ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

SVILUPPO DI UN MODULO DI VISUALIZZAZIONE, MEMORIZZAZIONE E ANALISI DATI IN REAL-TIME CON ELEVATO PARALLELISMO

Relazione finale in
PROGRAMMAZIONE AD OGGETTI

Relatore
Prof. MIRKO VIROLI

Presentata da
BAJRAM HUSHI

Seconda Sessione di Laurea
Anno Accademico 2014 – 2015

PAROLE CHIAVE

Elements

Elettrofisiologia

Acquisizione Real Time

Analisi in Real Time

High Throughput Data

Indice

Introduzione

Lo scopo dell'elaborato di tesi è lo sviluppo dei moduli di visualizzazione, memorizzazione e analisi di dati acquisiti in real-time da dispositivi per elettrofisiologia sviluppati dall'azienda Elements s.r.l. Elements produce amplificatori miniaturizzati, mantenendo alte frequenze e accuratezza, in grado di misurare correnti a bassa intensità prodotte dai canali ionici.

Data la continua evoluzione dei dispositivi prodotti dall'azienda, che introducono funzionalità e precisioni sempre migliori, l'azienda ha espresso la necessità di un sistema software che si interfacci al meglio con i dispositivi e presenti funzionalità all'avanguardia di analisi dei dati acquisiti in real-time.

Il software deve fornire un'interfaccia utente che permetta la configurazione del dispositivo connesso, acquisisca dati da esso ed effettui operazioni di visualizzazione, memorizzazione e analisi dei dati acquisiti. In questa tesi verranno esposte analisi, progettazione e implementazione dei moduli di visualizzazione, memorizzazione e alcuni tipi di analisi che il sistema sarà in grado di effettuare.

Lo scopo del progetto di tesi non è solo lo sviluppo del software in sé ma gestire grandi throughput di dati, visto che i dispositivi effettuano letture fino a 200 KHz a 16 bit di precisione e con elevato parallelismo, mantenendo allo stesso tempo un interfaccia fluida e un utilizzo basso delle risorse del computer (CPU, RAM, ecc..). Le specifiche fornite inizialmente erano un sistema che sia in grado di gestire un amplificatore a 4 canali con frequenza di campionamento a 200 KHz, con un throughput pari a 30 Mbit/s, con la richiesta di supportare throughput fino a 16 canali.

Un'altra caratteristica necessaria è quella di mantenere un ampliamento flessibile e alta scalabilità sia in ambito di nuove funzionalità, sia in previsione di nuovi dispositivi, minimizzando le modifiche necessarie al codice sorgente. E' stato quindi adottato uno sviluppo modulare dell'applicazione con la minima connessione possibile tra loro, in modo tale che le modifiche in un determinato

modulo non influenzino gli altri, e che l'aggiunta di nuovi moduli non influenzi in nessun modo i moduli prodotti in precedenza.

Limitare la latenza, limitare la perdita di dati, mantenere prestazioni elevate ed effettuare un uso efficiente delle risorse sono alcune delle specifiche richieste, inoltre il software deve essere portabile su diversi sistemi operativi, prevedere diversi dispositivi hardware con cui interfacciarsi e presentare un' interfaccia utente semplice e reattiva. Per rispettare tutte le specifiche è stato utilizzato il framework *Qt*, che è totalmente compatibile con tutti i sistemi operativi, il linguaggio di programmazione *c++*, che effettua operazioni in modo più efficiente di linguaggi più ad alto livello, l'uso della libreria *Qwt* per la rappresentazione di grafici e il driver FTDI per l'interfacciamento dei dispositivi che utilizzano questa tecnologia per la comunicazione via USB.

Per gestire al meglio i problemi prestazionali è stata utilizzata la tecnologia multi-thread, presente in tutti i sistemi operativi. Questa ha permesso una gestione parallela delle diverse funzionalità, garantendo prestazioni elevate. La comunicazione tra diversi thread è stata gestita mediante le funzionalità delle *signal* e delle *slot* presenti nel framework *Qt*. Non fornendo però buone prestazioni, nei segmenti più importanti sono state utilizzate strutture dati apposite per mantenere alte prestazioni e gestiti opportunamente con la mutua esclusione nella scrittura e lettura dei dati. Un altro elemento chiave per mantenere alte prestazioni sono state le strutture di base del linguaggio c++, come le struct e i puntatori, che permetto di effettuare operazioni in modo più efficiente rispetto ad altri linguaggi di alto livello.

Il sistema software sviluppato soddisfa tutti i requisiti concordati con l'azienda inizialmente, gestendo senza problemi i dispositivi finora prodotti. È stato inoltre provato, che il programma è in grado di gestire anche i dispositivi in fase di prototipazione a 16 canali e una bit-rate stimata a 103 Mbit/s. Il flusso di dati viene visualizzato, memorizzato e analizzato alla bitrate indicata, senza perdita di dati e mantenendo l'interfaccia grafica responsiva e fluida.

Per esprimere al meglio i requisiti nel paradigma di programmazione a oggetti, è stata necessaria una prima fase di analisi per la modellazione dei vari moduli e l'interfacciamento tra loro, una seconda fase di progettazione per la definizione dei vari algoritmi e strutture dati da utilizzare per poter proseguire con l'implementazione di un sistema software che mantenga un utilizzo intelligente delle risorse.

