dati *open source* fornito dall’azienda CAEN scritto in C. Utilizza le funzioni della libreria CAENDigitizer. Wavedump funziona su sistemi operativi Linux e Windows, è controllabile mediante linea di comando ed è configurabile mediante un file di configurazione. Consente di visualizzare i dati acquisiti con Gnuplot e di salvarli su disco in formato binario o in formato ASCII. Non dispone di

un’interfaccia TCP/IP o di aspetti di multithreading.

### **1.3 Documentazione a disposizione**

La documentazione relativa alle specifiche tecniche dei digitizer[6][7], quella relativa alle librerie software[13] e quella relativa al software Wavedump[14] può essere reperita gratuitamente online sul sito dell’azienda CAEN.

### **1.4 Hardware a disposizione**

L’hardware utilizzato per la scrittura del nuovo sistema di acquisizione e per i test preliminari è di proprietà dell’INFN-Sezione di Ferrara. Esso consiste in:

- Digitizer CAEN modello dt5751 (vedi il datasheet B.1)
- Digitizer CAEN modello v1742 (vedi il datasheet B.2)
- Pulsatore
- Crate VME CAEN
- Computer Dell con PCI card CAEN modello A3818 e hard disk Seagate Barracuda 7200.12 Serial ATA modello ST3320418AS (vedi il datasheet B.3). Specificare il modello di hard disk è importante per i test effettuati sui tempi di acquisizione.

### **1.5 Ambiente di sviluppo**

Si è scelto di utilizzare come ambiente di sviluppo e di sperimentazione centOS 7.0 a 64 bit e kernel Linux kernel-3.10.0-327.10.1.el7.x86\_64. Per il trattamento dei dati si è utilizzato GNU Octave, versione 3.6.4. Per costruire il diagramma di classi si è utilizzato yWorks yEd versione 3.14.3. I linguaggi di programmazione usati sono il C e il C++.

## 1.6 Principali limitazioni del panorama attuale

Il software *open source* Wavedump consente di interfacciarsi con un digitizer CAEN, settarlo con il contenuto del file di configurazione, acquisire dei dati, salvarli e visualizzarli con Gnuplot. Il software è una applicazione a linea di comando: è possibile controllarla mediante la digitazione di singoli tasti seguiti da invio.

Wavedump presenta delle limitazioni rispetto alle esigenze del WP09 di Eurogammas[1][2] che hanno reso necessaria la scrittura di un nuovo sistema di controllo e di acquisizione. La limitazione fondamentale è la velocità di acquisizione: il software non riesce ad acquisire dati a più di 4 MB/s (vedi sezione 2.4.1). Questo è un limite: è promessa una velocità di acquisizione fino a 80 MB/s con fibra ottica (vedi datasheet B.2), 4 MB/s sono limitanti. Velocità di acquisizione elevate sono indispensabili per seguire i treni di trigger hardware che possono arrivare a intervalli molto vicini.

Inoltre, il codice di Wavedump è difficilmente manutenibile. Considerando che il software deve essere utilizzato da ricercatori in contesti dove è normale adattare il codice alle proprie esigenze, la scarsa manutenibilità è un limite. Per sopperire a queste importanti limitazioni si è sviluppato il nuovo sistema di controllo e di acquisizione ObjectDump (vedi capitoli 3 e 4). L'attività di sviluppo è stata preceduta da una attività di test sui tempi di acquisizione come illustrato nel capitolo 2.

# Capitolo 2

## Considerazioni preliminari allo sviluppo di ObjectDump

In questo capitolo si intordurranno gli studi prodromici alla realizzazione del nuovo sistema di controllo e di acquisizione dati ObjectDump. Nello specifico, si spiegherà come è possibile controllare i digitizer, come può essere scritto un software di acquisizione di base e come il codice del software *open source* Wavedump possa essere ottimizzato per incrementare la velocità di trasferimento e di acquisizione dati.

### 2.1 Controllare il digitizer

Il digitizer è controllabile mediante la scrittura sui suoi registri di controllo. La descrizione dei registri è presente nel manuale del digitizer[6][7]. Le librerie CAENDigitizer fornite dall'azienda CAEN consentono di configurare il digitizer con un approccio ad alto livello astraendo i particolari della scrittura sui registri. È, comunque, disponibile una funzione per la scrittura su registro e una per la lettura. Si sottolinea che le funzioni di libreria non coprono tutte le possibili configurazioni consistenti dei registri di controllo del digitizer, quindi la possibilità di scrivere e leggere i registri è molto utile: l'azienda mette a disposizione delle librerie che consentono di configurare il digitizer in un certo modo e di effettuare il readout ma se queste funzioni di libreria non dovessero bastare, è possibile configurare i registri di controllo del digitizer usando l'apposita funzione di libreria. Il digitizer può essere, di conseguenza, controllato completamente.

Per controllare se le funzioni di libreria hanno effettivamente impostato il digitizer in un certo modo, si può procedere alla lettura dei registri che si presume siano stati scritti per il compimento dell'operazione: è il loro contenuto che determina le azioni svolte dal digitizer.

Si può comunicare direttamente col digitizer mediante USB 2.0 o mediante optical link.

Se si tratta di un digitizer VME, esso può essere inserito in un crate VME e, in questo caso, si può comunicare mediante USB 2.0 con il crate, e le informazioni vengono trasmette al digitizer tramite il VMEBus.

Nel caso venga utilizzato l'optical link, occorre necessariamente disporre di un'apposita PCI card fornita dall'azienda CAEN. Esistono due modelli di questa PCI card: il modello A2818[9] e il modello A3818[10]. Per il loro utilizzo devono essere installati gli appositi driver scaricabili dal sito dell'azienda CAEN[11][12].

Utilizzando i due digitizer, si è notato che i driver devono essere reinstallati ogni volta che il kernel del sistema operativo viene aggiornato.

## 2.2 Un software di readout di base

Supponiamo di avere appena acquistato un digitizer desktop dt5751 e di voler scrivere in linguaggio C un semplice software di readout. Si è deciso di trattare questo problema per fornire una linea guida su come installare i componenti hardware e software necessari al funzionamento del digitizer e su come utilizzare le librerie CAENDigitizer per effettuare una acquisizione.

Prima di scrivere il codice, infatti, occorre installare i driver necessari all'interfacciamento col digitizer. Per farlo, occorre scaricarli dal sito dell'azienda CAEN[9][10]. Successivamente si può procedere con la loro installazione spostandosi da terminale nella cartella scaricata con privilegi di root e digitando

i comandi configure, make e make install. I prodotti software per Linux dall’azienda CAEN seguono tutti questa procedura di installazione.

Una volta installati i driver, occorre installare le librerie necessarie alla scrittura di un software readout di base: si deve procedere al *download* e all’installazione della gerarchia di librerie necessaria all’utilizzo delle librerie CAENDigitizer illustrata nella sezione 1.2 [8].

A questo punto, è possibile iniziare a scrivere in linguaggio C il primo software di readout. Si inizia includendo la libreria CAENDigitizer mediante la direttiva di preprocessamento:

```
#include <CAENDigitizer>
```

Bisogna tener presente che, includendo questa libreria, in fase di linking occorre usare il flag -lCAENDigitizer.

Un programma di acquisizione di base si compone delle seguenti parti:

1. Inizializzazione e configurazione del digitizer
2. Inizio dell’acquisizione
3. Trattamento dei dati acquisiti
4. Termine dell’acquisizione

### **2.2.1 Inizializzazione e configurazione del digitizer**

Il digitizer viene inizializzato utilizzando la funzione

```
CAEN_DGTZ_OpenDigitizer (CAEN_DGTZ_ConnectionType LinkType, int  
LinkNum, int ConetNode, uint32_t VMEBaseAddress, int *handle)
```

impostandone i relativi argomenti: indicando l’handle da assegnare al digitizer, il tipo di link da utilizzare, l’eventuale *VMEBaseAddress* (ad esempio

0x11110000) e l'eventuale numero del digitizer nella daisy-chain<sup>1</sup>. Utilizzando questa funzione, viene resa disponibile la risorsa digitizer riferita dal suo handle.

Si sottolinea che ogni funzione della libreria CAENDigitizer restituisce un codice di errore. I possibili significati dei codici di errore sono molti e sono elencati nella documentazione della libreria[15]. Può essere utile implementare delle funzioni per stampare in modo comprensibile il significato dei codici di errore (il sistema ObjectDump implementa queste funzioni, vedi sezione 3.4.4).

Dopo aver aperto con successo il digitizer, è possibile configurarlo utilizzando le funzioni fornite dalla libreria CAENDigitizer: ognuna di queste riceve come primo argomento la risorsa handle ottenuta con la funzione

**CAEN\_DGTZ\_OpenDigitizer**

Se non è possibile ottenere la configurazione desiderata usando le funzioni di libreria, si può usare la funzione

**CAEN\_DGTZ\_WriteRegister**

e scrivere sul registro desiderato. Per leggere le informazioni contenute su un registro si può utilizzare la funzione

**CAEN\_DGTZ\_ReadRegister**

Si ricorda che prima di iniziare il setup del digitizer è necessario eseguire la funzione

**CAEN\_DGTZ\_Reset**

---

<sup>1</sup>Con il termine daisy-chain si definisce un'interconnessione di apparecchiature del computer, di unità periferiche, o di nodi di una rete, in serie tra loro: uno dopo un altro[3].

per riportare allo stato di *default* i suoi registri interni. Successivamente, è possibile eseguire le funzioni necessarie alla configurazione richiesta. Il setup può essere più o meno lungo a seconda della complessità della configurazione cercata. Nel manuale della libreria CAENDigitizer[13] sono reperibili informazioni dettagliate sulle funzioni di configurazione.

### 2.2.2 Inizio dell'acquisizione

Per ordinare al digitizer di iniziare l'acquisizione, si usa la funzione

`CAEN_DGTZ_StartAcquisitionSoftware`

poi, occorre leggere i dati resi disponibili dal digitizer con la funzione

`CAEN_DGTZ_ReadData`

La funzione ritorna sia che si siano letti dati, sia che non si siano letti. In questo caso, la grandezza dello stream di dati letto è uguale a zero. L'informazione sulla grandezza dello stream di dati è scritta dalla funzione *ReadData* nel suo secondo argomento *size*. Nell'esempio sotto riportato è stato usato un trigger software con la funzione

`CAEN_DGTZ_SendSWtrigger`

Il trigger software può risultare inadeguato in contesti sperimentali ma in fase di test è utile.

### 2.2.3 Trattamento dei dati acquisiti

La funzione

`CAEN_DGTZ_ReadData`

ottiene uno stream di dati e, nell'esempio, lo mette in *buffer*.

Per decodificare lo stream di dati ottenuto, cioè per mettere le informazioni

ivi contenute in una struttura apposita (nell'esempio nella struttura *Evt*, vedi la sezione 2.3), si utilizza la funzione

`CAEN_DGTZ_DecodeEvent`

#### 2.2.4 Termine dell'acquisizione

L'acquisizione termina con l'esecuzione della funzione

`CAEN_DGTZ_StopSoftwareAcquisition`

Successivamente, è possibile chiudere l'interfacciamento col digitizer con la funzione

`CAEN_DGTZ_CloseDigitizer`

Il codice riportato nel listato 2.1 è un esempio di un programma di acquisizione tratto dal manuale delle librerie CAENDigitizer[13] fornito dall'azienda CAEN. In esso sono contenute le funzioni descritte in questa sezione.

```
#include <stdio.h>
#include <CAENDigitizer.h>

int main(int argc, char* argv[])
{
    CAEN_DGTZ_ErrorCode ret;
    CAEN_DGTZ_BoardInfo_t BoardInfo;
    CAEN_DGTZ_EventInfo_t eventInfo;
    CAEN_DGTZ_UINT16_EVENT_t *Evt = NULL;
    char *buffer = NULL;
    int i,b,handle;
    int c = 0, count;
    char * evtptr = NULL;
    uint32_t size,bsize;
    uint32_t numEvents;

    //Apro il digitizer
    ret = CAEN_DGTZ_OpenDigitizer(CAEN_DGTZ_USB, 0, 0, 0x11110000,
        handle);
```

```
if(ret != CAEN_DGTZ_Success)
{
    printf("Can't open digitizer\n");
    return 1;
}

//Setup del digitizer
ret = CAEN_DGTZ_Reset(handle);
ret = CAEN_DGTZ_GetInfo(handle, &BoardInfo);
printf("\nConnected to CAEN Digitizer Model %s\n",
    BoardInfo.ModelName);
ret = CAEN_DGTZ_SetRecordLength(handle, 4096);
ret = CAEN_DGTZ_SetChannelEnableMask(handle, 1);
ret = CAEN_DGTZ_SetChannelTriggerThreshold(handle, 0, 32768);
ret = CAEN_DGTZ_SetChannelSelfTrigger(handle,
    CAEN_DGTZ_TRGMODE_ACQ_ONLY,1);
ret = CAEN_DGTZ_SetSWTriggerMode(handle,
    CAEN_DGTZ_TRGMODE_ACQ_ONLY);
ret = CAEN_DGTZ_SetMaxNumEventsBLT(handle, 3);
ret = CAEN_DGTZ_SetAcquisitionMode(handle,
    CAEN_DGTZ_SW_CONTROLLED);

if(ret != CAEN_DGTZ_Success)
{
    printf("Errors during Digitizer Configuration.\n");
    return 1;
}

//Alloco il buffer di readout
ret = CAEN_DGTZ_MallocReadoutBuffer(handle, &buffer, &size);

//Invio un trigger software
ret = CAEN_DGTZ_SendSWtrigger(handle);

//Acquisizione dell'evento triggerato
ret = CAEN_DGTZ_ReadData(handle,
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT, buffer, &bsize);

//Elaborazione dell'evento triggerato
ret = CAEN_DGTZ_GetNumEvents(handle, buffer, bsize, &numEvents);
printf(".");
count +=numEvents;
```

```

for (i=0;i<numEvents;i++)
{
    ret = CAEN_DGTZ_GetEventInfo(handle, buffer, bsize, i,
        &eventInfo, &evtptr);
    ret = CAEN_DGTZ_DecodeEvent(handle, evtptr, &Evt);
    //Libero la memoria
    ret = CAEN_DGTZ_FreeEvent(handle, &Evt);
}

printf("\nRetrieved %d Events\n", count[b]);

//Libero il buffer dell'evento
ret = CAEN_DGTZ_FreeReadoutBuffer(&buffer);

//Chiudo il digitizer
ret = CAEN_DGTZ_CloseDigitizer(handle);
return 0;
}

```

Listato 2.1: Esempio di un semplice programma C di readout

## 2.3 La decodifica dei dati acquisiti

Occorre precisare un concetto ricorrente in questo documento e nel linguaggio dei digitizer CAEN, cioè che cosa si intenda per decodifica dei dati. Quando viene utilizzata la funzione *CAEN\_DGTZ\_ReadData*, è ottenuto uno stream di dati diverso per ogni digitizer. Ad esempio, in figura 2.1 è riportato lo stream di dati ottenuto usando la funzione *CAEN\_DGTZ\_ReadData* con il modello v1742. La figura 2.2 mostra come è strutturato il campo channel data.

Per decodifica dei dati si intende effettuare l'unmarshalling dello stream di dati ottenuto con la funzione *CAEN\_DGTZ\_ReadData*, ovvero salvare le informazioni ottenibili da uno stream di dati in strutture appropriate.

La libreria CAENDigitizer offre la funzione *CAEN\_DGTZ\_DecodeEvent* per effettuare tale operazione: essa prende come argomenti il puntatore allo

stream di dati di cui effettuare l'unmarshalling e il puntatore alla struttura dati che conterrà i dati ben disposti.

Questo aspetto delle librerie è delicato: la funzione *CAEN\_DGTZ\_DecodeEvent* deve poter essere usata per la decodifica di uno stream di dati ottenuto da un digitizer qualsiasi.