Il progetto è stato sviluppato in team con il collega Matteo Marra, suddividendo equamente i vari moduli. Questo elaborato di tesi tratterà quindi lo sviluppo del modulo di Visualizzazione, Memorizzazione e Analisi, mentre il mio collega si è occupato di realizzare i moduli di Connessione, Acquisizione e Rilevazione.

La tesi è suddivisa in sette capitoli:

Nel primo capitolo verrà fatta una breve descrizione dell'azienda e dei dispositivi da loro prodotti.

Il secondo capitolo introduce le tecnologie utilizzate ai fini dello sviluppo ottimale del progetto.

Il terzo capitolo espone tutti i requisiti necessari, espressi dall'azienda, che il sistema deve mantenere.

Nel quarto capitolo verrà fatta un'analisi approfondita dei vari moduli, presentando vari diagrammi quali i casi d'uso e il diagramma delle classi, espressi via UML.

Il quinto capitolo mostra tutte le scelte progettuali, gli algoritmi utilizzati e la struttura dell'interfaccia utente.

Il sesto capitolo mostra scelte implementative fatte basandosi sulle scelte progettuali.

Infine l'ultimo capitolo mostra tutte le varie fasi di testing effettuate per una valutazione prestazionale e funzionale dell'applicazione.

Segue inoltre una breve conclusione.

Capitolo 1

Elements s.r.l.

1.1 Chi è Elements

Elements s.r.l. è una start-up nel campo della microelettronica, che progetta e sviluppa strumentazione di misura miniaturizzata e a elevata accuratezza. I numerosi anni di ricerca nel campo della microelettronica applicata a bio e nanosensori bio-sensibili dà al team di Elements un vantaggio unico nell'analisi di segnali a bassa ampiezza. Elements può inoltre vantare diverse collaborazioni con molti centri di ricerca universitari e aziende sia europee che statiunitensi. [2]

1.2 Descrizione dispositivi

I prodotti elements, distribuiti già a livello globale nel settore dell'elettrofisiologia, sono amplificatori miniaturizzati in grado di leggere e analizzare correnti ioniche di bassissima intensità. Le correnti ioniche sono segnali elettrici prodotti dai canali ionici, ossia proteine presenti sulla membrana cellulare che assumono la forma di pori nanometrici. I canali ionici sono coinvolti in tutti i processi vitali di funzionamento delle cellule del corpo umano e sono indispensabili per analizzare l'azione dei farmaci a livello cellulare.

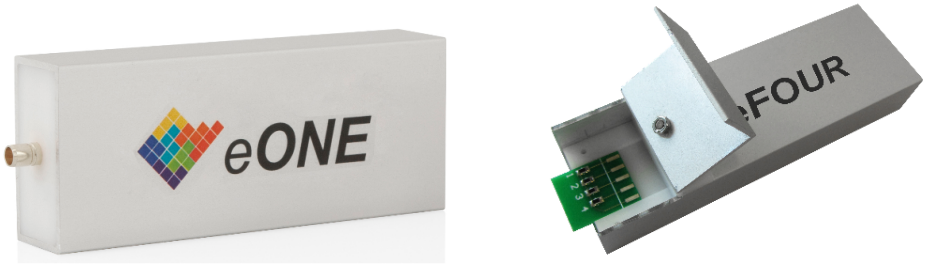
Si tratta di dispositivi molto piccoli, con una dimensione di 30x15x74mm, che al loro interno racchiudono un amplificatore a basso rumore, un digitalizzatore e dei filtri. Contengono inoltre un generatore di stimoli di tensione e il tutto è completamente alimentato da un unico cavo USB.

Tale strumentazione di misura trova applicazione in tutti i settori riguardanti la misura di segnali provenienti da diverse famiglie di nano-sensori (sensori biologici o artificiali delle dimensioni nanometriche), in particolare nel-

l'elettrofisiologia, uno dei rami della fisiologia che studia il funzionamento dell'organismo e delle cellule umane dal punto di vista elettrico.

eONE ed eFOUR

Gli amplificatori più piccoli al mondo di nanopori e applicazione elettrofisiologica. I dispositivi, alimentati via USB, sono sistemi completi di acquisizione dati. Attualmente sono stati sviluppate due versioni di dispositivo, eONE ed eFOUR, che a loro volta presentano delle sotto-versioni con diverse funzionalità. L'eONE acquisisce dati da un singolo canale, mentre l'eFOUR acquisisce dati parallelamente da quattro canali.



(a) Singolo canale (b) Quattro canali

Figura 1.1: Dispositivi attualmente in produzione

eSIXTEEN

Nuovo dispositivo ancora in fase prototipale dove sarà possibile acquisire dati da 16 canali simultaneamente con frequenze di campionamento fino a 200 kHz, mantenendo le ridotte dimensioni e la struttura dei dispositivi eONE ed eFOUR citati in precedenza.

1.3 Caratteristiche dei dispositivi

Si tratta di dispositivi di dimensioni molto ridotte (30x15x74mm), ma nonostante ciò consente numerosi tipi di analisi ed esperimenti, fornendo un'alta risoluzione e diverse modalità di lavoro.

Le funzionalità principali offerte sono :

- Frequenza selezionabile, da un minimo di 1.25 kHz fino ad un massimo di 200 kHz
- Range selezionabile, scegliendo tra 20 nA e 200 pA

- Risoluzione ADC a 14 bit
- Range degli stimoli di tensione di ± 380 mV

1.3.1 Protocolli di tensione

I dispositivi possono utilizzare diversi tipi di protocolli di tensione, che verranno settati e configurati attraverso il software che è stato sviluppato. La maggior parte degli esperimenti è soddisfatta grazie alla presenza di questi protocolli, che vengono, caso per caso, scelti e configurati dall'elettrofisiologo che sta operando l'analisi.

1.3.2 Connessione al PC

Per poter funzionare i dispositivi devono essere collegati ad un PC sia per l'alimentazione sia per l'interfacciamento col software che ne permette la lettura e l'analisi dei dati acquisiti. La connessione via USB permette la portabilità dei dispositivi verso qualsiasi PC abbia un'interfaccia USB 2.0 e il software di interfacciamento installato (Elements Data Reader).

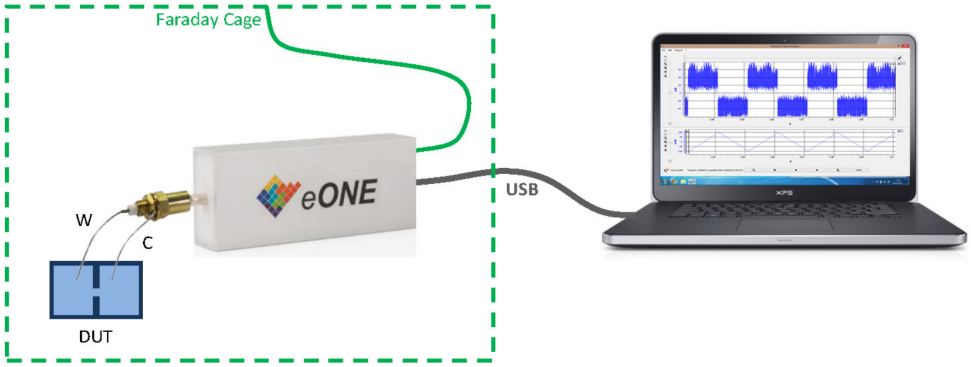


Figura 1.2: Connessione eONE al PC

1.4 Applicazioni in Elettrofisiologia

I dispositivi trattati sono utilizzati per lo studio e l'analisi dei canali ionici, settori di analisi riguardanti l'elettrofisiologia. L'elettrofisiologia è una branca della fisiologia che ha come studio il funzionamento dell'organismo dal punto di vista elettrico, sia in condizioni fisiologiche normali sia sotto l'influsso di un potenziale elettrico esterno.

1.4.1 I canali ionici [1]

I canali ionici sono proteine presenti nella membrana che circonda tutte le cellule biologiche e organuli intracellulari (nuclei, mitocondri, ecc..). Sono implicati in molte varietà di processi fisiologici che includono rapidi cambiamenti nelle cellule come le cellule cardiache, scheletriche, e nelle contrazioni muscolari. Regolano il passaggio di sostanze attraverso le membrane cellulari per via di un trasporto selettivo degli ioni. Nello specifico un canale ionico è una proteina trans-membrana che permette il passaggio di determinati ioni dall'esterno all'interno della cellula e viceversa in modo selettive per una o pochi tipi di ioni. Questo transito di ioni, se connesso ad appropriati elettrodi, può essere convertito in corrente elettrica, che viene quindi acquisita dai dispositivi sopracitati.

I ruoli critici che i canali ionici assumono li hanno resi molto importanti nell'analisi di medicinali, poiché la modulazione farmacologica dei canali ionici consente sottili correzioni delle loro disfunzioni, implicandoli in molti disturbi e malattie soprattutto nel sistema nervoso, gastrointestinale e cardiovascolare.

Capitolo 2

Tecnologie utilizzate

Descrizione delle tecnologie hardware e software utilizzate per lo sviluppo del progetto.

2.1 USB

L'interfaccia di comunicazione hardware tra il dispositivo e il computer è l'USB, che viene utilizzato anche per l'alimentazione stessa del dispositivo. USB (Universal Serial Bus) è uno standard di comunicazione seriale tra dispositivi di diversa struttura, solitamente usata per la connessione di periferiche al computer.

2.1.1 Descrizione USB

Storia [3]

Nella sua fase iniziale, USB è stato sviluppato in modo congiunto da sette compagnie: Compaq, DEC, IBM, Intel, Microsoft, NEC e Nortel. L'obbiettivo dichiarato della ricerca era di semplificare in modo sostanziale il metodo di collegamento dei dispositivi al PC; in questo senso USB è andato, nel corso degli anni, a sostituire molti altri tipi di collegamento.
L'esigenza di questo protocollo si è mostrata nei primi anni '90, dove esistevano un gran numero di protocolli e connettori seriali che però non erano compatibili tra di loro, spesso non plug'n play e non erano in grado di gestire contemporaneamente più dispositivi connessi.