Essa può prendere in input uno stream proveniente da ogni digitizer supportato dalla libreria, lo decodifica e pone il risultato nella struttura dati apposita: la libreria CAENDigitizer fornisce diversi tipi di queste strutture, che cambiano a seconda del tipo di digitizer.

Ci si aspetta che la funzione *CAEN\_DGTZ\_DecodeEvent* determini il tipo di procedura di unmarshalling da seguire riconoscendo il tipo dello stream di dati ricevuto in input, cioè capendo dallo stream stesso da che tipo di digitizer proviene. Invece, essa determina il tipo di procedura unmarshalling da utilizzare interrogando il digitizer stesso.

Questo è un grosso limite per l'utilizzo della funzione *CAEN\_DGTZ\_DecodeEvent*: occorre che il digitizer sia interfacciato con il calcolatore rendendo impossibile una decodifica completamente offline. Fortunatamente, c'è un'eccezione: per i modelli della famiglia 742 è offerta una funzione di decodifica (*X742\_DecodeEvent*) che funziona senza la necessità di interrogare il digitizer, rendendo possibile effettuare una decodifica completamente offline.

## 2.4 La velocità di acquisizione

Uno dei requisiti per sviluppare un buon sistema di controllo e di acquisizione è quello di scrivere un codice che consenta di acquisire dati alla massima velocità possibile.

L'azienda promette una velocità di trasferimento dati dal digitizer fino a 80 MB/s tramite optical link (vedi datasheet B.2). Il sustained data transfer

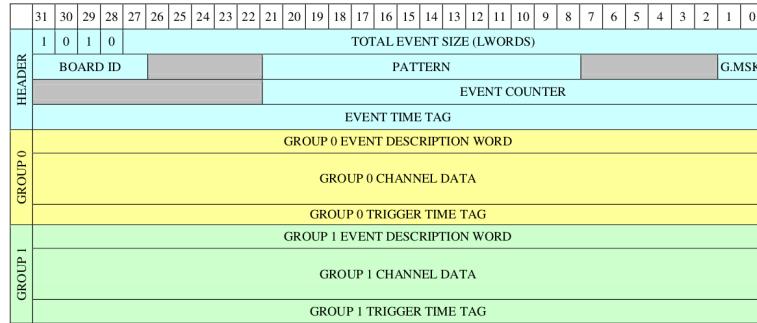


Figura 2.1: Struttura di uno stream di dati acquisiti da un digitizer di tipo v1742[6]

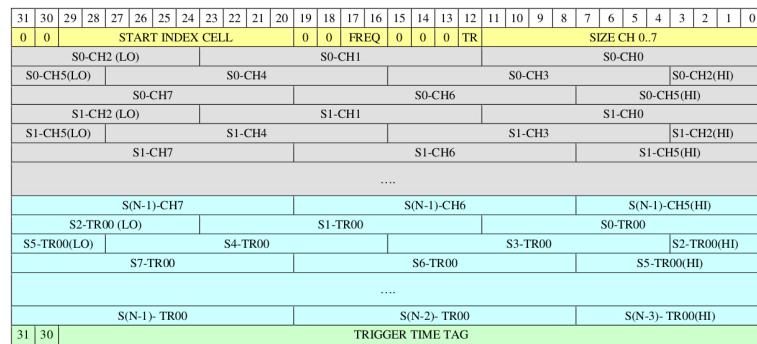


Figura 2.2: Contenuto del campo channel data[6]

rate degli hard disk della serie Seagate Barracuda® 7200.12 Serial ATA (il cui modello ST3320418AS è stato usato per i test) è di 125 MB/s (vedi datasheet B.3). Questo vuol dire che la capacità di scrittura su disco risulta superiore del 50% circa rispetto alla velocità di trasferimento dati dal digitizer al calcolatore.

Ora, ci si aspetta che un'acquisizione dati che preveda un ciclo di readout e scrittura su disco proceda ad una velocità dipendente dal tempo di trasferimento e dal tempo di scrittura.

Invece, utilizzando la struttura proposta dal software Wavedump, si ottengono risultati nell'ordine di appena 2-4 MB/s. Nel listato 2.2 è riportata la

struttura del codice usato da Wavedump.

```
for (i=0; i<MAX_X742_GROUP_SIZE; i++)
    if (Evt->GrPresent[i] == 1)
        for (j=0; j<MAX_X742_CHANNEL_SIZE; j++)
            for (k = 0; k < Evt->DataGroup[i].ChSize[j] ; k++)
                fprintf(wave, "%f\n", Evt->DataGroup[i].DataChannel[j][k]);
```

Listato 2.2: Codice C per il trasferimento e la decodifica utilizzato da Wavedump

Per capire come migliorare il codice, occorre capire come funziona l'I/O buffering nei sistemi Linux[16].

Nei sistemi Linux, la scrittura su disco è asincrona rispetto alla chiamata di sistema. Questo vuol dire che una *system call write* non fa altro che copiare i dati dallo *user-space* al *kernel-space* e poi il kernel in un secondo momento si occupa in modo estremamente ottimizzato di scrivere i dati su disco. Inoltre, il kernel bufferizza i dati prima di scriverli su disco[16]. Quindi, è più conveniente chiedere al sistema operativo di scrivere grossi burst di dati, utilizzando il minor numero possibile di *system call*.

Un codice come quello riportato sopra non si conforma con questa organizzazione del sistema: viene effettuata, nel caso peggiore in cui ogni canale abbia acquisito dati, una *system call* per ogni campione acquisito generando un'overhead notevole. Per ottimizzare questa situazione abbiamo organizzato il codice nella maniera che può essere apprezzata nel listato 2.3.

```
while (go)
{
    CAEN_DGTZ_ReadData (handle,
        CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT, buffer, &bsize);
    if (bsize>0) fwrite(buffer,1,bsize, wave2);
}
```

Listato 2.3: Pseudocodice per il trasferimento ottimizzato

Con un codice di questo tipo è possibile acquisire dati e scriverli su disco a una velocità di circa 80 MB/s. Ciò avviene perché la funzione

```
CAEN\_\_DGTZ\_\_ReadData (int handle, CAEN\_\_DGTZ\_\_ReadMode\_\_t mode,
    char *buffer, uint32\_\_t *bufferSize)
```

trasferisce dati ad 80 Mb/s più l'overhead dato dalla chiamata di funzione, e la funzione *fwrite* impedisce asincronicamente al sistema operativo l'ordine di scrivere su disco i dati acquisiti. Alla seconda iterazione del ciclo, la funzione *fwrite* termina prima della nuova funzione *CAEN\_DGTZ\_ReadData*, viene eseguita una nuova *fwrite*, il ciclo ricomincia con la terza iterazione e così via.

Abbiamo effettuato dei test con tre versioni diverse del codice per suffragare questa tesi. Con ognuna delle tre versione si sono acquisiti cento eventi di 49,2 KB l'uno per cento volte di fila (quindi cento burst da 4920 KB, 4,8 MB circa). La dimensione di ogni evento dipende dall'ampiezza del campionamento e dai canali abilitati per il readout. Per i test abbiamo utilizzato il digitizer v1742 connesso al computer mediante optical link con tutti i gruppi di canali abilitati (tranne i due di fast triggering, vedi datasheet B.2) e un'ampiezza di campionamento di 1024 sample, il massimo consentito da questo digitizer.

Di seguito viene riportato, per ogni versione, il codice utilizzato per le acquisizioni e il grafico con i risultati. L'appendice B riporta i dati risultanti.

#### 2.4.1 Versione 1

Il codice di questa versione è analogo a quello usato da Wavedump (vedi listato 2.4).

```
CAEN_DGTZ_ReadData (handle,
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT, buffer, &bsize);
CAEN_DGTZ_GetNumEvents (handle, buffer, bsize, &numEvents);
```

```

for (i = 0; i < numEvents; i++)
{
    CAEN_DGTZ_GetEventInfo (handle, buffer, bsize, i, &eventInfo,
                            &evtptr);
    X742_DecodeEvent(evtptr, (void**)&Evt742);
    for (k = 0; k<4; k++)
    {
        for (o = 0; o<8; o++)
        {
            for (j = 0; j < Evt742->DataGroup[k].ChSize[o]; j++)
            {
                fprintf (wave2, "%f\n",
                         Evt742->DataGroup[k].DataChannel[o][j]);
            }
        }
    }
    ret = CAEN_DGTZ_FreeEvent (handle, (void **)&Evt742);
}

```

Listato 2.4: Codice C per il trasferimento e la decodifica utilizzato da Wavedump

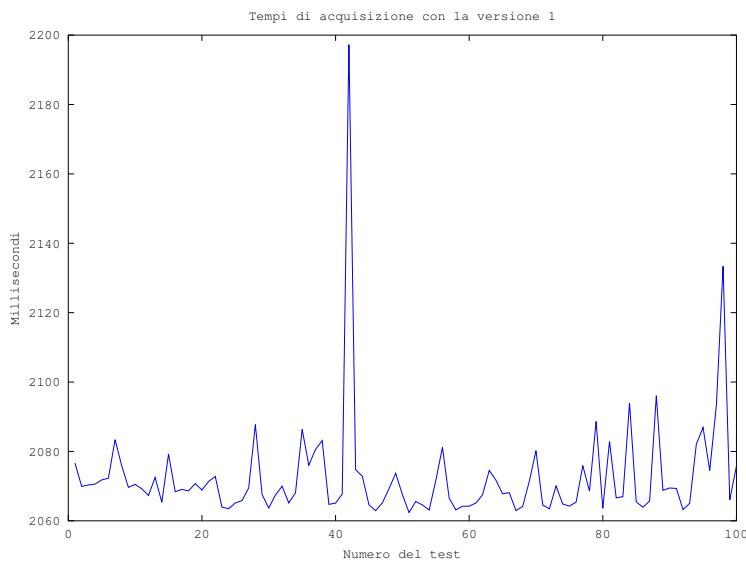


Figura 2.3: Risultati dei test di acquisizione con la versione 1 (vedi sezione 2.4.1), il cui codice è analogo a quello usato da Wavedump

Dai risultati (vedi tabella A.1 e figura 2.3) si è ottenuto un tempo medio di acquisizione di 2072.69 ms con una deviazione standard di 15.93 ms ed un range di 134.86 ms. La velocità di acquisizione media è di 2.32 MB/s, in linea con i risultati ottenibili usando Wavedump.

#### 2.4.2 Versione 2

Lettura dei dati e scrittura su disco carattere per carattere (vedi listato 2.5).

```
CAEN_DGTZ_ReadData (handle,
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT, buffer, &bsize);
for(j=0;j<bsize;j++)
    fprintf (wave2, "%c\n", buffer[j]);
```

Listato 2.5: Codice C per il trasferimento e la scrittura dei dati su disco carattere per carattere

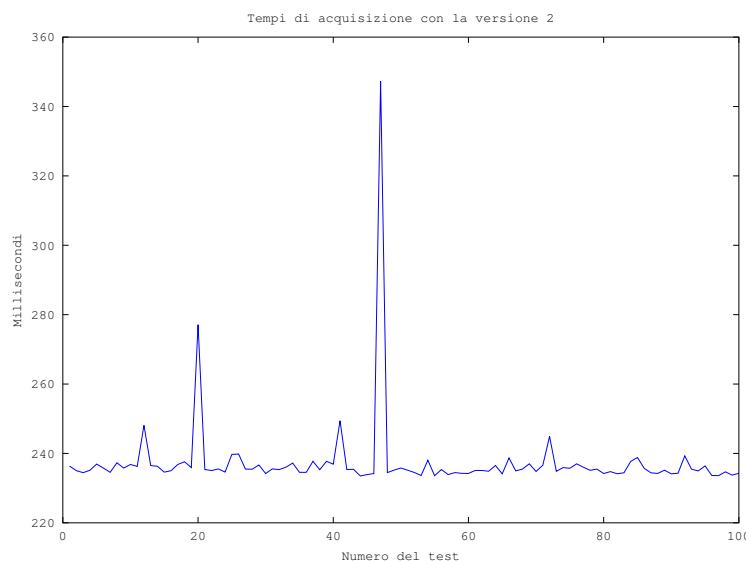


Figura 2.4: Risultati dei test di acquisizione con la versione 2 (vedi sezione 2.4.2), lettura dei dati e scrittura su disco carattere per carattere

Dai risultati (vedi tabella A.2 e figura 2.4) si è ottenuto un tempo medio di acquisizione di 237.43 ms con una deviazione standard di 12.09 ms e un range di 113.74 ms. La velocità di acquisizione media è di 20.21 MB/s, migliore della precedente ma distante dal massimo raggiungibile.

### 2.4.3 Versione 3

Lettura dei dati e scrittura su disco bufferizzata (vedi listato 2.6).

```
CAEN_DGTZ_ReadData (handle,
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT, buffer, &bsize);
if (bsize>0)
    fwrite(buffer,1,bsize,wave2);
```

Listato 2.6: Codice C per il trasferimento e la scrittura dei dati su disco usando fwrite

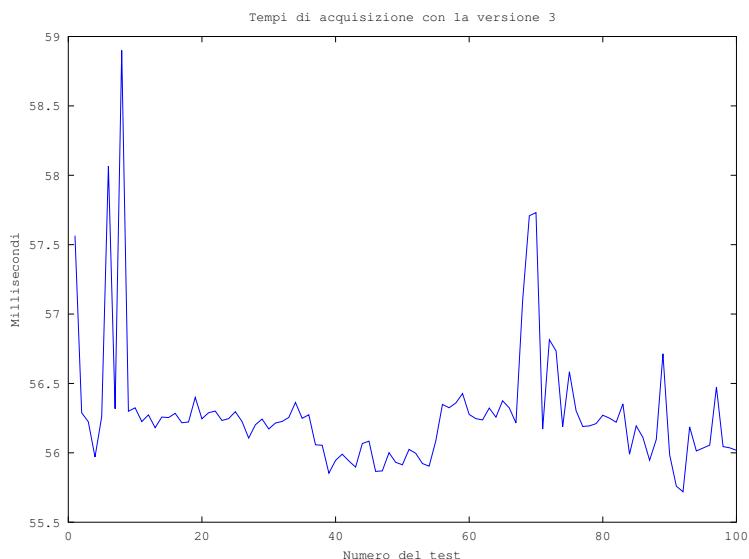


Figura 2.5: Risultati dei test di acquisizione con la versione 3 (vedi sezione 2.4.3), lettura dei dati e scrittura su disco bufferizzata.

Dai risultati (vedi tabella A.3 e figura 2.5) si è ottenuto un tempo medio di acquisizione di 56.28 ms con una deviazione standard di 0.46 ms e un range

di 3.18 ms. La velocità di acquisizione media è di 85.29 MB/s, maggiore di quanto promesso dalla casa costruttrice (vedi datasheet B.2).

Visti i risultati riportati e viste le considerazioni precedenti, possiamo affermare che la Versione 3 è da preferire. In figura 2.6 sono confrontati graficamente i tempi di acquisizione.

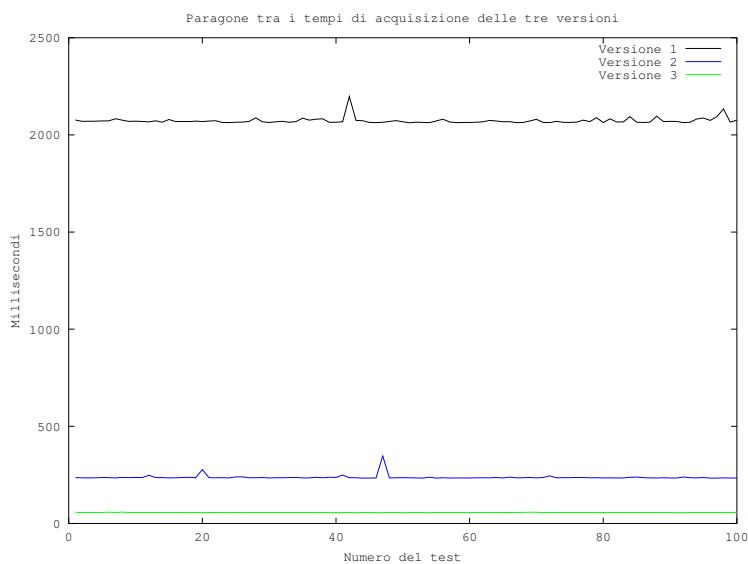


Figura 2.6: Paragone tra i tempi di acquisizione delle tre versioni

## 2.5 Decodificare offline i dati

Nel paragrafo precedente, abbiamo studiato una soluzione per salvare su disco i dati acquisiti sfruttando l'architettura del sistema operativo. A questo punto, il problema che si pone è quello di decodificare i dati acquisiti e di salvarli su disco. Questa operazione, per rendere massima la velocità di acquisizione, è da effettuare offline, ad acquisizione conclusa, oppure può essere eseguita parallelamente da un altro flusso di esecuzione. Per effettuarla

occorre apportare una modifica al codice della versione 3, salvando su disco anche la dimensione del burst di dati da decodificare (vedi listato 2.7).

```
CAEN_DGTZ_ReadData (handle,
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT, buffer, &bsize);
if (bsize>0)
{
    fwrite(buffer,1,bsize,wave2);
    fwrite(&bsize,1,sizeof(int),wavedimension);
}
```

Listato 2.7: Codice C per trasferimento e scrittura dei dati su disco usando `fwrite`, salvando anche la dimensione del burst scritto

Questo è necessario perché i bursts di eventi salvati non hanno tutti la stessa dimensione: settando il registro corrispondente a *MAX\_NUM\_EVT-BLT*, abbiamo garantito che gli eventi acquisiti siano al massimo quelli indicati da *MAX\_NUM\_EVT\_BLT*, ma potrebbero anche essere meno. Inoltre, si potrebbe voler salvare in un singolo file più burst di dati con una dimensione di per sé diversa (perché sono stati, ad esempio, disattivati o attivati dei canali nel readout). Si rileva anche che le funzioni *GetNumEvents* e *GetEventInfo* abbisognano della dimensione del burst di dati e tale dimensione è ottenuta dalla funzione di readout *CAEN\_DGTZ\_ReadData* che non è utilizzata ad acquisizione conclusa. Quindi, l'idea sottostante alla scrittura di una applicazione di decodifica offline è questa:

1. Ricavare il numero di burst di dati da decodificare.
2. Per ogni burst passare alla funzione di decoding il puntatore al burst corrispondente nello stream di dati e passare la dimensione corrispondente del burst di dati alle funzioni *CAEN\_DGTZ\_GetNumEvents* e *CAEN\_DGTZ\_GetEventInfo*.

A questo punto, subentra un problema notevole legato alla struttura della libreria CAENDigitizer. Il digitizer più sofisticato a nostra disposizione è

il modello v1742. Le librerie CAENDigitizer offrono per i digitizer della famiglia 742 un tipo di dato dedicato, strutturato come nel listato 2.8.

```
typedef struct
{
    uint32_t ChSize[MAX_X742_CHANNEL_SIZE]; //The number of samples
                                                stored in DataChannel array
    float *DataChannel[MAX_X742_CHANNEL_SIZE]; //The array of ChSize
                                                samples
    uint32_t TriggerTimeTag;
    uint16_t StartIndexCell;
} CAEN_DGTZ_X742_GROUP_t;

typedef struct
{
    uint8_t GrPresent[MAX_X742_GROUP_SIZE]; //If the group has data
                                              the value is 1 otherwise is 0
    CAEN_DGTZ_X742_GROUP_t DataGroup[MAX_X742_GROUP_SIZE]; //The
                                              array of ChSize samples
} CAEN_DGTZ_X742_EVENT_t;
```

Listato 2.8: Strutture dati per salvare i dati decodificati forniti dalle librerie CAENDigitizer per i modelli della famiglia 742

Inoltre, vengono rese disponibili delle funzioni *open source* per eseguirne il decoding, cioè per depositare i dati contenuti nel burst acquisito in una struttura di tipo *CAEN\_DGTZ\_X742\_EVENT\_t*.

Il problema è che per gli altri digitizer è proposta una soluzione diversa: per eseguire il decoding è necessario che l'applicazione sia interfacciata col digitizer. Infatti, la funzione *CAEN\_DGTZ\_DecodeEvent* non può essere definita polimorfa: al suo interno non viene eseguito l'algoritmo di decoding corretto in base al tipo di dato che le viene passato, ma l'algoritmo viene scelto in base alle informazioni sul digitizer usato ottenute interrogando il digitizer stesso. Di conseguenza, la funzione *CAEN\_DGTZ\_DecodeEvent* non funziona se non è preceduta da una funzione *CAEN\_DGTZ\_OpenDigitizer*. Inoltre, a differenza della funzione di decoding specifica per il modello v1742, il codice

sorgente non è disponibile quindi non è possibile modificarlo.

Questa è una limitazione nell'utilizzo delle librerie CAENDigitizer. Una soluzione può essere la scrittura di funzioni di decoding apposite, oppure è possibile chiedere all'azienda di fornire delle funzioni per il decoding offline per gli altri digitizer, analoghe a quella offerta per i digitizer della famiglia 742. La seconda soluzione è da preferire perché si presuppone che l'azienda disponga già di algoritmi di decoding abbondantemente testati e inseriti tutti nella generica funzione *CAEN\_DGTZ\_DecodeEvent*.

L'affermazione precedente è suffragata dalla seguente prova. Non si dispone del codice sorgente, però si hanno a disposizione strumenti di debugging sofisticati come Valgrind che, tra l'altro, possono visualizzare a run time le funzioni lanciate dall'applicazione. Si è, dunque, provato a lanciare in ambiente Valgrind un'applicazione che esegue la funzione *CAEN\_DGTZ\_DecodeEvent* con due digitizer diversi, uno della famiglia 742 (v1742) e uno della famiglia 751 (dt5751). Si è ottenuto il seguente risultato: nel primo caso la funzione *CAEN\_DGTZ\_DecodeEvent* chiama una funzione diversa rispetto a quella chiamata nel secondo caso. Quindi, l'azienda CAEN dispone già di algoritmi di decoding diversi a seconda del tipo di dato da utilizzare ma questi algoritmi sono scelti non in base al tipo di dato ma in base al modello di digitizer e questa informazione è ricavata interfacciandosi con esso.

L'azienda non ha accettato di fornire il codice sorgente della funzione *CAEN\_DGTZ\_DecodeEvent* ma ha accettato di fornire funzioni di decoding ad hoc su richiesta degli interessati.

## 2.6 Problema relativo alla compilazione dei driver

Un'applicazione può interfacciarsi ai digitizer o mediante USB 2.0 o mediante optical link. Per farlo occorre eseguire il *download* degli appositi driver dal sito dell'azienda ed installarli (vedi sezione 2.2). Lo strumento di installazione offerto è molto semplice, un makefile con due target: all ed install. Si è verificato usando i digitizer che i driver vanno reinstallati ogni qualvolta venga modificato il kernel del sistema operativo. Questa è una procedura semplice ma non conoscerla può provocare diversi disgradi: basta aggiornare il sistema operativo e i driver smettono di funzionare con conseguente fallimento di ogni applicazione che vuole interfacciarsi al digitizer.

È buona norma, dunque, reinstallare i driver ad ogni aggiornamento del sistema.

# Capitolo 3

## Sviluppo di ObjectDump

Viste le limitazioni del software *open source* Wavedump rispetto alle esigenze del WP09 di Eurogammas[1][2] trattate nel capitolo 2 (attinenti, in particolare, alla velocità di acquisizione), si è proceduto alla costruzione del nuovo sistema di controllo e di acquisizione dati ObjectDump.

Di seguito, verrà illustrata la prima fase di prototipazione, verranno esposte le caratteristiche esterne e interne del sistema, verranno presentati i lati server, client e i programmi di decodifica; infine, verrà illustrata l'installazione di ObjectDump e verranno spiegati i parametri del suo file di configurazione.

Il sistema ObjectDump è formato da tre parti:

- ObjectDump lato server, realizzato in C++.
- ObjectDump lato client, realizzato in C.
- Applicazione di decodifica dei dati acquisiti, realizzata in C.

Lo sviluppo del sistema tiene conto delle considerazioni sulla velocità di acquisizione e sulla decodifica offline fatte nel capitolo precedente. Il lato server del sistema è quello principale: gli altri due, essendo più piccoli sia in termini di grandezza che di complessità, possono essere sostituiti con implementazioni diverse.

Per spiegare il funzionamento del lato client e rendere più chiaro quello del lato server, si è deciso di illustrare un caso concreto: l'aggiunta di un nuovo comando. In questo modo il lettore può capire le caratteristiche fondamentali del lato client e procedere, se ritiene il caso, ad una propria implementazione.

I sorgenti dei tre lati del sistema sono completamente disaccoppiati tra loro e ognuno di essi può essere compilato separatamente dagli altri.

### **3.1 Sviluppo di un prototipo**

Il sistema ObjectDump utilizza le librerie CAENDigitizer: visto che il codice prodotto può essere definito a basso livello, nel senso che il suo scopo è l’interazione con un hardware, si è ritenuto necessario iniziare lo sviluppo implementando un prototipo. Per prototipo intendiamo un programma costruito con lo scopo di familiarizzare con le funzioni offerte dalla libreria e per far luce sull’effettivo funzionamento del digitizer. Inoltre, la sua costruzione permette di evidenziare alcune problematiche progettuali ed implementative.

Innanzitutto, si è partiti dalla dal codice di Wavedump. Poi, si è utilizzato un esempio di software di readout di base offerto dall’azienda nella documentazione delle librerie CAENDigitizer[13].

Partendo dal software di base lo si è via via ampliato. Prima si sono volute testare le funzioni offerte dalla libreria CAENDigitizer[13], poi si è cercato di dare al prototipo una sua organicità dividendone i compiti in threads. Utilizzando questo prototipo si sono effettuati i test sui tempi di acquisizione illustrati nel capitolo 2.

#### **3.1.1 Aspetti emersi dallo sviluppo del prototipo**

Sviluppando il prototipo si sono evidenziati alcuni aspetti da tenere in considerazione per la costruzione del sistema finale:

- Il sistema finale deve avere un meccanismo consistente per gestire i parametri di configurazione del digitizer.
- Il sistema finale deve gestire l’input da tastiera o via TCP/IP senza

generare rallentamenti al flusso del programma.

- Il sistema finale deve gestire l'acquisizione seguendo i risultati riportati nel capitolo 2, cioè separando l'acquisizione dalla decodifica, salvando direttamente su disco i dati non ancora decodificati per poi, eventualmente, decodificarli offline ad acquisizione conclusa.

## 3.2 Caratteristiche esterne del nuovo sistema

Il sistema ObjectDump presenta le caratteristiche esterne descritte di seguito, che rispecchiano l'esigenza di avere un sistema che possa essere padroneggiato completamente dall'utente sia per la potenza delle configurazioni offerte, sia per la conoscibilità delle operazioni svolte.

### 3.2.1 Interfaccia utente

ObjectDump è controllabile da linea di comando, l'interfaccia è user-friendly nel senso che il programma non si interrompe se viene inserito un comando non riconosciuto o un carattere numerico.

### 3.2.2 Aiuti all'utente

ObjectDump fornisce un meccanismo per rintracciare immediatamente se una funzione di configurazione del digitizer non ha dato l'esito sperato. Le funzioni della libreria CAENDigitizer ritornano un codice d'errore: il sistema lo interpreta e lo pone all'attenzione dell'utente. Queste informazioni vengono salvate nel file di log.

### 3.2.3 File di configurazione

ObjectDump presenta un'opzione che consente di stampare a video i parametri letti dal file di configurazione. Questa funzionalità consente di capire se il sistema stia configurando il digitizer nella maniera voluta.

### 3.2.4 Affidabilità

Il programma non si interrompe se l’utente inserisce una successione di comandi generanti un flusso di esecuzione inconsistente. Se, ad esempio, l’utente richiede di iniziare l’acquisizione senza aver inizializzato il digitizer, il sistema non si interrompe ma avverte l’utente di inserire i comandi necessari alla ripresa del corretto funzionamento. Questo evita all’utente di riavviare il sistema qualora abbia inserito un’erronea successione di comandi.

### 3.2.5 Controllo completo sul digitizer

Il programma consente di leggere e di scrivere un registro del digitizer a *run time* rendendolo completamente controllabile.

### 3.2.6 Supporto TCP/IP

Ogni comando impartibile ad ObjectDump da tastiera può essere imparato via TCP/IP utilizzando il lato client del sistema, ovvero l’applicazione ObjectDumpClient.

### 3.2.7 Documentazione con Doxygen

Il codice è documentato seguendo le convenzioni di Doxygen.

## 3.3 Caratteristiche interne del nuovo sistema

Il sistema funziona su sistemi operativi Linux e utilizza le librerie C CA-ENDigitizer. Di seguito, verranno introdotte le caratteristiche interne più importanti del sistema che rendono il suo codice manutenibile, robusto ed efficiente.

### 3.3.1 Acquisizione multithreaded

L’acquisizione presenta il seguente aspetto multithread: un thread acquisisce i dati e li scrive su disco, un altro li visualizza e un altro li preprocessa. I thread possono essere attivati o disattivati dall’utente separatamente gli uni dagli altri mediante comandi appositi (vedi sezione 3.10).

### 3.3.2 Manutenibilità del codice e defensive programming

Il codice prodotto è pensato per poter essere modificato dagli utilizzatori del sistema. Essi sono tendenzialmente ricercatori o esperti che devono poter facilmente modificare il codice per adeguarlo alle loro esigenze. Sono evitate, quindi, variabili globali, costrutti *go-to* e l’utilizzo dei puntatori è fatto seguendo i principi del *defensive programming*.

### 3.3.3 Paradigma a oggetti

Per aumentarne la modularità, il sistema è costruito seguendo il paradigma ad oggetti, utilizzando prevalentemente lo schema della composizione e, all’occorrenza, il *singleton design pattern*.

## 3.4 Sviluppo del lato server del sistema

In questa sezione si tratterà dello sviluppo del lato server di ObjectDump, il cuore del sistema. Esso è scritto in C++ ed è organizzato interamente ad oggetti per renderlo modulare. Di seguito, verranno descritti i moduli principali di cui si compone.

L’implementazione di ogni classe è separata dalla sua definizione, come si conviene ad un programma orientato agli oggetti. La dimensione del codice sorgente supera le diecimila linee di codice. Se il numero di linee di codice è un indicatore banale per descrivere la grandezza di un software, rende

comunque l'idea della vastità del lavoro: in questa sede verrà esposto il funzionamento degli oggetti più importanti che costituiscono la struttura di base del programma (vedi diagramma di classe in figura 3.1).