Figura 2.1: Cavi di comunicazione vs USB

Evoluzione del protocollo

Le specifiche USB 1.0 sono state introdotte per la prima volta nel corso del mese di Gennaio del 1996: la prima ad essere usata dal grande pubblico è stata però la versione 1.1, rilasciata solo nel mese di Settembre del 1998. La prima azienda ad aver sviluppato un chip funzionante con lo standard USB è stata Intel, nel 1995. Nella sua fase iniziale, USB era capace di garantire una velocità di trasferimento dati pari a 12 Mbit/s per i dispositivi di memorizzazione dati e di 1,5 Mbit/s per periferiche di input.

Solo due anni dopo, nell'Aprile del 2000, fanno capolino le specifiche USB 2.0, ratificate dall'USB-IF per la fine del 2001. A lavorare, anche in questo caso in modo congiunto, dietro alle rinnovate specifiche sono state Hewlett-Packard, Intel, Lucent Technologies, NEC e Philips. USB 2.0 ha consentito agli utenti di poter godere di una velocità di trasferimento dati fino a 480Mbit/s, un incremento sostanziale se paragonato ai 12Mbit/s di cui era capace USB 2.0.

Il 17 di Novembre del 2008 è stata la volta dell'apparizione delle specifiche 3.0, sempre ad opera dell'implementers forum USB, aprendo così il campo alle aziende hardware che decidessero di cominciare a sviluppare soluzioni basate sulla nuova connessione. Le prime proposte capaci di supportare USB 3.0 sono state presentate nel CES di Gennaio del 2010 e solo nei mesi a seguire hanno cominciato a diffondersi, timidamente, sul mercato. Anche in questo caso, come avvenuto nella precedente transizione, assistiamo ad un importante incremento della velocità di trasferimento dati: USB 3.0 è infatti capace di garantire una banda pari a 4,8GBit/s (600MB/s).

2.1.2 Funzionamento USB 2.0

Siccome i dispositivi progettati da Elements utilizzano la versione 2.0 dell'USB verrà fatta una breve panoramica di esso.
La comunicazione USB si basa su tre componenti fondamentali:

- Computer Host
- Device USB
- Bus Fisico, cavo fisico, che collega il device al computer

L'USB segue un architettura Master-Slave. Tutti i dispositivi USB hanno il ruolo di Slave e il computer host quello di Master. Il Master inizializza tutti i trasferimento dati attraverso uno scheduler.
Siccome il master controlla il bus, due dispositivi connessi allo stesso host non possono comunicare direttamente tra loro.

Comunicazione

Per la comunicazione viene utilizzato un protocollo half-duplex asincrono, con il segnale differenziale che viaggia su un doppino intrecciato.
Esistono tre tipi di trasferimento:

- **Isocrono:** viene garantita una bitrate fissa ma senza controllo d'errore. Usata specialmente per applicazione che non possono permettersi latenze, come microfoni e webcam, ma si possono permettere una minima perdita di informazione.
- **Bulk:** utilizzate per il trasferimento di grandi moli di dati, con latenze variabili ma con controllo di errore perché deve essere garantita l'affidabilità dei dati.
- **Interrupt:** utilizzata da dispositivi che hanno priorità più alte rispetto ad altre, es mouse, tastiera. Utilizza il meccanismo di interruzione hardware per una risposta in real-time.

2.2 Framework QT

Per mantenere alcune qualità del software è stato scelte come framework di sviluppo **QT**, integrante con il linguaggio di programmazione *C++*.

QT è un framework multi-piattaforma che consente lo sviluppo di applicazioni che possono essere eseguite su diversi sistemi col minimo cambiamento di codice sorgente, mantenendo però la potenza e la velocità delle applicazioni native.

Architettura Software [4]

Il framework QT si basa su tre concetti fondamentali:

- **Astrazione della GUI** - Nella prima versione QT usa un proprio motore grafico emulando i componenti dei diversi sistemi in cui va in esecuzione. Nelle recenti versioni di QT vengono usati API nativi del sistema perché l'utilizzo dell'emulatore era imperfetto quanto i componenti emulati non erano totalmente indipendenti dal sistema. QT è un'API nativa di alcune piattaforme come MeeGo e KDE.

- **Signals e Slots** - sono un costrutto del linguaggio introdotti da QT per la comunicazione tra oggetti che permette l'implementazione semplificata del pattern *observer*.

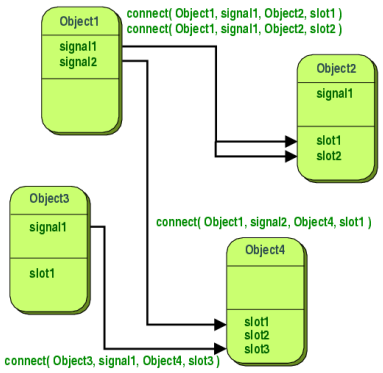


Figura 2.2: Esempio utilizzo Signal e Slot

- **Compilatore di *Meta-Oggetti*** - Il compilatore dei meta-oggetti è un tool che viene eseguito nel sorgente di un programma Qt. Interpreta le macro dal codice C++ come annotazioni e li usa per generare codice C++ con le meta informazioni delle classi usate nel programma. Le meta informazioni sono usati da Qt per aggiungere funzionalità che non sono presenti nativamente nel C++, come le signal e le slot.

L'ambiente Qt Creator

Qt creator è l'IDE di sviluppo utilizzato per l'implementazione del software. L'IDE contiene delle funzionalità che aiutano i nuovi utenti Qt ad essere più veloci e produttivi, aumentando l'esperienza. Include un debugger visuale per un correzione di bug più efficiente.

Libreria Qwt [5]

Qwt, o *Qt Widget for Technical Applications*, è un set di Qt widget, componenti e classi di utilità per programmi in ambiti tecnici. Fornisce un framework per la visualizzazione 2D di grafici, con scale, slider, e molte altre componenti che permettono di controllare e visualizzare valori, array e intervalli di tipo double. La libreria Qwt è stata utilizzata per la visualizzazione dei dati in real-time e anche per mostrare i risultati delle analisi in real-time dei dati acquisiti.

2.3 FTDI

In tutti i dispositivi descritti precedentemente è presente un chip di comunicazione USB prodotto dall'azienda FTDI, Future Technlogy Devices International. L'utilizzo di questo chip aumenta la facilità d'uso e la portabilità dei dispositivi, poichè non è necessario implementare il protocollo di comunicazione USB. L'utilizzo delle librerie FTDI permetterà l'interazione con i dispositivi mediante il driver FTDI corretto.

2.4 EEPROM

EEPROM è un tipo di memoria non volatile, usata nei computer e altri dispositivi elettronici per memorizzare piccole quantità di dati che devono essere mantenuti quando viene tolta l'alimentazione elettrica. Le operazioni di scrittura, cancellazione e riscrittura hanno luogo elettricamente. Tutti i dispositivi descritti in precedenza contengono una EEPROM per la configurazione di ogni singolo dispositivo con valori di correzione e offset.

2.5 Repository

Siccome il software è stato sviluppato da due persone è stato necessario l'utilizzo di un repository mercurial salvati online da un servizio di hosting

chiamato bitbucket. L'utilizzo della repository ha permesso l'allineamento semplificato delle diverse versioni del codice.

BitBucket

Bitbucket è un servizio di hosting web-based, dove è possibile mantenere diversi repository sia pubbliche, utilizzati per software open-source, che private dove solo elementi di un team ristretto possono interagire.

Capitolo 3

Analisi Dei Requisiti

Descrizione dei requisiti richiesti dal sistema.

3.1 Descrizione del Sistema

3.1.1 Prospettiva del Sistema

Il sistema software progettato e sviluppato è un sistema di gestione dati in real-time ad elevato throughput di dati, acquisiti dai dispositivi prodotti da Elements s.r.l. L'azienda ha richiesto l'aggiornamento del sistema software per la risoluzione di problematiche di performance e scalabilità rispetto al software attualmente in uso.

3.1.2 Funzionalità Principali

Le funzionalità principali che il software deve rispettare sono:

- Riconoscimento Dispositivi
- Acquisizione Dati
- Elaborazione Dati
- Visualizzazione Dati in real-time
- Memorizzazione su disco in diversi formati
- Analisi dei Dati in real-time
- Portabilità su diverse piattaforme (Windows, OS Mac...)

La suddivisione delle funzionalità in questo modo permette una progettazione modulare del sistema e una maggiore scalabilità. Le varie funzionalità verranno poi descritte dettagliatamente nei capitoli successivi.

3.1.3 Requisiti richiesti dall'azienda

Progettare e sviluppare un sistema che consenta la lettura in real-time di dati prodotti dai dispositivi prodotti dall'azienda stessa. Il sistema deve visualizzare, memorizzare su disco ed effettuare analisi dei dati letti, mantenendo alte prestazioni ed evitando il più possibile la perdita di dati. Il sistema deve essere scalabile per future modifiche e aggiunte di nuove funzionalità oppure per l'utilizzo di nuovi dispositivi che possono avere un protocollo di comunicazione diversa da quella utilizzata attualmente.

Le prestazioni minime richieste sono quelle di poter gestire i due dispositivi già in produzione, eONE ed eFOUR, alla massima frequenza di campionamento (200 kHz) per una bit-rate calcolata di 12 MBit/s per eONE e circa 30 MBit/s per l'eFOUR. Deve inoltre essere predisposto a gestire il nuovo dispositivo a sedici canali (eSIXTEEN) alla massima frequenza per una bit-rate totale di circa 103 MBit/s.

3.1.4 Utente Finale

Il tipico utente finale è *l'elettrofisiologo* che effettua operazioni di lettura e analisi dei dati durante gli esperimenti sui canali ionici. E' quindi necessario rendere disponibili specifiche funzionalità che semplifichi il lavoro all'utente finale, con un interfaccia utente semplice da usare ma che presenti anche diverse funzionalità tecniche utilizzate nel campo dell'elettrofisiologia. Normalmente l'utente utilizza tutte le funzionalità del software per un tempo prolungato, è dunque necessaria l'affidabilità verso il sistema.

3.1.5 Ambiente d'uso e Limitazioni

Il sistema dovrà essere multi-piattaforma, quindi dovrà essere eseguita sia in macchine Windows che in macchine Mac di fascia medio-alta, mantenendo buone prestazioni e affidabilità. Il sistema progettato deve effettuare letture in real-time, dai dispositivi specificati, memorizzazione e analisi con la minima latenza per evitare perdita di dati. I dispositivi sono dotati tutti di driver di interazione con il computer, sarà quindi necessario installare questi prima dell'utilizzo di essi.