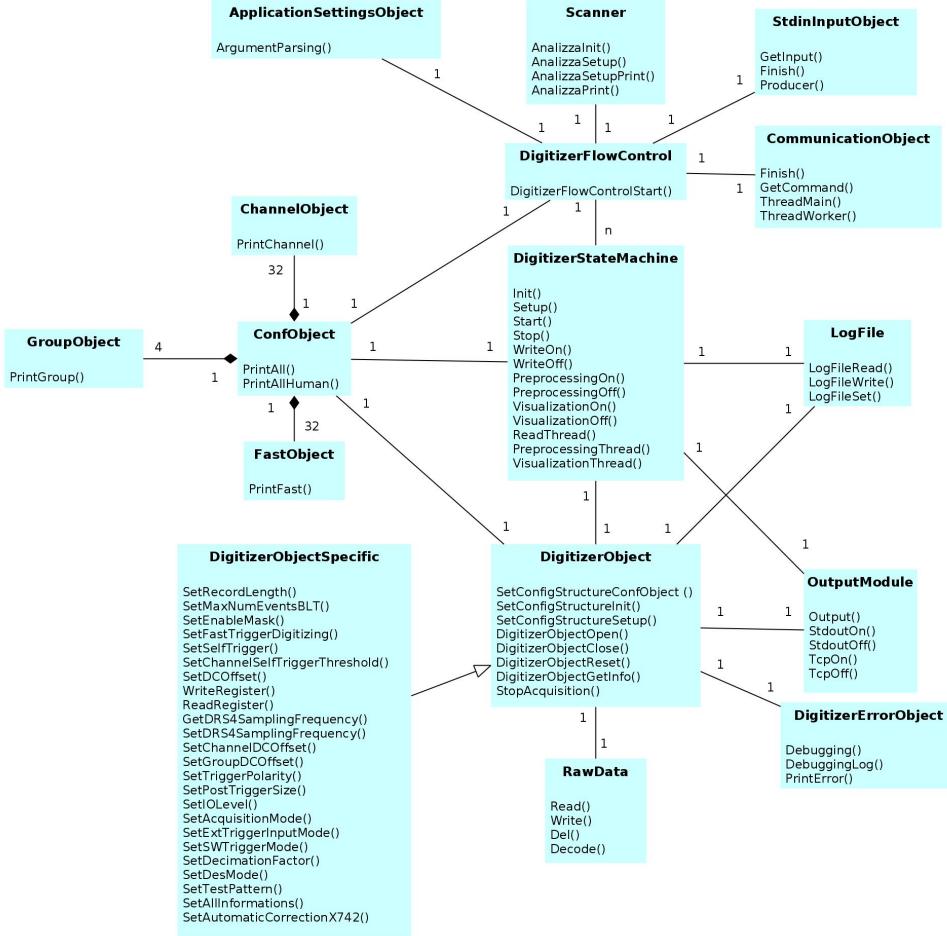


Figura 3.1: Diagramma di classe del lato server del sistema

Due aspetti rilevanti dell'implementazione sono i seguenti:

- Il codice non presenta variabili globali. Quando è necessario che un oggetto sia unico per tutti è utilizzato il *singleton design pattern*[18].

- Per gestire i threads è utilizzata la libreria thread dello standard 11 del C++.

Utilizzare i *singleton* invece delle variabili globali statiche e la libreria thread invece della libreria pthread, rende il codice maggiormente orientato agli oggetti. I *singleton*, infatti, sono la maniera orientata agli oggetti per gestire globalmente delle informazioni e la libreria thread consente di creare threads mantenendo la rientranza del codice: la libreria pthread richiede informazioni statiche che avrebbero reso il codice non rientrante.

### 3.4.1 Creazione dello scanner per leggere il file di configurazione

Come primo passo della costruzione del lato server del sistema si è creato lo scanner per il file di configurazione. Per farlo si è utilizzato il programma Flex. Flex è un generatore automatico di scanner lessicali, particolarmente diffuso nell'ambiente GNU/Linux. Esso consente di scrivere uno scanner lessicale focalizzandosi nella costruzione delle espressioni regolari e sulle azioni da intraprendere per ogni pattern (cioè per ogni stringa riconosciuta). I particolari del trattamento dell'input e quelli relativi all'allocazione della memoria necessaria alle operazioni di analisi lessicale vengono gestiti da Flex.

### 3.4.2 Salvataggio delle informazioni ottenute dal file di configurazione

Per avere un meccanismo per salvare le informazioni ottenute scannellizzando il file di configurazione, si è costruito l'oggetto *ConfObject*. Tale oggetto raccoglie le informazioni lette dallo scanner e, se l'utente lo richiede, le stampa a video (vedi sezione 3.10).

Esso viene passato come argomento allo scanner lessicale e ai costruttori degli oggetti che interagiscono col digitizer.

*ConfObject* è un oggetto composto nel senso che possiede come attributi gli

oggetti *GroupObject*, *ChannelObject* e *FastObject* che servono per raccogliere in modo organizzato le informazioni di configurazione specifiche di un gruppo, di un canale normale e di un canale *fast*.

### **3.4.3 Wrapper attorno alle funzioni della libreria CAENDigitizer**

Si è creato un oggetto *DigitizerObject* che dispone dei metodi per comunicare con il digitizer utilizzando le informazioni contenute nel *ConfObject*. Tale oggetto si costruisce passandogli come argomenti il *ConfObject*. Ogni suo metodo lancia una funzione della libreria CAENDigitizer. Organizzando le funzioni della libreria CAENDigitizer in questo modo si ottiene un livello di astrazione più alto: operazioni complesse come la configurazione dei parametri di più canali viene fatta in automatico lanciando un metodo dell'oggetto *DigitizerObject*, il quale utilizza le informazioni contenute nel *ConfObject*.

### **3.4.4 Gestione dei codici di errore**

Ogni funzione della libreria CAENDigitizer restituisce un codice di errore. Il significato del codici di errore è stabilito dall'azienda CAEN[13]. Si è creato un oggetto *DigitizerErrorObject* con lo scopo di interpretare i codici di errore stampandone il significato. L'oggetto può stampare il significato dei codici di errore o a video o sul file di *log*. Inoltre, viene indicato il nome della funzione che ha generato l'errore e la linea di codice dove l'errore è avvenuto.

Un'opportuna combinazione di questo oggetto, dell'oggetto *DigitizerObject* e di un oggetto per gestire la scrittura e lettura di un file di *log* (vedi sezione 3.4.11) consente di impartire ordini al digitizer registrando la riuscita o meno di un'operazione.

### 3.4.5 La macchina a stati

Si è deciso di gestire l’interazione con il digitizer basandosi su una macchina a stati, caratterizzata dagli stati seguenti:

- Inizializzazione del digitizer
- Configurazione del digitizer
- Inizio acquisizione
- Fine acquisizione
- Chiusura del digitizer

Per implementarli, occorre lanciare più di un metodo dell’oggetto *DigitizerObject*: a tal scopo si è creato l’oggetto chiamato *DigitizerStateMachine*.

L’operazione di settaggio del digitizer, ad esempio, necessita di più operazioni semplici eseguibili dall’oggetto *DigitizerObject*: l’oggetto *DigitizerStateMachine* presenta un metodo per ordinare all’oggetto *DigitizerObject* di eseguire le operazioni necessarie al settaggio.

### 3.4.6 Organizzazione di un modello producer-consumer

Si è dimostrato che, eseguendo una *system call fwrite* successivamente alla funzione *CAEN\_DGTZ\_ReadData*, è possibile acquisire e salvare i dati su disco a 80 MB/s circa mediante optical link con il digitizer v1742. L’esecuzione di operazioni sui dati quali la loro visualizzazione con Gnuplot e il loro preprocessamento non devono rallentare il flusso di scrittura su disco. Per risolvere questo problema ObjectDump implementa un’architettura multithreaded basata sul modello *producer-consumer*. Il modello è implementato nell’oggetto *DigitizerStateMachine* (vedi listato 3.1).

Il thread principale legge i dati disponibili con la funzione *CAEN\_DGTZ\_-ReadData*, esegue la funzione *fwrite* (vedi sezione 2.4) e copia i dati letti nella coda utilizzata dagli altri thread.

Per creare e gestire i thread è utilizzata la libreria *thread* inclusa nello standard 11 del C++. Questa scelta è stata obbligata per rendere il codice rientrante: la libreria C *pthread* richiede che le funzioni eseguite dai thread siano statiche e questo non si addice a un programma orientato agli oggetti.

```

while (go_general)
{
    tmp.RawDataRead ();
    if (tmp.bsize > 0)
    {
        if (go_raw_data && go_general)
        {
            tmp.RawDataWriteOnFile (application_setup->
                ApplicationSetupGetDataFilePunt (),
                application_setup->
                ApplicationSetupGetDataFileSizePunt ());
        }

        ReservedPreprocessingInputAreaHandle.lock();
        if (go_preprocessing && go_general)
        {
            if (num_mex_preprocessing < PREPROCESSINGQUEUE)
            {
                circular_buffer_preprocessing[coda_preprocessing] = tmp;
                if (coda_preprocessing == PREPROCESSINGQUEUE - 1)
                    coda_preprocessing = 0;
                else
                    coda_preprocessing = coda_preprocessing + 1;
                num_mex_preprocessing++;
                Acquisition_Cond1.notify_one();
            }
        }
        ReservedPreprocessingInputAreaHandle.unlock();

        ReservedVisualizationInputAreaHandle.lock();
        if (go_visualization && go_general)

```

```

{
    if (num_mex_visualization < VISUALIZATIONQUEUE)
    {
        circular_buffer_visualization[coda_visualization] = tmp;
        if (coda_visualization == VISUALIZATIONQUEUE - 1)
            coda_visualization = 0;
        else
            coda_visualization = coda_visualization + 1;
        num_mex_visualization++;
        Acquisition_Cond2.notify_one();
    }
}
ReservedVisualizationInputAreaHandle.unlock();
}
}

```

Listato 3.1: Codice eseguito dal thread produttore dell'oggetto DigitizerStateMachine

È da sottolineare che il thread produttore, qualora la coda di uno degli altri due thread sia piena, non si ferma su una variabile di condizione ma continua il suo flusso di esecuzione: si è operata questa scelta perché la priorità è quella di acquisire i dati dal digitizer e di salvarli su disco alla massima velocità possibile. Se la coda di un thread che compie operazioni onerose sui dati (ad esempio quello che li visualizza con Gnuplot) si riempie, il rallentamento rimane confinato a quel thread. La dimensione delle code dei thread è definita in un file apposito, DefineGeneral.h, che contiene tutte le impostazioni di define del lato server del sistema.

### 3.4.7 Strumenti per gestire l'input via tastiera e via TCP/IP

Per gestire l'input si è creato un oggetto per organizzare quello proveniente da stdin e uno per organizzare quello proveniente via TCP/IP.

L'oggetto che gestisce l'input da tastiera contiene due thread, il primo raccolge l'input da tastiera e il secondo fornisce tale input all'oggetto richiedente. L'oggetto che gestisce l'input via TCP/IP ha un'architettura multithread più

complicata: un thread principale ascolta le connessioni in ingresso e crea un thread worker per ogni client che si connette al sistema. I messaggi inviati dai client vengono depositati in un buffer circolare dai thread worker. Dal buffer circolare vengono inviati agli oggetti che li richiedono.

### **3.4.8 L'oggetto FlowControl**

L'oggetto *DigitizerFlowControl* organizza gli oggetti sopra esposti generando il flusso di esecuzione del programma: esso controlla, facendo richiesta ai due oggetti di gestione dell'input, se è presente un input e lancia il metodo corrispondente dell'oggetto *DigitizerStateMachine*. L'oggetto *DigitizerFlowControl* continua a raccogliere input finché il programma non termina.

### **3.4.9 La funzione getopt**

Gli argomenti delle funzione *main* del sistema sono raccolti usando la funzione *getopt*, una funzione GNU di uso comune per analizzare gli argomenti del main. Il listato 3.2 è tratto dal manuale della funzione[17]. Si sottolinea che il terzo argomento della funzione *getopt* deve essere costruito come una stringa che concatena tutti i flag possibili con l'accortezza che, se dopo un flag occorre un valore, il carattere corrispondente va seguito da ':'.

The following trivial example program uses getopt() to handle two program options: -n, with no associated value; and -t val, which expects an associated value.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int flags, opt;
    int nsecs, tfnd;
```

```
nsecs = 0;
tfnd = 0;
flags = 0;
while ((opt = getopt(argc, argv, "nt:")) != -1)
{
    switch (opt)
    {
        case 'n':
        flags = 1;
        break;

        case 't':
        nsecs = atoi(optarg);
        tfnd = 1;
        break;

        default: /* '?' */
        fprintf(stderr, "Usage: %s [-t nsecs] [-n] name\n",
                argv[0]);
        exit(EXIT_FAILURE);
    }
}

printf("flags=%d; tfnd=%d; nsecs=%d; optind=%d\n",
       flags, tfnd, nsecs, optind);

if (optind >= argc)
{
    fprintf(stderr, "Expected argument after options\n");
    exit(EXIT_FAILURE);
}
printf("name argument = %s\n", argv[optind]);

/* Other code omitted */

exit(EXIT_SUCCESS);
}
```

Listato 3.2: Esempio manualistico dell'utilizzo della funzione getopt()[17]

### 3.4.10 L'oggetto OutputModule

L'oggetto *OutputModule* è una soluzione semplice per gestire l'output del sistema e segue il *singleton design pattern*. Può essere usato con beneficio quando l'output è sequenziale rispetto all'input. Ad esempio: l'utente richiede di iniziare l'acquisizione, il programma inizia l'acquisizione e viene stampato a video la scritta 'Starting acquisition'.

Essenzialmente, l'oggetto memorizza se il comando ricevuto in input è stato inviato da tastiera o via TCP/IP e in questo caso memorizza il *socketid* dell'utente. Poi, quando viene chiamato il metodo *output\_module->Output('Stringa')*, 'Stringa' è stampata su stdout oppure è inviata al client via TCP/IP.

Il metodo non funziona quando l'invio dell'output è spostato rispetto all'invio dell'input. Se, ad esempio, l'utente richiede di meccicare un insieme di valori acquisiti, il server può voler inviare la risposta un po' di tempo dopo e, soprattutto, il server può voler eseguire nel frattempo altri comandi di altri utenti. Quindi, occorre registrare da qualche parte la richiesta e inviare la risposta in un momento successivo: in questo caso l'oggetto *OutputModule* non può essere usato.

L'oggetto *OutputModule* è frutto di un'idea semplice: quando si scrive un programma in locale si utilizza l'stdout, cioè si tendono ad usare le funzioni *printf* o *cout*. Quindi, per iniziare a scrivere un'interfaccia TCP/IP, si è pensato di creare una funzione che non stampasse l'output del programma solo su stdout ma che lo inviasse anche via TCP/IP all'utente che lo ha richiesto. Quello riportato nel listato 3.3 è un esempio tratto dal file DigitizerObject.cpp.

```
//Ottengo una istanza di OutputModule
OutputModule *output_module;
```

```
output_module = OutputModule::Instance ();  
  
//Invio una stringa di output  
output_module->Output("My output\n");  
  
//Compongo una stringa di output  
bzero (stringa, STANDARDBUFFERLIMIT);  
snprintf (stringa, STANDARDBUFFERLIMIT, "Board model: %s\n",  
BoardInfo.ModelName);  
  
//Invio la stringa composta  
output_module->Output(stringa);
```

Listato 3.3: Codice per ottenere un’istanza dell’oggetto OutputModule

In tale esempio, viene ottenuta un’istanza dell’oggetto *OutputModule*, è composta la stringa ’stringa’ da stampare in output e poi è chiamato il metodo *output\_module->Output(stringa)*.

#### 3.4.11 Il file di log

Per aiutare l’utente a capire se le operazioni eseguite hanno avuto successo, si è creato un oggetto per gestire la scrittura e la lettura del file di *log*. Il percorso del file di *log* è scelto dall’utente al lancio del programma o in *run time*. Se una operazione ha riportato un codice di errore, il programma avverte l’utente che può visualizzare il contenuto del file di *log* con il comando ’more’ (vedi sezione 3.10).

### 3.5 Il lato client ed esempio di implementazione di un comando

Il lato client di ObjectDump è sviluppato in linguaggio C: è meno complesso rispetto al server. Esso si compone di una funzione main, una funzione per il parsing dei comandi, un thread per leggere l’input da tastiera e uno per ascoltare i messaggi provenienti dal server. In questa sezione è spiegato il

funzionamento del lato client del sistema fornendo una guida per aggiungere al sistema un nuovo comando.

Come spiegato in precedenza, uno dei problemi da risolvere per scrivere un sistema di acquisizione è quello di fornirgli un'interfaccia TCP/IP. La soluzione adottata lato server è la creazione di un thread main che accetti le connessioni in entrata e la creazione di un thread worker per ogni client per ascoltare i messaggi in ingresso. Il responsabile della creazione di questi thread è l'oggetto del server *CommunicationObject*.

Dall'altro lato, deve esserci un client in grado di connettersi al server e di inviargli messaggi validi. Il codice sorgente del client è contenuto nella cartella SourceCodeClient e viene compilato con il comando make client. Nel file DefineClient.h ogni codice che può essere potenzialmente inviato al server è definito con un nome: in questo modo ci si può riferire al comando da inviare con un nome invece che con un numero. Ad inizio file, è contenuta un'altra definizione utile: *STANDARDBUFFERLIMIT* che indica la dimensione massima del buffer che può essere inviato al server. Il resto del codice è contenuto nel file ClientApplication.c.

La variabile *go* controlla il flusso del programma: quando viene settata a zero il programma termina.

Nell'array *char inputline[STANDARDBUFFERLIMIT]* viene salvato l'input ricevuto da tastiera.

L'array *char sendline[STANDARDBUFFERLIMIT]* è il buffer che verrà inviato al server.

L'array *char receiveline[STANDARDBUFFERLIMIT]* contiene i dati ricevuti dal server.

Di seguito verrà illustrato il funzionamento della funzione main del lato client del sistema, della funzione *ricevitore*, *command\_parser* e *reg\_matches*; infine,

verrà illustrato come aggiungere un comando al sistema.

### 3.5.1 Funzione main del lato client

Inizialmente, viene ottenuto l'indirizzo del server dagli argomenti del main (vedi listato 3.4). Lavorando in locale, l'indirizzo è 127.0.0.1. Poi, vengono predisposti gli elementi necessari a connettersi al server. Gli elementi del buffer da inviare al server (*sendline*) vengono settati con un preambolo di tre caratteri che costituiscono un codice che consente al client di inviare al server messaggi riconoscibili. Quindi, quando il server riceve un messaggio, controlla che i primi tre caratteri siano '25', '21', '27' e, se non sono quelli, il messaggio viene scartato (i tre numeri sono stati scelti arbitrariamente, si è solamente voluto cercare un modo semplice per controllare la consistenza dei messaggi scambiati).

```
int main (int argc, char **argv)
{
    //Variabili per ottenere l'indirizzo del server dagli argomenti
    //del main.
    const char *server_address;
    server_address == NULL;
    int c = 0;
    int flag_arg = 0;
    opterr = 0;
    while ((c = getopt (argc, argv, "i:")) != -1)
        switch (c)
    {
        case 'i':
            server_address = (char *)malloc(strlen(optarg) + 1);
            strcpy((char *)server_address, optarg);
            flag_arg = 1;
            break;
        case '?':
            if (optopt == 'i')
                fprintf (stderr, "Option -%c requires an argument.\n",
                        optopt);
            else if (isprint (optopt))
```

```

        fprintf (stderr, "Unknown option '-%c'.\n", optopt);
    else
        fprintf (stderr, "Unknown option character '\\x%x'.\n",
            optopt);
        break;
}
if (flag_arg == 0)
{
    fprintf(stderr, "You have not insert server address: use -i
        flag\n");
    fprintf(stderr, "usage: [executablepath] -i
        [serveraddress]\n");
    return 1;
}
[...]

```

Listato 3.4: Utilizzo della funzione getopt() nel main dell'applicazione client

Poi, viene creato il thread ricevitore, viene settata a 1 la variabile *go* (il thread ricevitore è controllato da questa variabile nel senso che può procedere solo se *go* vale 1), e inizia il ciclo di *fetching* dell'input da tastiera. Se l'utente digita il comando exit il programma termina (vedi sezione 3.10 e listato 3.5).

```

[...]
//Inserisco queste informazioni dentro sendline in modo tale che
//il server riconosca i dati inviategli.
sendline[0] = 25;
sendline[1] = 21;
sendline[2] = 17;
sendline[4] = '\0';
const char * my_punt;
//thread id del thread che ascolta i messaggi provenienti dal
//server
pthread_t ricevitore;
//Codice necessario all'apertura di una socket con il client.
struct sockaddr_in servaddr, cliaddr;
char recvline[STANDARDBUFLIMIT];
sockfd = socket (AF_INET, SOCK_STREAM, 0);
bzero (&servaddr, sizeof (servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr (server_address);
servaddr.sin_port = htons (1111);

```

```
fprintf(stdout, "Welcome to ObjectDump tcp service, press help for
    getting the available command
list\n");
connect (sockfd, (struct sockaddr *) &servaddr, sizeof (servaddr));
perror(" ");
[...]
//Creo il thread ricevitore, controllato dalla variabile go.
go = 1;
pthread_create(&ricevitore, NULL, ricevitore_function, NULL);
do
{
    fflush(stdout);
    //Prelevo un input da tastiera
    fgets (inputline, STANDARDBUFFERLIMIT, stdin);
    if (inputline[strlen (inputline) - 1] == '\n')
        inputline[strlen (inputline) - 1] = '\0';
[...]
```

Listato 3.5: Codice della funzione main dell'applicazione client 1

La funzione fgets raccoglie al più *STANDARDBUFFERLIMIT* caratteri da stdin e li deposita nel buffer *inputline*. Se l'ultimo carattere raccolto è quello di *newline*, esso viene sostituito con il carattere di fine stringa. Il quarto carattere di *sendline*, *sendline*[3], deve contenere il codice del comando da inviare. Questo codice è ritornato dalla funzione *command\_parser* (*inputline*, *sendline*) (vedi listato 3.6).

È necessario inserire negli argomenti anche *sendline* perché la funzione *command\_parser*, qualora il messaggio da inviare sia la richiesta di lettura o di scrittura su un registro, raccoglie da *inputline* le informazioni necessarie e le inserisce dentro *sendline*.

Se il messaggio da inviare richiede al server il cambiamento del path del file di configurazione, il cambiamento del path dove salvare i dati o il cambiamento del path del file di log, esso viene copiato da *inputline* e messo in *sendline*. Se la funzione *command\_parser* non riesce a trovare in *inputline* un comando valido, restituisce -1. Il ciclo *do-while* finisce quando l'utente inserisce il

comando exit (vedi sezione 3.10).

Al termine del ciclo la variabile *go* è settata a 0 interrompendo, quindi, anche il thread *ricevitore* e la socket di comunicazione col server viene chiusa.

```
[...]
    sendline[3] = (char) command_parser (inputline, sendline);

    if (sendline[3] == CHANGECONF || sendline[3] == CHANGEDATA ||
        sendline[3] == CHANGELOG)
    {
        my_punt = inputline + 3;
        strcpy(sendline + 4, my_punt);
    }

    if (sendline[3] != -1)
    {
        send (sockfd, sendline, STANDARDBUFLIMIT, 0);
    }
}
while (reg_matches (inputline, "^[Ee][Xx][Ii][Tt][ \t]*$") ==
       false);
go = 0;
shutdown (sockfd, 2);
}
```

Listato 3.6: Codice della funzione main dell'applicazione client 2

### 3.5.2 Funzione ricevitore

Il thread ricevitore è controllato da un ciclo *while* che viene eseguito fino a quando la variabile *go* è diversa da zero.

Come prima, con la funzione *bzero*, viene pulito il contenuto del buffer di ricezione *receiveline*.

Poi, viene depositato in *receiveline* quanto mandato dal server. Se la funzione *recv* ritorna un valore negativo, vuol dire che la comunicazione col server è interrotta e quindi il thread *ricevitore* termina. Altrimenti, viene stampato il contenuto del messaggio inviato dal server.

Il ramo *else* contiene il codice necessario a formattare correttamente il testo mandato dal server. Innanzitutto, viene controllato se l'ultimo carattere ricevuto è quello di a capo. Se è così, il messaggio viene stampato altrimenti viene stampato il messaggio seguito da un carattere di *newline*.

La variabile *no\_new\_line == 0* sta a indicare che ci si aspetta la risposta a un comando che non prevede il carattere di a capo alla fine della stringa.

Queste accortezze sono necessarie unicamente per presentare i messaggi ricevuti in modo sufficientemente ordinato e non costituiscono il cuore del client (vedi listato 3.7).

```
//Funzione eseguita dal thread che riceve l'output dal server
void * ricevitore_function (void * nothing)
{
    int i;
    while (go)
    {
        bzero(receiveline, STANDARDBUFFERLIMIT);
        if (recv (sockfd, receiveline, STANDARDBUFFERLIMIT, 0) <= 0)
        {
            fprintf (stdout, "Comunicazione col server interrotta\n");
            shutdown (sockfd, 2);
            go = 0;
        } else {
            if (receiveline[strlen(receiveline)-1] == 10)
                fprintf(stdout, "%s", receiveline);
            else if(receiveline[strlen(receiveline)-1] != 10)
            {
                fprintf(stdout, "%s", receiveline);
                if (strlen(receiveline) > 1 && no_new_line == 0)
                    fprintf (stdout, "\n");
            }
        }
    }
}
```

Listato 3.7: Codice della funzione ricevitore dell'applicazione client

### 3.5.3 Funzione command\_parser

La funzione command parser riceve in input *inputline*, *sendline* e restituisce un intero corrispondente al codice del comando trovato dentro *inputline*. *Sendline* è il buffer inviato al server: è inserito come argomento della funzione *command\_buffer* perché, in caso di comandi complessi, esso viene formattato di conseguenza. Nell'esempio il comando inserito dall'utente è quello di lettura di un registro.

Il codice corrispondente viene ritornato dalla funzione *command\_parser* e quindi inserito in *sendline[3]*; l'indirizzo del registro è inserito in *sendline* dalla funzione *command\_parser* stessa.

Le funzione *FindPointer* è una funzione di libreria dichiarata in *AnalizzatoreUtils.h* e restituisce, data una stringa con più parole come argomento, il puntatore alla seconda parola. In questo caso viene usata per trovare il puntatore all'indirizzo del registro da leggere come si vede nel listato 3.8.

```
int command_parser (char *inputline , char *sendline)
{
    [...]
    if (reg_matches(inputline, "^[iI] [Nn] [Ii] [Tt] [ \t]*$"))
        return INIT;
    else if (reg_matches (inputline, "^[Ss] [Ee] [Tt] [Uu] [Pp] [ \t]*$"))
        return SETUP;
    else if (reg_matches (inputline,
        "^[pP] [rR] [eE] [sS] [tT] [aA] [rR] [tT] [ \t]*$"))
        return PRESTART;
    else if (reg_matches (inputline, "^[pP] [Rr] [Ee] [Ss] [Tt] [Oo] [Pp] [
        \t]*$"))
        return PRESTOP;
    [...]
}
[...]
else if (reg_matches (inputline, "^[Rr] [Ee] [Aa] [Dd] [
    \t]+[Rr] [Ee] [Gg] [Ii] [Ss] [Tt] [Ee] [Rr] [ \t]+(0x[0-9a-fA-F]{1,4}) [
    \t]*$"))
{

```

```

my_punt = FindPointer (inputline);
my_punt = FindPointer (my_punt);
strcpy(sendline + 4, my_punt);
return READREGISTER;
}
[...]

```

Listato 3.8: Codice della funzione command\_parser dell'applicazione client

### 3.5.4 Funzione reg\_matches

La funzione *reg\_matches* è usata per confrontare il contenuto di una stringa con un'espressione regolare. Essa è usata per verificare se l'input inserito da tastiera corrisponde ad un comando. Permette di più rispetto alla funzione *strcmp* perché la funzione *strcmp* compara due stringhe, la funzione *reg\_matches* compara una stringa e un'espressione regolare. Le funzioni con cui è costruita *reg\_matches* sono contenute nella libreria di sistema *regex.h*.

### 3.5.5 Aggiungere un nuovo comando al sistema

L'applicazione client fornisce un'interfaccia per spedire messaggi al server: si connette al server, spedisce il comando da eseguire e si aspetta un'output. Per inviare un nuovo comando dal client basta aggiungere un ramo nell'*if* della funzione *command\_parser*.

Esempio: vogliamo aggiungere il comando *newcommand* che ha codice 33.

Primo step: Aggiungere *newcommand* nel file *DefineClient.h* (vedi listato 3.9).

```

[...]
#define READREGISTER 25
#define PRINTFILES 28
#define STATUS 30
#define NEWCOMMAND 33
[...]

```

Listato 3.9: Codice per aggiungere un nuovo command all'applicazione

client 1

Secondo step: Aggiungere newcommand nella funzione *command\_parser*

Nel caso il comando *newcommand* preveda l'inserimento di parametri occorre modificare *sendline* di conseguenza a partire da *sendline + 4*, cioè a partire dal carattere dopo i 3 di preambolo e il codice del comando. Quindi, a questo punto, il client invia al server il comando *newcommand* seguito da una serie di parametri (vedi listato 3.10).

```
[...]
if (reg_matches(inputline, "^ [iI] [Nn] [Ii] [Tt] [ \t]*$"))
    return INIT;
else if (reg_matches (inputline,
    "^ [Nn] [Ee] [Ww] [Cc] [Oo] [Mm] [Mm] [Aa] [Nn] [Dd] [ \t]*$"))
{
    strcpy(sendline + 4, mystring);
    return NEWCOMMAND;
}
else if (reg_matches (inputline, "^ [Ss] [Ee] [Tt] [Uu] [Pp] [ \t]*$"))
    return SETUP;
[...]
```

Listato 3.10: Codice per aggiungere un nuovo comando all'applicazione client 2

Ora, occorre predisporre il server perché possa capire ed eseguire il comando *NEWCOMMAND*. Come prima cosa occorre aggiungere la definizione del comando nel file DefineCommands.h. Poi, nel caso il comando preveda anche un parametro, bisogna modificare il file CommunicationObject.cpp nel modo riportato nel listato 3.11.

Quindi, se il comando inviato è di tipo *newcommand*, il suo parametro viene copiato nell'attributo *first parameter*.

```
[...]
#define READREGISTER 25
```

```
#define PRINTFILES 28
#define STATUS 30
#define NEWCOMMAND 33
[...]
if (num_mex == MAXCOMMAND)
    BlockedProducerInput.wait(ReservedInputAreaHandle);
coda = (coda + 1) % MAXCOMMAND;
command[coda].command_sent_by_user = buffer[3];
if(buffer[3] == NEWCOMMAND)
{
    bzero (command[coda].first_parameter, STANDARDBUFFERLIMIT);
    my_punt = buffer + 4;
    strncpy (command[coda].first_parameter, my_punt,
              STANDARDBUFFERLIMIT - 1 );
}
if (buffer[3] == CHANGECONF || buffer[3] == CHANGEDATA
|| buffer[3] == CHANGELOG || buffer[3] == WRITEREGISTER
|| buffer[3] == READREGISTER)
{
    bzero (command[coda].first_parameter, STANDARDBUFFERLIMIT);
    my_punt = buffer + 4;
    strncpy (command[coda].first_parameter, my_punt,
              STANDARDBUFFERLIMIT - 1 );
    fprintf (stderr, "command[coda].first_parameter: %s\n",
             command[coda].first_parameter);
}
[...]
```

Listato 3.11: Codice per aggiungere un nuovo command al server 1

Supponiamo che il comando newcommand debba essere seguito da una serie di valori, ad esempio: *newcommand 2012-2013-2014-2015*. Un modo per implementarlo sfruttando l'architettura del programma è quello di separare i valori con un separatore ben preciso, ad esempio i ':' e '-'.

Poi, occorre modificare il file DigitizerFlowControl.cpp per aggiungere un ramo al costrutto *if-else* che determina il tipo di comando da eseguire (vedi listato 3.12). Nell'esempio, il nuovo ramo è stato aggiunto tra il comando start e il comando stop. Teniamo presente che se si vuole spedire un messaggio all'utente che ha inviato il messaggio newcommand basta usare l'oggetto

*output\_module* (vedi sezione 3.4.10).