3.1.6 Linguaggio e piattaforma di sviluppo

La scelta del linguaggio di programmazione (C++) e del framework Qt è stato necessario per mantenere alcuni requisiti che il sistema deve avere, quali efficienza e portabilità. Per mantenere una latenza bassa è stato necessario un linguaggio che non passa prima da una macchina virtuale, ma che comunichi direttamente con il sistema operativo, es per la lettura dei dati da USB. La scelta di Qt è stata vincolata anche a una continuità con i software preesistenti e sviluppati dall'azienda, in modo da poter unire tutto in un unico sistema.

3.2 Interfacciamento con l'esterno

3.2.1 Interfaccia utente

Il software deve fornire un'interfaccia utente completa, intuitiva e funzionale mantenendo caratteristiche della versione precedente per fornire una continuità agli utenti finali. L'interfacci deve essere reattiva, rispondendo sempre a comandi dell'utente anche se esegue operazioni complesse o letture ad alta frequenza, mostrando graficamente i dati letti. La UI (user interface) deve essere contenuta tutta in un unica finestra ma con la possibilità di personalizzazione da parte dell'utente.

3.2.2 Interfaccia Hardware

I dispositivi sono collegati al computer via USB, verrà quindi utilizzato il driver FTDI(sezione 2.3) per la comunicazione con essi. Il software deve prevedere la possibilità di nuovi tipi di interfacciamento hardware (es USB 3.0) che usano driver diversi.

3.2.3 Interfaccia software

Il software comunica col dispositivo secondo un protocollo definito da Elements prima della creazione del software. Deve consentire la memorizzazione dei dati acquisiti in diversi formati per poter fare post-analisi dei dati da altri software, interni (EDA[1]) o esterni (Mathlab), garantendo quindi interoperabilità con essi.

[1]EDA (Elements Data Analyzer) è un software che consente funzionalità di analisi in post processing dei dati letti dal software. E' stato sviluppato dallo stesso team durante lo svolgimento del tirocinio presso l'azienda.

3.3 Suddivisione in moduli

Verranno descritte dettagliatamente i diversi moduli introdotti precedentemente.

3.3.1 Riconoscimento Dispositivi

Il software controlla periodicamente i dispositivi connessi e rilevare quando un nuovo dispositivo viene connesso, oppure quando un dispositivo viene rimosso. L'utente potrà scegliere il dispositivo da connettere, con possibilità di cambiare file di configurazione. Una volta connesso il dispositivo saranno abilitate le funzionalità di memorizzazione e analisi.

3.3.2 Acquisizione dati

Una volta connesso il dispositivo si avvia l'acquisizione dati dal dispositivo che può opportunamente essere configurato. I dati letti vengono elaborati e applicati i vari valori di calibrazione e offset e memorizzati in un buffer che viene gestita come un producer e consumer, questo è il caso del producer. L'acquisizione deve essere affidabile, limitando la perdita di dati e mantenendo la sincronizzazione mediante il protocollo di comunicazione del dispositivo.

3.3.3 Elaborazione Dati

Questo modulo si occupa della spartizione dei dati tra i diversi moduli, è il consumer nella gestione del buffer producer e consumer introdotta precedentemente. Dev'essere gestito da un thread diverso da quello di acquisizione per evitare ritardi in lettura e quindi perdita di dati.

3.3.4 Visualizzazione Dati in Real-Time

Il software deve fornire i dati letti in output all'utente graficamente, dove nell'ascissa vengono mantenuti i tempi e nell'ordinata la corrente (pA o nA), oppure la tensione (mV). In caso di più canali deve visualizzare tutti i canali con possibilità di mostrarne solo uno di interesse. Deve inoltre fornire delle funzionalità di interazione come la selezione dell'intervallo di tempo da mostrare oppure il range delle ordinate da mostrare. L'interfaccia utente deve rimanere reattiva nonostante la visualizzazione dei dai in real-time con elevato throughput.

3.3.5 Memorizzazione Dati

Il software deve gestire la memorizzazione dei dati su disco in diversi formati per poter fare analisi in post-processing. Prevede un file di intestazione dove sono salvati alcune informazioni necessarie per il post processing come la frequenza di campionamento e il range. I dati verranno salvati su file di diversi formati:

- *.dat* formato binario dove i dati mantengo un ordine rispetto ai canali acquisiti.
- *.csv* formato testuale dove i dati vengono incolonnati rispetto al canale di riferimento, questo formato occupa più risorse.
- *.abf* formato proprietario dell'azienda *Axon*, permette l'interoperabilità con software di terze parti per l'analisi in post-processing.

La memorizzazione dei dati deve essere fatta in modo efficiente e affidabile, quindi dovrà fare un ottimo utilizzo delle risorse (tempo e memoria) per non diventare il collo di bottiglia del software in questione, e garantire il salvataggio di tutti i dati che gli arrivano dal modulo di acquisizione.

3.3.6 Analisi dei Dati in Real-Time

Il software deve fornire la possibilità di effettuare analisi in real-time dei dati. E' molto importante in questo modulo l'efficienza quanto le operazione di analisi possono essere algoritmi che non hanno complessità computazionale lineare. Le funzionalità di analisi attualmente fornite sono:

- **Statistica:** vengono fatte statistiche sui dati letti come media e deviazione standard.
- **Istogramma:** viene calcolato l'istogramma in base alla dimensione dei bin definito dall'utente, il risultato più frequente è una serie di gaussiane.
- **Event Detection:** Rileva gli eventi di cambio di stato rispetto a due soglie definite dall'utente. Il risultato sarà la media della durata tra ciascun evento.
- **Dwell Time:** Rileva gli eventi di cambio di stato rispetto a due soglie definite dall'utente. Il risultato sarà la media della durata di ciascun evento.