```
[...]
else if (DigitizerFlowControl::
reg_matches (buffer, "^[sS] [tT] [aA] [rR] [tT] [ \t]*$")
|| command_received_by_user.command_sent_by_user == START)
{
    output_module->Output("Data acquisition starting...\n");
    miodigitizer.DigitizerStateMachineStartReading ();
}
else if ( command_received_by_user.command_sent_by_user ==
    NEWCOMMAND)
{
    //Nuovo codice
}
else if (DigitizerFlowControl::
reg_matches (buffer, "^[sS] [tT] [oO] [pP] [ \t]*$")
|| command_received_by_user.command_sent_by_user == STOP)
{
    //Il programma termina l'acquisizione dei dati dal digitizer.
    output_module->Output("Stopping data acquisition...\n");
    miodigitizer.DigitizerStateMachineStopReading ();
}
[...]
else if ( command_received_by_user.command_sent_by_user ==
    NEWCOMMAND)
{
    //Codice da aggiungere considerando che in
    //command_received_by_user.first_parameter
    //è presente la stringa 2012-2013-2014-2015
    output_module->Output(for_client_string);
}
[...]
```

Listato 3.12: Codice per aggiungere un nuovo comando al server 2

Per scrivere il codice che deve essere eseguito con il comando *newcommand*, bisogna, innanzitutto, dichiarare il nuovo metodo corrispondente nel file DigitizerStateMachine.h (ad esempio, *void DigitizerStateMachineNewcommand(TcpUser newcommand)*). Poi, occorre aggiungere il codice del metodo nel file DigitizerStateMachine.cpp come si può vedere nel listato 3.13.

```

else if ( command_received_by_user.command_sent_by_user ==
    NEWCOMMAND)
{
    output_module->Output("Hai eseguito il comando newcommand\n");
    miodigitizer.DigitizerStateMachineStopReading ();
}
else if ( command_received_by_user.command_sent_by_user ==
    NEWCOMMAND)
{
    output_module->Output("Hai eseguito il comando newcommand, che
        interrompe l'acquisizione\n");
    miodigitizer. DigitizerStateMachineNewcommandCommand
        (command_received_by_user);
}
[...]
void DigitizerStateMachine::DigitizerStateMachineNewcommandCommand
    (TcpUser newcommand)
{
    OutputModule *output_module;
    output_module = OutputModule::Instance ();
    output_module->Output("Sto eseguendo il metodo newcommand\n");
}
[...]

```

Listato 3.13: Codice per aggiungere un nuovo comando al server 3

Si è giunti alla situazione in cui il client può inviare il comando *newcommand*, il server lo riconosce e lo esegue, che è l'obiettivo che ci si era prefissi.

## 3.6 Il programma di decodifica offline

Una delle idee su cui è fondato il progetto è quella di salvare su disco i dati acquisiti per decodificarli in un secondo momento, ad acquisizione conclusa. Il programma decode permette di decodificare i dati acquisiti.

Attualmente, per via del funzionamento delle funzioni di decoding della libreria CAENDigitizer, per lanciare il programma decode è necessario comunque interfacciarsi al digitizer (vedi sezione 2.5). L'unica eccezione è quella dei

digitizer della famiglia 742: l'azienda ha disposto delle funzioni specifiche per decodificare offline i dati acquisiti con questa famiglia di digitizer. Visto che il sistema è stato sviluppato utilizzando il modello di digitizer v1742, è stato creato il programma decode742 specifico per i digitizer della famiglia 742: utilizzandolo è possibile procedere con la decodifica dei dati acquisiti senza interfacciarsi col digitizer.

Entrambi i programmi sono scritti in C e hanno una struttura molto semplice. L'utente li può eseguire indicando il percorso dei dati da decodificare. Visto che il programma decode deve interfacciarsi col digitizer, è presente un file di configurazione dove è possibile inserire i parametri con cui aprire il digitizer. Per il programma Decode742 questo non è necessario.

Successivamente, viene lanciata la funzione di decodifica *CAEN\_DGTZ\_DecodeEvent* nel caso del programma decode e *X742\_DecodeEvent* nel caso del programma decode742. I dati ottenuti vengono salvati su disco nel percorso indicato dall'utente al lancio del programma.

Anche questi programmi usano la funzione *getopt* per gestire i parametri di lancio.

### **3.7 Organizzazione del codice sorgente del sistema**

Di seguito si riporta il contenuto della cartella principale del sistema:

- MakeFile
- LogFile
- ConfigurationFile
- Manuale d'uso del programma
- Documentazione generata con Doxygen

- Esegibile del lato server
- Esegibile del lato client
- La *directory* SourceCode contenente il codice sorgente del lato server del sistema
- La *directory* SourceCodeClient contenente il codice sorgente del lato client del sistema
- La *directory* RawData contenente il codice sorgente dell'applicazione di decodifica offline
- La *directory* ObjectCode contenente i file oggetto \*.o risultanti dalla compilazione

### 3.8 Installazione del sistema

Per installare il sistema è sufficiente eseguire il Makefile con il comando 'make'.

Per la compilazione sono necessarie le stesse librerie di Wavedump: se nel computer è già installato Wavedump, anche ObjectDump viene compilato senza problemi.

Dopo la compilazione vengono prodotti tre eseguibili: objectdump, objectdumpclient, decode e decode742.

I primi due vengono posti nella cartella principale, il terzo e il quarto nella sottocartella RawData.

### 3.9 Avvio dei programmi

In questa sezione verrà spiegato come avviare il sistema, cioè come avviare il lato server, il lato client e i programmi di decodifica.

### 3.9.1 Avvio del lato server

Il lato server del programma può essere lanciato con il seguente comando:

```
./objectdump [-m user | tcp | all] [-f configurationfilepath] [-d rawdatapath]  
[-l logfilepath]
```

L'opzione -m permette di scegliere la modalità di inserimento dell'input:

-m user: il programma accetta input solo da tastiera

-m tcp: il programma accetta input solo via TCP/IP. In questo caso è lanciato in modalità demone.

-m all: il programma accetta input sia da tastiera sia via TCP/IP.

-m all è l'opzione di default.

L'opzione -f permette di specificare il path del file di configurazione. Di default esso è *ConfigurationFile*, posto nella cartella principale del programma.

L'opzione -d permette di specificare il path del file dove salvare i dati non decodificati. Di default esso è 'data.txt' nella sottocartella RawData.

L'opzione -l permette di specificare il path del file di log. Di default esso è *LogFile*, posto nella cartella principale del programma.

### 3.9.2 Avvio del lato client

Il lato client del programma può essere lanciato con il comando:

```
./objectdumpclient -i serveripaddress
```

L'opzione -i permette di inserire l'indirizzo ip del server. Per testare il programma in locale è sufficiente digitare:

```
./objectdumpclient -i 127.0.0.1
```

### 3.9.3 Avvio del programma di decoding

Il programma decode è presente nella sottocartella RawData. Nella stessa sottocartella è presente il file 'DigitizerConfig.conf'.

Per lanciare decode è necessario impostare il file DigitizerConfig.conf modi-

ficando il parametro OPEN (vedi sezione 3.11) per permettere l'apertura del digitizer.

Il programma può essere eseguito con:

```
./decode -r rawdatopath -s rawdatasize -o decodeeventpath
```

Tutte le opzioni sono necessarie.

L'opzione -r specifica il path del file con i dati non decodificati.

L'opzione -s specifica il path del file con le dimensioni dei dati non decodificati.

N.B. Quando ObjectDump salva su disco i dati non decodificati, crea automaticamente questo file accanto a quello dove sono salvati i rawdata aggiungendo i caratteri 'sz' alla fine di questo.

Esempio: di default ObjectDump salva i dati non decodificati nel file data.txt della sottocartella rawdata e le dimensioni nel file data.txtsz della sottocartella rawdata.

L'opzione -o specifica il path del file contenente i dati decodificati.

Esempio:

```
./decode -r data.txt -s data.txtsz -o events.txt
```

## 3.10 Comandi eseguibili dal sistema

Di seguito si elencano i comandi eseguibili da ObjectDump.

- init: apre il digitizer.
- setup: imposta il digitizer.
- start: inizia la DAQ.
- stop: interrompe la DAQ.
- prestart: inizia il preprocessamento.
- prestop: interrompe il preprocessamento.

- vistart [channelnumber]: visualizza i dati ricevuti nel canale indicato da channelnumber.
- vistop: ferma la visualizzazione.
- rawstart: inizia a salvare su disco i dati acquisiti.
- rawstop: interrompe il salvataggio su disco dei dati ricevuti.
- close: chiude il digitizer.
- send: invia un software trigger.
- help: visualizza la lista dei comandi disponibili.
- check: stampa il contenuto delle impostazioni lette nel file di configurazione e controlla la presenza e la correttezza dei parametri fondamentali.
- chkconf: stampa il contenuto delle impostazioni lette nel file di configurazione.
- write register 0x[register] 0x[data]: scrive nel registro indicato da register i dati indicati da data.
- read register 0x[register]: legge il registro indicato da register.
- -f [conf file path]: imposta il path del file di configurazione.
- -d [data file path]: imposta il path del file dove vengono salvati i dati non decodificati.
- -l [log file path]: cambio il path del logfile.
- print: stampa il contenuto della configurazione interna del programma.
- print files: stampa il path del file di configurazione, del file dove vengono salvati i dati non decodificati e del log file.

- status: stampa lo status dei thread del programma (cioè se sono attivi o spenti).
- more: stampa il contenuto del log file.
- exit/quit: esce dal programma.

Tutti i comandi possono essere inviati anche con objectdumpclient via TCP/IP, con la differenza che exit interrompe objectdumpclient, non objectdump.

## 3.11 Contenuto del file di configurazione del sistema

Il file di configurazione di objectdump ConfigurationFile è posizionato di default nella cartella principale del programma.

L'utente può modificarne il path o lanciando il programma con l'opzione -f configurationfilepath oppure inserendo il comando -f configurationfilepath durante l'esecuzione del programma.

I parametri impostabili sono in larga misura quelli già utilizzati dal software *open source* Wavedump.

### 3.11.1 Impostazioni comuni a tutti i canali

- **OPEN** *usb|pci LinkNumber NodeNumber BaseAddress*

Il parametro open consente di specificare le informazioni necessarie per aprire il digitizer. Se una di queste informazioni non è necessaria (ad esempio il BaseAddress), occorre settarla con il parametro 0.

Esempio: OPEN PCI 0 0 0

- **MAX\_NUM\_EVENTS\_BLT** *maximum\_number\_of\_events*

Il parametro MAX\_NUM\_EVENTS\_BLT imposta il numero di eventi massimo che può essere trasferito in un block transfer.

Esempio: MAX\_NUM\_EVENTS\_BLT 2

- **RECORD\_LENGTH** *number\_of\_samples*

Il parametro RECORD\_LENGTH indica il numero di campioni da acquisire ad ogni trigger.

Esempio: RECORD\_LENGTH 1024

- **POST\_TRIGGER** *value*

Il parametro POST\_TRIGGER indica la dimensione del post-trigger in percentuale della grandezza di record\_length.

Esempio: POST\_TRIGGER 10

- **TEST\_PATTERN** *yes|no*

Il parametro TEST\_PATTERN permette di sostituire alla ADC un'onda triangolare di test con un range da 0 al massimo acquisibile.

Esempio: TEST\_PATTERN yes

- **FPIO\_LEVEL** *ttl|nim*

Il parametro FPIO\_LEVEL indica il tipo dell'input/output dei front panel LEMO connectors.

Esempio: FPIO\_LEVEL nim

- **DECIMATION\_FACTOR** *number\_of\_samples*

Il parametro DECIMATION\_FACTOR, significativo solo per i digitizer della famiglia 740, specifica il decimation factor dell'acquisizione.

Esempio: DECIMATION\_FACTOR 1

- **ENABLED\_FAST\_TRIGGER\_DIGITIZING** *yes|no*

Il parametro ENABLED\_FAST\_TRIGGER\_DIGITIZING, significativo solo per i digitizer della famiglia 742, indica se digitalizzare e rendere disponibili nel readout i segnali acquisiti dai canali di fast triggering.

Esempio: ENABLED\_FAST\_TRIGGER\_DIGITIZING yes

- **FAST\_TRIGGER** *acquisition\_only|disabled*

Il parametro FAST\_TRIGGER permette di usare l'input proveniente dai canali di fast triggering come segnale di trigger per i gruppi 0-1 e 2-3.

Esempio: FAST\_TRIGGER acquisition\_only

- **EXTERNAL\_TRIGGER** *acquisition\_only|acquisition\_and\_trgout|disabled*

Il parametro EXTERNAL\_TRIGGER permette di impostare il modo con cui usare il segnale di trigger.

Esempio: EXTERNAL\_TRIGGER acquisition\_only

- **ENABLE\_DES\_MODE** *yes|no*

Il parametro ENABLE\_DES\_MODE permette di abilitare la Dual Edge Sampling (DES) mode per i digitizer delle famiglie 731 e 751. Quando la DES mode è attiva, solo metà dei canali è abilitata (pari per la famiglia 731, dispari per la famiglia 751).

Esempio: ENABLE\_DES\_MODE yes

- **GNUPLOT\_PATH** *gnuplotcommand|gnuplotprogrampath*

Il parametro GNUPLOT\_PATH indica il comando che ObjectDump utilizzerà per lanciare Gnuplot. Quindi, il parametro deve essere impostato o con il comando utilizzato nella shell per lanciare Gnuplot o con il path assoluto del programma Gnuplot.

Esempio: GNUPLOT\_PATH gnuplot

- **DRS4\_FREQUENCY** *0|1|2*

Il parametro DRS4\_FREQUENCY, significativo solo i digitizer della famiglia 742, permette di impostare la frequenza di campionamento.

0—> 5 Ghz (valore di default)

1—> 2.5 Ghz

2—> 1 Ghz.

Esempio: DRS4.FREQUENCY 1 (cioè viene impostata la frequenza di campionamento a 2.5 Ghz).

- **GROUP\_ENABLE\_MASK** *groupenablemask*

Il parametro GROUP\_ENABLE\_MASK consente di impostare quali gruppi di canali saranno presenti nell'acquisizione. Questo parametro ha senso per le famiglie 740, 742 e 743.

Esempio: GROUP\_ENABLE\_MASK 0x9. In questo caso saranno presenti solo il gruppo 0 e il gruppo 3 (0x9 = 1001 in base 2).

- **CHANNEL\_ENABLE\_MASK** *channeleablemask*

Il parametro CHANNEL\_ENABLE\_MASK consente di impostare quali canali saranno presenti nell'acquisizione. Questo parametro non ha senso per le famiglie 740, 742 e 743.

Esempio: CHANNEL\_ENABLE\_MASK 0x3. In questo caso saranno presenti solo il canale 0 e il canale 2 (0x3 = 11 in base 2).

- **ALL DC\_OFFSET** *dc\_offset*

Il parametro ALL DC\_OFFSET consente di eseguire lo shift dell'input di tutti i canali disponibili della dimensione indicata in dc\_offset. Per avere maggiori informazioni sul significato di tale dimensione, consultare la documentazione tecnica del digitizer.

Esempio: ALL DC\_OFFSET 0x3fff

- **ALL TRIGGER\_THRESHOLD** *triggerthreshold*

Il parametro ALL TRIGGER\_THRESHOLD consente di impostare su tutti i canali disponibili la soglia di self triggering indicata da triggerthreshold.

Esempio: ALL TRIGGER\_THRESHOLD 0x0100

- **SELF\_TRIGGER\_ENABLE\_MASK** *selftriggerenablemask acquisition\_only|acquisition\_and\_trgout|disabled*

Il parametro TRIGGER\_ENABLE\_MASK consente di impostare quali canali generano un segnale di trigger nel caso in cui il loro input superi la TRIGGER\_THRESHOLD impostata.

Esempio (nel caso in cui il digitizer abbia 4 canali): SELF\_TRIGGER\_ENABLE\_MASK 0x9 acquisition\_only. In questo modo i canali che possono generare il trigger sono lo 0 e il 3 ( 0x9 = 1001 in base 2).

### 3.11.2 Impostazioni per singolo canale o gruppo

- **CH channelnumber TRIGGER\_THRESHOLD** *triggerthreshold*

Esempio: CH 2 TRIGGER\_THRESHOLD 0x100

Imposta a 0x100 la soglia di self triggering del canale 2.

- **GR groupnumber TRIGGER\_THRESHOLD** *triggerthreshold*

Esempio: GR 2 TRIGGER\_THRESHOLD 0x100

Imposta a 0x100 la soglia di self triggering del gruppo 2.

- **FAST fastnumber TRIGGER\_THRESHOLD** *triggerthreshold*

Esempio: FAST 1 TRIGGER\_THRESHOLD 0x100

Imposta a 0x100 la soglia di self triggering del canale di fast triggering

1. L'impostazione ha senso solo per i digitizer della famiglia 742.

- **CH channelnumber DC\_OFFSET** *dcoffset*

Esempio: CH 2 DC\_OFFSET 0x3fff

Imposta a 0x3fff il dc offset del canale 2.

- **GR groupnumber DC\_OFFSET** *dcoffset*

Esempio: GR 2 DC\_OFFSET 0x3fff

Imposta a 0x3fff il dc offset del gruppo 2.

- **FAST groupnumber DC\_OFFSET *dcoffset***

Esempio: FAST 1 DC\_OFFSET 0x3fff

Imposta a 0x3fff il dc offset del canale di fast triggering 1. L'impostazione ha senso solo per i digitizer della famiglia 742.

### 3.12 Compilazione di ObjectDump

Il Makefile mette a disposizione i seguenti target:

- all: produce gli eseguibili objectdump, objectdumpclient, decode e decode742. Il codice oggetto prodotto dalla compilazione è posto nella cartella objectcode.
- remove: rimuove gli eseguibili objectdump, objectdumpclient, decode, decode742 e il contenuto della cartella objectcode.
- flex: partendo dal file AnalizzatoreLessicale.flex, produce il file Analizzatore.c.

N.B. Se si modifica il file AnalizzatoreLessicale.flex, occorre eseguire make flex per produrre un nuovo file Analizzatore.c e rendere quindi effettive le modifiche alla successiva compilazione (eseguibile semplicemente con il comando make).

# Capitolo 4

## Test del sistema

Il sistema di controllo e di acquisizione dati ObjectDump è stato utilizzato per la prima volta in condizioni sperimentali nell'agosto del 2015 durante il test di un sistema di rivelatori presso il Super Proton Syncrotron (SPS) del CERN di Ginevra.

Il test è stato eseguito dal gruppo INFN di Catania del WP09 nell'ambito del progetto Eurogammas[1][2]. Il sistema di rivelatori testato era costituito dai cristalli e dai fotomoltiplicatori che compongono il Nuclear Resonance Scattering System (NRSS) progettato per il sistema di diagnostica e calibrazione della facility ELI-NP che verrà costruita a Magurele, Romania. Il rivelatore di NRSS era costituito da 4 scintillatori di Floruro di Bario ( $\text{BaF}_2$ ) delle dimensioni  $5 \times 5 \times 8 \text{ cm}^3$  e da uno scintillatore LYSO delle dimensioni di  $3 \times 3 \times 6 \text{ cm}^3$ .

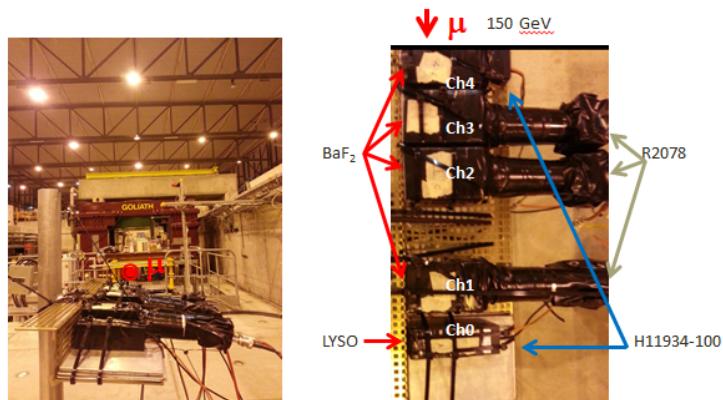


Figura 4.1: Disposizione dei rivelatori

Per poter effettuare la misura della risoluzione energetica dei cinque rivelatori, questi sono stati disposti parallelamente l'uno di fianco all'altro (vedi figura 4.1).

Un fascio di muoni prodotto dall'SPS con un'energia di 150 GeV ha quindi attraversato i cinque cristalli. La luce di scintillazione prodotta nei cristalli è stata raccolta dai fotomoltiplicatori ad essi accoppiati ed i rispettivi segnali inviati al digitalizzatore CAEN modello dt5743. I segnali sono stati registrati con un campionamento di 3.2 GS/s ed il canale 4, sul quale era connesso il fotomoltiplicatore collegato al cristallo di dimensioni più piccole, è stato utilizzato come trigger per l'acquisizione.

Il desktop digitizer dt5743 è stato collegato ad un computer, sul quale era installato ObjectDump, tramite un cavo USB. Un esempio delle forme d'onda ottenute è mostrato in figura 4.2. Nell'utilizzo di ObjectDump si è rivelata utile la possibilità offerta dal sistema di controllare il digitizer scrivendo sui registri di controllo (vedi sezioni 2.1 e 3.10).

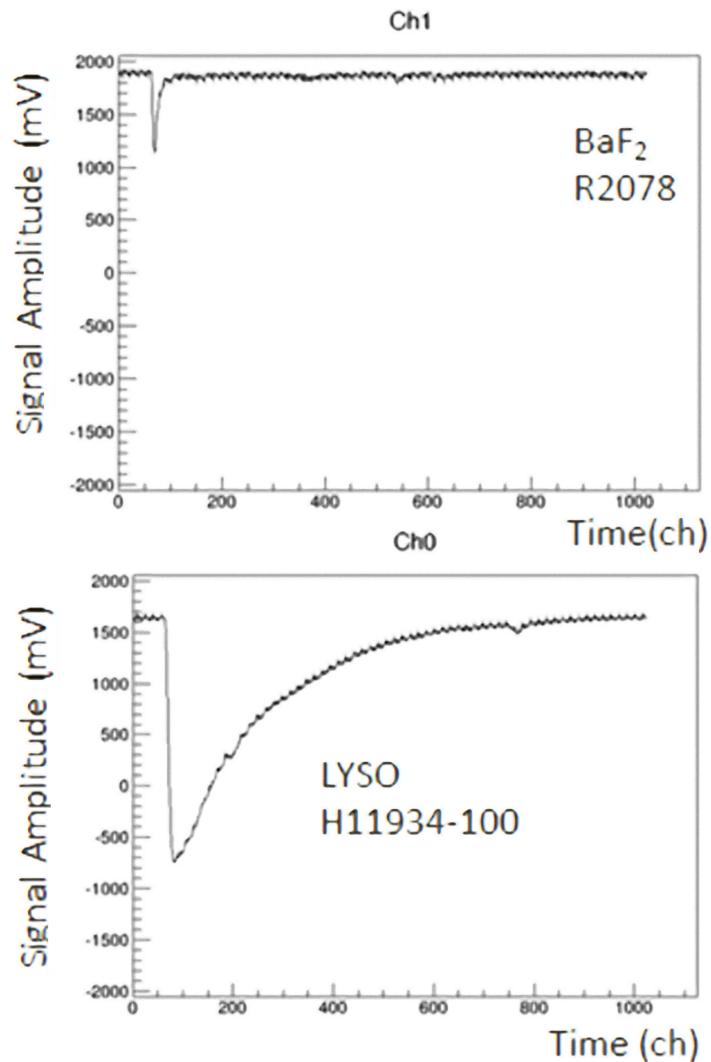


Figura 4.2: Un esempio delle forme d'onda ottenute



# Conclusioni

Considerato che il software *open source* Wavedump fornito dall’azienda CAEN non soddisfava le esigenze del WP09 nell’ambito del progetto Eurogammas, si è proposta una soluzione per sviluppare un sistema che sfrutti al meglio l’architettura dei sistemi operativi Unix-like per incrementarne la velocità di acquisizione. Inoltre, è stata presentata l’idea di separare l’acquisizione dei dati dalla loro decodifica.

Partendo da queste idee è stato sviluppato ObjectDump che migliora le caratteristiche interne ed esterne di Wavedump.

ObjectDump lato server è orientato agli oggetti, quindi è modulare: è possibile modificarlo facilmente e si presta a sviluppi futuri. Ad esempio, per aggiungere un’interfaccia grafica, basta sostituire l’oggetto *DigitizerFlowControl* con uno o più oggetti che ne implementino una. Il resto del codice può rimanere inalterato.

ObjectDump ha un’interfaccia TCP/IP e questo rende il suo utilizzo compatibile con sistemi complessi.

Il sistema è già stato utilizzato sperimentalmente presso il Super Proton Syncrotron (SPS) del CERN di Ginevra dal gruppo INFN di Catania del WP09 nell’ambito del progetto Eurogammas: è stata una prova importante per riscontrarne l’utilità. In futuro, si propende per l’implementazione di un’interfaccia grafica e per la modifica dell’attuale sistema di visualizzazione dei dati in *run time* per renderla simile a quella offerta da un oscilloscopio.

Si conclude affermando che qualsiasi sistema di controllo e acquisizione dati per digitizer CAEN per un sistema operativo Unix-like deve, se l’obiettivo è acquisire ad alte velocità, tenere conto dei risultati presentati in questo elaborato.



# Appendice A

## Risultati dei test di data transfer rate

Per i test riportati di seguito è stato usato un digitizer CAEN v1742 (vedi datasheet B.2) e un disco rigido Seagate ATA ST3320418AS (vedi datasheet B.3). Il digitizer è stato collegato al computer mediante optical link usando una PCI card CAEN A3818[10].

Il digitizer è stato configurato per avere abilitati per il readout tutti i canali tranne i due di fast triggering[6]. Con questa configurazione, ogni evento acquisito ha una dimensione di 49,2 KB. In ogni test sono stati acquisiti 100 eventi (quindi 4920 KB di dati) ed è stato misurato il tempo di esecuzione del codice in nanosecondi con la funzione *clock\_gettime*[19]. In tabella, esso è riportato in millisecondi.

Numero del test	ms	Numero del test	ms
1	2076.658937	51	2062.407468
2	2069.910425	52	2065.570336
3	2070.327249	53	2064.587441
4	2070.585619	54	2063.095083
5	2071.792443	55	2071.513721
6	2072.279071	56	2081.183699
7	2083.394525	57	2066.591381
8	2075.876341	58	2063.168949

9	2069.656169	59	2064.21746
10	2070.507532	60	2064.22509
11	2069.211374	61	2065.131198
12	2067.330696	62	2067.500433
13	2072.49003	63	2074.533544
14	2065.343945	64	2071.718264
15	2079.20402	65	2067.795955
16	2068.377485	66	2068.111038
17	2069.07196	67	2062.960276
18	2068.621465	68	2064.115972
19	2070.736866	69	2071.473452
20	2068.867468	70	2080.222066
21	2071.369487	71	2064.580119
22	2072.773941	72	2063.465957
23	2063.955	73	2070.031431
24	2063.506814	74	2064.851987
25	2065.159733	75	2064.235759
26	2065.828223	76	2065.372593
27	2069.475054	77	2075.938107
28	2087.754249	78	2068.56463
29	2067.596803	79	2088.642428
30	2063.690363	80	2063.587995
31	2067.412469	81	2082.850021
32	2069.96874	82	2066.629367
33	2065.165642	83	2066.968973
34	2068.068751	84	2093.937508

---

35	2086.36548	85	2065.466758
36	2076.035867	86	2063.916324
37	2080.558501	87	2065.694047
38	2083.128198	88	2096.067436
39	2064.753714	89	2068.78097
40	2065.108104	90	2069.464863
41	2067.723661	91	2069.320562
42	2197.265839	92	2063.259717
43	2074.738195	93	2065.005371
44	2072.863096	94	2082.147082
45	2064.602624	95	2086.903735
46	2062.959645	96	2074.52026
47	2065.167181	97	2093.518121
48	2069.2991	98	2133.381591
49	2073.682917	99	2066.044202
50	2067.554299	100	2075.774845

Tabella A.1: Risultati dei test per valutare i tempi di acquisizione con la versione 1 (vedi sezione 2.4.1), il cui codice è analogo a quello usato da Wavedump

Numero del test	ms	Numero del test	ms
1	236.3422	51	235.17698
2	235.037894	52	234.504602

3	234.448916	53	233.638726
4	235.140939	54	238.03833
5	236.937211	55	233.560973
6	235.782948	56	235.368013
7	234.553657	57	233.912026
8	237.317936	58	234.482203
9	235.781347	59	234.258406
10	236.812694	60	234.230506
11	236.254902	61	235.091648
12	248.102061	62	235.082412
13	236.490999	63	234.877161
14	236.289055	64	236.543043
15	234.598311	65	234.107746
16	235.024457	66	238.697846
17	236.826316	67	234.96594
18	237.604644	68	235.502311
19	235.939843	69	237.008566
20	277.107308	70	234.805613
21	235.378814	71	236.58551