- **FFT:** viene calcolata la trasformata discreta di fourier con l'algoritmo chiamato *Fast Fourier Trasform* (FFT), con costo computazionale O($nlog(n)$) su un numero di bin definito dall'utente. Il risultato mostrato è il modulo.
- **Grafico I/V:** mostra graficamente la corrente in corrispondenza dei valori di tensione per stimare la conduttanza del canale ionico sotto esame. Solitamente segue una funzione lineare si può quindi interpolare con una retta.

Capitolo 4

Analisi

In questo capitolo verrà presentata un'introduzione dei vari moduli.

Verrà inoltre effettuata un'analisi approfondito dei moduli da me implementati. Inoltre verrà introdotto il file di configurazione e il protocollo con cui la view interagisce col dispositivo.

4.1 Suddivisione in Moduli

Come già introdotto nelle analisi dei requisiti (capitolo 3) il lavoro è stato suddiviso in moduli, per una maggiore scalabilità e manutenibilità. La suddivisione è stata effettuata nella seguente modalità:

- Rilevamento
- Connessione e Acquisizione
- Elaborazione
- Visualizzazione
- Memorizzazione
- Analisi

4.2 Suddivisione del lavoro

Date la complessità del sistema da progettare, il lavoro è stato effettuato da due persone. E' stato quindi necessaria una divisione del lavoro in base ai moduli presentati.

4.2.1 Diagramma a blocchi dei moduli

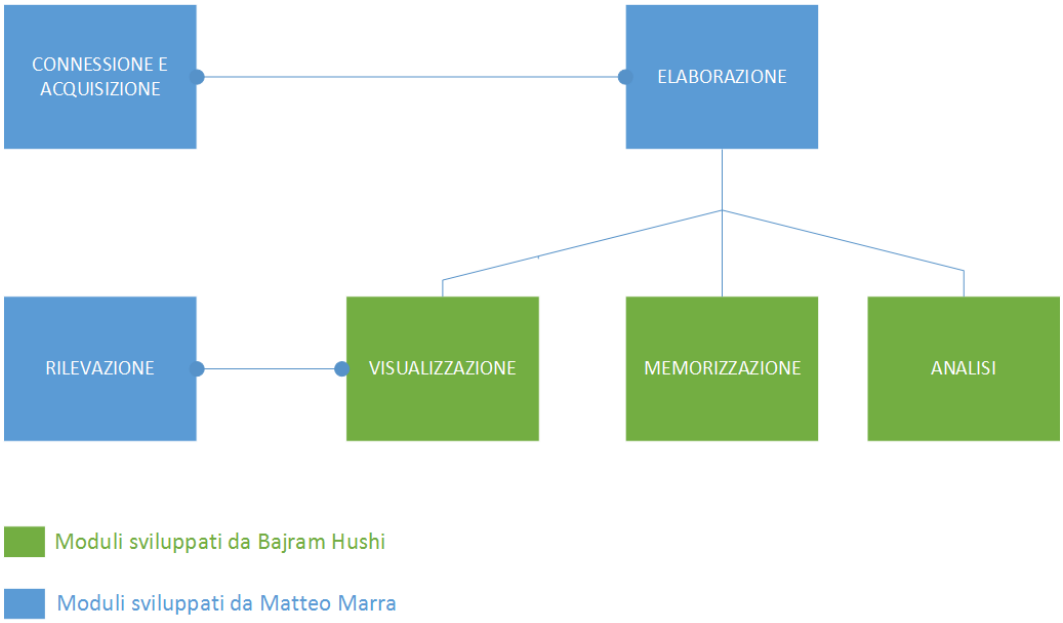


Figura 4.1: Suddivisione del lavoro

4.2.2 Processo di produzione

Data le specifiche richieste è stato opportuno seguire un processo di produzione del software. Il processo seguito si è basato sul modello RAD (Rapid Application Development). Il modello RAD è un modello di processo incrementale che punta a un ciclo di sviluppo molto breve. Ogni applicazione modularizzabile in modo che ciascuna funzionalità principale sia completata entro tre mesi è candidata al RAD. I vari moduli vengono sviluppati parallelamente da team diversi.

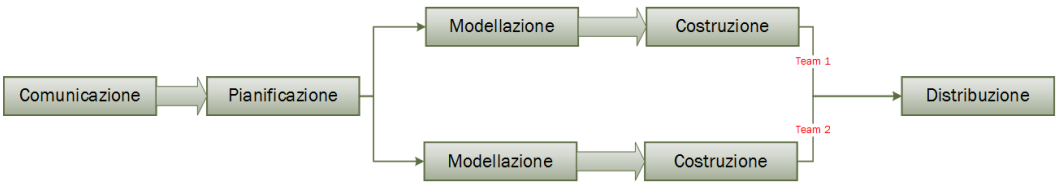


Figura 4.2: Diagramma a blocchi del modello RAD

4.2.3 Comunicazione tra i moduli

Siccome il progetto è stato fatto da due persone è stato necessario definire la comunicazione tra i diversi moduli. Lo sviluppo di queste parte è stata effettuata da entrambi gli elementi del team in modo tale da ottenere una comunicazione ottimale mediante la definizione di specifiche strutture dati.