22	235.064051	72	244.907498
23	235.543286	73	234.85764
24	234.631032	74	235.910053
25	239.712697	75	235.725703
26	239.828895	76	237.015321
27	235.489502	77	236.025529
28	235.446388	78	235.145127

---

29	236.662339	79	235.476448
30	234.218306	80	234.215587
31	235.559433	81	234.764189
32	235.322564	82	234.126219
33	236.041936	83	234.423802
34	237.220057	84	237.687346
35	234.554715	85	238.81809
36	234.557013	86	235.708214
37	237.737902	87	234.404798
38	235.301897	88	234.25731
39	237.722874	89	235.167313
40	236.891088	90	234.110333
41	249.430565	91	234.29943
42	235.35896	92	239.298102
43	235.420004	93	235.437095
44	233.51063	94	234.971736
45	233.909492	95	236.407756
46	234.18271	96	233.635233
47	347.252977	97	233.600986
48	234.466535	98	234.706326
49	235.20314	99	233.747156
50	235.799042	100	234.287407

Tabella A.2: Risultati dei test per valutare i tempi di acquisizione con la versione 2 (vedi sezione 2.4.2), lettura dei dati e scrittura su disco carattere per carattere

Numero del test	ms	Numero del test	ms
1	57.564951	51	56.024661
2	56.288436	52	55.996881
3	56.223816	53	55.923121
4	55.96999	54	55.903933
5	56.260264	55	56.084546
6	58.066705	56	56.348488
7	56.317178	57	56.32518
8	58.899592	58	56.358893
9	56.300014	59	56.427127
10	56.324414	60	56.276631
11	56.225335	61	56.246477
12	56.273195	62	56.236797
13	56.181028	63	56.322379
14	56.257839	64	56.256998
15	56.253929	65	56.375218
16	56.284385	66	56.324202
17	56.216633	67	56.215382
18	56.222341	68	57.099139
19	56.399252	69	57.708728
20	56.245391	70	57.730655
21	56.289015	71	56.171071
22	56.300806	72	56.815054

---

23	56.233421	73	56.733533
24	56.246279	74	56.188053
25	56.296638	75	56.584398
26	56.225385	76	56.303175
27	56.106476	77	56.189673
28	56.202214	78	56.194572
29	56.243364	79	56.210402
30	56.172418	80	56.271052
31	56.21408	81	56.249917
32	56.225995	82	56.220656
33	56.254766	83	56.352909
34	56.362679	84	55.990263
35	56.248902	85	56.194202
36	56.27427	86	56.110839
37	56.057557	87	55.946425
38	56.054832	88	56.097053
39	55.853539	89	56.713927
40	55.945425	90	55.987387
41	55.989974	91	55.759874
42	55.940743	92	55.719604
43	55.897558	93	56.186704
44	56.066882	94	56.013534
45	56.084427	95	56.03441
46	55.865944	96	56.055654
47	55.869829	97	56.474322
48	56.000777	98	56.044745

49	55.931341	99	56.035887
50	55.91399	100	56.017916

Tabella A.3: Risultati dei test per valutare i tempi di acquisizione con la versione 3 (vedi sezione 2.4.3), lettura dei dati e scrittura su disco bufferizzata

# Appendice B

## Datasheet

<b>Packaging</b>	Desktop module; 154x50x164 mm <sup>3</sup> (WxHxD), Weight: 680 gr
<b>Analog Input</b>	4 channels, single-ended (SE); 2 channel in DES mode Input range: 1Vpp; Bandwidth: 500MHz. Programmable DAC for Offset Adjust x ch.
<b>Digital Conversion</b>	Resolution: 10 bit; Sampling rate: 1GS/s (2GS/s in DUAL EDGE SAMPLING mode) simultaneously on each channel; multi board synchronization (one board can act as clock master). External Gate Clock capability (NIM/TTL) for burst or single sampling mode.
<b>ADC Sampling Clock generation</b>	Three operating modes: - PLL mode - internal reference (50 MHz loc. oscillator). - PLL mode - external reference on CLK_IN ( $\pm 100\text{ppm}$ tolerance). - PLL Bypass mode: Ext. 1GHz clock on CLK_IN drives directly ADC clocks for 1GS/s; 2GS/s in DUAL EDGE SAMPLING mode).
<b>Digital I/O</b>	CLK_IN (AMP Modu II): - AC coupled differential input clock LVDS, ECL, PECL, LVPECL, CML (single ended NIM/TTL available with cable adapter) - Jitter<100ppm TRG_IN (LEMO, NIM/TTL, Zin = 50 Ohm) GPI (LEMO, NIM/TTL, Zin = 50 Ohm) GPO (LEMO, NIM/TTL, across 50 Ohm)
<b>Memory Buffer</b>	1.835 MSamples/ch size (becomes 3.6 MSamples/ch in Dual Edge Sampling mode); Multi Event Buffer with independent read and write access. Programmable event size and pre-post trigger. Divisible into 1 + 1024 buffers.
<b>Trigger</b>	Common Trigger - TRG_IN (External signal) - Software (from USB or Optical Link) - Self trigger (Internal threshold auto-trigger) Daisy chain trigger propagation among boards (using GPO)
<b>Trigger Time Stamp</b>	31-bit counter – 16ns resolution - 17s range.
<b>AMC FPGA</b>	One Altera Cyclone EP3C16 per couple of channels
<b>Multi Modules Synchronization</b>	Allows data alignment and consistency across multiple DT5751 modules: - CLK_IN allows the synchronization to a common clock source - GPI ensures Trigger time stamps and start acquisition times alignment
<b>USB interface</b>	USB2.0 and USB1.1 compliant Up to 30 MB/s transfer rate
<b>Optical Link</b>	CAEN proprietary protocol, up to 80 MB/s transfer rate, with Optical Link Controller (Mod. A2818/A3818).
<b>Upgrade</b>	Firmware can be upgraded via Optical Link or USB interface
<b>Software</b>	General purpose C and LabView Libraries Demo and Software Tools for Windows and Linux
<b>Electrical Power</b>	Voltage range: 12 $\pm$ 10% Vdc Power consumptions (@12Vdc): 1.8 A (Typical)

Figura B.1: Datasheet del digitizer CAEN modello dt5751 tratto dal manuale dell'utente scritto dall'azienda CAEN[7]

<b>Package</b>	1-unit wide VME module
<b>Analog Input</b>	32 channels (MCX 50 Ohm); Single-ended; Input range: 1 Vpp Bandwidth: >500MHz; Programmable DAC for Offset Adjust x ch. adjustment range: ±1V
<b>Sampling frequency</b>	Programmable: 5, 2.5 or 1GS/s
<b>TR0, TR1 Input</b>	MCX 50 Ohm, NIM/TTL; fast local trigger (TR0 for ch0..15, TR1 for ch16..31) and high resolution timing reference
<b>Switched Capacitor array</b>	Based on DRS4 chip Switched capacitor ADC 1024 storage cells per channels simultaneously sampled at 5 - 2.5 - 1GS/s (selectable) on all channels After trigger analog samples are digitized by external ADC.
<b>Digital Resolution</b>	12 bit
<b>Dead Time</b>	110µs Analog inputs only; 181µs Analog inputs + TR0, TR1 inputs
<b>ADC Sampling Clock generation</b>	sampling clock generation supports two operating modes: PLL mode - internal reference (50 MHz local oscillator) PLL mode - external reference on CLK_IN (Jitter<100ppm, Freq. 50 MHz).
<b>Digital I/O</b>	CLK_IN (AMP Modu II): AC coupled differential input clock LVDS, ECL, PECL, LVPECL, CML (single ended NIM/TTL available on request) Jitter<100ppm TRG_IN (LEMO 50 Ohm, NIM/TTL) S_IN (LEMO 50 Ohm, NIM/TTL)
<b>ADC &amp; Memory control FPGA</b>	1 Altera Cyclone EP3C16 for 16+1 channels
<b>Memory Buffer</b>	128 event/ch, (1024 samples per event); Multi Event Buffer with independent read and write access.
<b>Trigger</b>	Common Trigger TRG_IN (External signal) Software (from VME or Optical Link) Fast local trigger Fast local trigger TR0 and TR1 with individual programmable analog threshold
<b>Optical Link</b>	CAEN proprietary protocol, up to 80 MB/s transfer rate, Daisy chainable: it is possible to connect up to 8/32 ADC modules to a single Optical Link Controller (Mod. A2818/A3818)
<b>Multi Modules Synchronization</b>	Allows data alignment and consistency across multiple V1742 modules: CLK_IN allows the synchronization to a common clock source S_IN ensures start acquisition times alignment
<b>Upgrade</b>	Firmware can be upgraded via VME/Optical Link
<b>VME interface</b>	VME64X compliant D32, BLT32, MBLT64, CBLT32/64, 2eVME, 2eSST, Multi Cast CyclesTransfer rate: 60MB/s (MBLT64), 100MB/s (2eVME), 160MB/s (2eSST). Sequential and random access to the data of the Multi Event Buffer. The Chained readout allows to read one event from all the boards in a VME crate with a BLT access.
<b>Software</b>	Libraries (C and LabView), Demos and Software tools for Windows and Linux

Figura B.2: Datasheet del digitizer CAEN modello v1742 tratto dal manuale dell'utente scritto dall'azienda CAEN[6]

---

Drive specification	ST3500418AS and ST3500410AS	ST3320418AS
Formatted capacity (512 bytes/sector)*	500 Gbytes	320 Gbytes
Guaranteed sectors	976,773,168	625,142,448
Heads	2	
Discs	1	
Bytes per sector	512	
Default sectors per track	63	
Default read/write heads	16	
Default cylinders	16,383	
Recording density	1413 kbytes/in max	
Track density	236 ktracks/in avg	
Areal density	329 Gbytes/in <sup>2</sup> avg	
Spindle speed	7,200 RPM	
Internal data transfer rate	1695 Mbytes/sec max	
Sustained data transfer rate OD	125 Mbytes/sec max	
I/O data-transfer rate	300 Mbytes/sec max	
ATA data-transfer modes supported	PIO modes 0–4 Multivord DMA modes 0–2 Ultra DMA modes 0–6	
Cache buffer	16 Mbytes	
Height (max)	19.98 mm (0.787 inches)	
Width (max)	101.6 mm (4.000 inches) +/- 0.010 inches	
Length (max)	146.99 mm (5.787 inches)	
Weight (typical)	415 grams (0.915 lb.)	
Average latency	4.16 msec	
Power-on to ready	<8.5 sec max	
Standby to ready	<8.5 sec max	
Track-to-track seek time	<1.0 msec typical read; <1.2 msec typical write	
Average seek, read	<8.5 msec typical	
Average seek, write	<9.5 msec typical	
Startup current (typical) 12V (peak)	2.0 amps	
Voltage tolerance (including noise)	5V +10% / -7.5% 12V +10% / -7.5%	
Ambient temperature	0° to 60°C (operating) -40° to 70°C (nonoperating)	
Temperature gradient	20°C per hour max (operating) 30°C per hour max (nonoperating)	
Relative humidity	5% to 95% (operating) 5% to 95% (nonoperating)	
Relative humidity gradient	30% per hour max	
Wet bulb temperature	37.7°C max (operating) 40.0°C max (nonoperating)	
Altitude, operating	-304.8 m to 3,048 m (-1000 ft. to 10,000+ ft.)	
Altitude, nonoperating (below mean sea level, max)	-304.8 m to 12,192 m (-1000 ft. to 40,000+ ft.)	
Operational Shock	70 Gs max at 2 msec	
Non-Operational Shock	350 Gs max at 2 msec	

Figura B.3: Datasheet dell'hard disk Seagate ATA ST3320418AS tratto dal manuale offerto dall'azienda Seagate[5]



# Bibliografia

- [1] <http://www.e-gammas.com>
- [2] <http://www.elи-nр.ro>
- [3] [https://it.wikipedia.org/wiki/Daisy\\_chain\\_%28informatica%29](https://it.wikipedia.org/wiki/Daisy_chain_%28informatica%29)
- [4] <http://www.caen.it/jsp/Template2/Function.jsp?parent=62&idfun=102>
- [5] <http://www.seagate.com/staticfiles/support/disc/manuals/desktop/Barracuda%207200.12/100529369c.pdf>
- [6] <http://www.caen.it/csite/CaenProd.jsp?parent=11&idmod=661>
- [7] <http://www.caen.it/jsp/Template2/CaenProd.jsp?parent=62&idmod=879>
- [8] <http://www.caen.it/csite/Product.jsp?parent=38&Type=Product>
- [9] <http://www.caen.it/csite/CaenProd.jsp?parent=15&idmod=450>
- [10] <http://www.caen.it/csite/CaenProd.jsp?parent=15&idmod=627>
- [11] <http://www.caen.it/csite/CaenProd.jsp?parent=15&idmod=450>
- [12] <http://www.caen.it/csite/CaenProd.jsp?parent=15&idmod=627>
- [13] <http://www.caen.it/csite/Function.jsp?parent=38&idfun=99>
- [14] <http://www.caen.it/csite/CaenProd.jsp?parent=38&idmod=692>
- [15] <http://www.caen.it/csite/CaenProd.jsp?parent=38&idmod=717>

- [16] Advanced programming in the UNIX Environment di W. Richard Stevens e Stephen A. Rago, Second Edition (Addison-Wesley Professional Computing Series).
- [17] <http://man7.org/linux/man-pages/man3/getopt.3.html>
- [18] Design Patterns: Elements of Reusable Object-Oriented Software di E. Gamma, R. Helm, R. Johnson e J. Vlissides, Addison Wesley, 1995.
- [19] [http://linux.die.net/man/3/clock\\_gettime](http://linux.die.net/man/3/clock_gettime)

# Elenco delle figure

1.1	Livelli hardware e software dei prodotti CAEN[13] . . . . .	4
2.1	Struttura di uno stream di dati acquisito da un digitizer di tipo v1742[6] . . . . .	16
2.2	Contenuto del campo channel data[6] . . . . .	16
2.3	Risultati dei test di acquisizione con la versione 1 (vedi sezione 2.4.1), il cui codice è analogo a quello usato da Wavedump . .	19
2.4	Risultati dei test di acquisizione con la versione 2 (vedi sezione 2.4.2), lettura dei dati e scrittura su disco carattere per carattere	20
2.5	Risultati dei test di acquisizione con la versione 3 (vedi sezione 2.4.3), lettura dei dati e scrittura su disco bufferizzata. . . . .	21
2.6	Paragone tra i tempi di acquisizione delle tre versioni . . . . .	22
3.1	Diagramma di classe del lato server del sistema . . . . .	32
4.1	Disposizione dei rivelatori . . . . .	65
4.2	Un esempio delle forme d'onda ottenute . . . . .	67
B.1	Datasheet del digitizer CAEN modello dt5751 tratto dal manuale dell'utente scritto dall'azienda CAEN[7] . . . . .	79
B.2	Datasheet del digitizer CAEN modello v1742 tratto dal manuale dell'utente scritto dall'azienda CAEN[6] . . . . .	80
B.3	Datasheet dell'hard disk Seagate ATA ST3320418AS tratto dal manuale offerto dall'azienda Seagate[5] . . . . .	81



# Elenco delle tabelle

A.1 Risultati dei test per valutare i tempi di acquisizione con la versione 1 (vedi sezione 2.4.1), il cui codice è analogo a quello usato da Wavedump . . . . .	73
A.2 Risultati dei test per valutare i tempi di acquisizione con la versione 2 (vedi sezione 2.4.2), lettura dei dati e scrittura su disco carattere per carattere . . . . .	75
A.3 Risultati dei test per valutare i tempi di acquisizione con la versione 3 (vedi sezione 2.4.3), lettura dei dati e scrittura su disco bufferizzata . . . . .	78



# Elenco dei listati

2.1	Esempio di un semplice programma C di readout . . . . .	12
2.2	Codice C per il trasferimento e la decodifica utilizzato da Wavedump . . . . .	17
2.3	Pseudocodice per il trasferimento ottimizzato . . . . .	17
2.4	Codice C per il trasferimento e la decodifica utilizzato da Wavedump . . . . .	18
2.5	Codice C per il trasferimento e la scrittura dei dati su disco carattere per carattere . . . . .	20
2.6	Codice C per il trasferimento e la scrittura dei dati su disco usando fwrite . . . . .	21
2.7	Codice C per trasferimento e scrittura dei dati su disco usando fwrite, salvando anche la dimensione del burst scritto . . . . .	23
2.8	Strutture dati per salvare i dati decodificati forniti dalle libbre- ria CAENDigitizer per i modelli della famiglia 742 . . . . .	24
3.1	Codice eseguito dal thread produttore dell'oggetto Digitizer- StateMachine . . . . .	36
3.2	Esempio manualistico dell'utilizzo della funzione getopt()[17] .	38
3.3	Codice per ottenere un'istanza dell'oggetto OutpuModule . .	40
3.4	Utilizzo della funzione getopt() nel main dell'applicazione client	43
3.5	Codice della funzione main dell'applicazione client 1 . . . . .	44
3.6	Codice della funzione main dell'applicazione client 2 . . . . .	46
3.7	Codice della funzione ricevitore dell'applicazione client . . . . .	47
3.8	Codice della funzione command_parser dell'applicazione client	48
3.9	Codice per aggiungere un nuovo comando all'applicazione client 1 . . . . .	49

3.10 Codice per aggiungere un nuovo comando all'applicazione client 2 . . . . .	50
3.11 Codice per aggiungere un nuovo comando al server 1 . . . . .	50
3.12 Codice per aggiungere un nuovo comando al server 2 . . . . .	52
3.13 Codice per aggiungere un nuovo comando al server 3 . . . . .	52

# Ringraziamenti

Ringrazio Mirco Andreotti per avermi accolto nel suo laboratorio dove ho imparato molto. Ringrazio Bruno Zerbo e Michela Lenzi per il contributo fornитomi nella fase di testing del sistema ObjectDump e Maria Grazia Pellegriti per avermi fornito le informazioni relative al test di Agosto 2015. Ringrazio Paolo Cardarelli per l'aiuto fornитomi in fase di documentazione. Ringrazio il Dipartimento di Fisica e Scienze della Terra dell'Università degli Studi di Ferrara e la sezione di Ferrara dell'Infn per gli strumenti messi a disposizione per la costruzione del sistema.

Ringrazio Michele Gambetti per la disponibilità e la fiducia che ha sempre dimostrato nei miei confronti. Ringrazio Alberto Gianoli per le lezioni di Ingegneria del Software di spessore. Ringrazio Fabio Sebastiano Schifano per il grande impegno speso per il corso di Sistemi Operativi. Ringrazio il Professor Gaetano Zanghirati per l'umanità del suo insegnamento. Ringrazio Luca Tomassetti per avermi introdotto al mondo dell'informatica con il primo esame di Programmazione.

Infine, ringrazio particolarmente il Professor Carlo Morini per le illuminanti lezioni di Matematica e di Logica che ho seguito con grandissimo interesse.