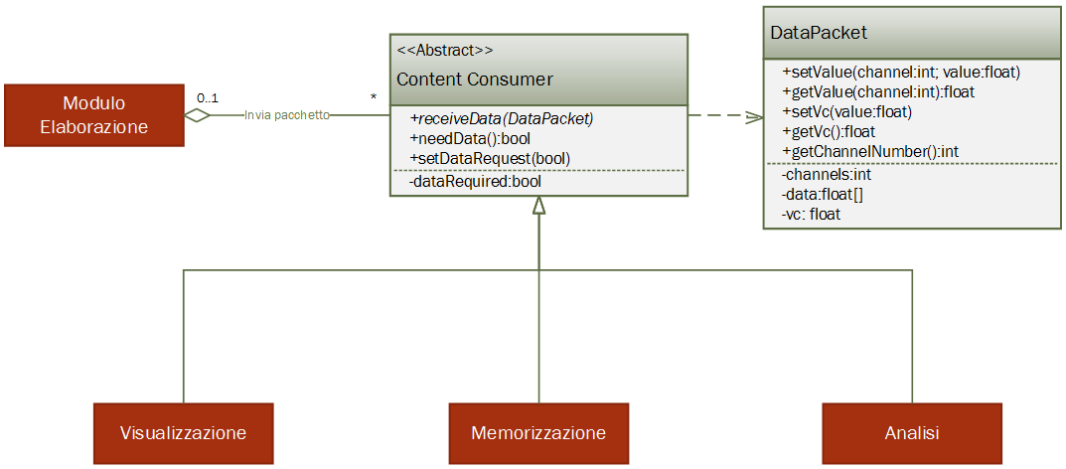


Figura 4.3: Diagramma delle classi che introduce la comunicazione tra moduli

Tutte le componenti che devono agire sui dati dovranno estendere la classe *ContentConsumer* e registrare la classe nel modulo di elaborazione. I dati passeranno in una classe specifica, *DataPacket*, per mantenere semplificato il passaggio dei dati. Una classe che estende *ContentConsumer* deve implementare il metodo astratto *receiveData(data:DataPacket)* e utilizzare i dati che riceverà periodicamente.

Nota: Da questo punto in avanti verranno descritti solo i moduli progettati e implementati da me.

4.3 Analisi Modulo di Visualizzazione

4.3.1 Specifiche

Realizzare un modulo che legga dei dati da un dispositivo collegato e mostri all'utente i grafici in real-time dei dati acquisiti. L'utente deve avere la possibilità di personalizzare i grafici, le funzionalità richieste sono:

- finestra del tempo (in secondi) da visualizzare.
- ampiezza del segnale letto.
- traslazione dei dati
- trigger del segnale
- sovrapposizione dei segnali letti
- personalizzazioni grafiche

L'utente può inoltre configurare il dispositivo in base a un file di configurazione. La visualizzazione dei dati deve essere fatta in real-time si deve quindi mantenere in interazione fluida col software da parte dell'utente.

4.3.2 Casi d'uso

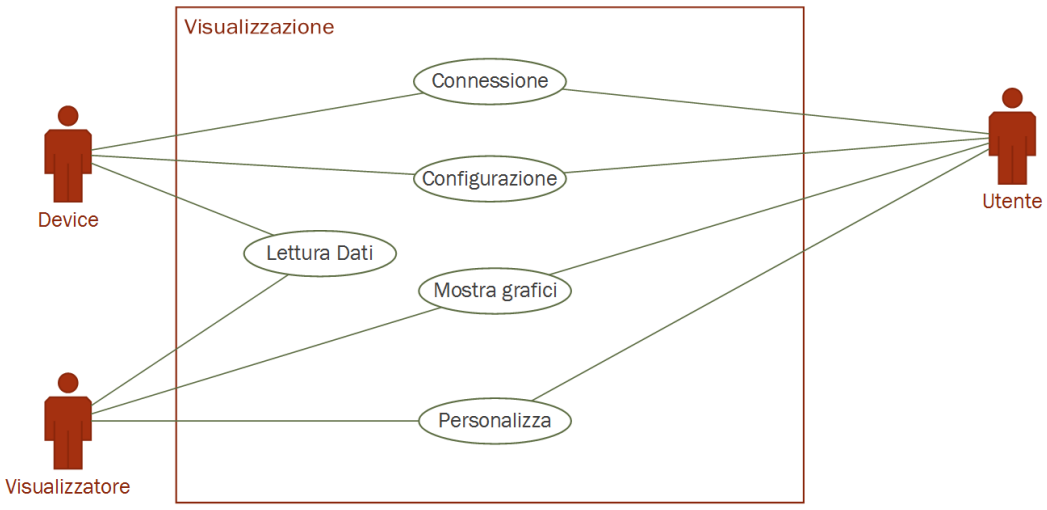


Figura 4.4: Casi d'uso dello scenario di visualizzazione

4.3.3 Descrizione dello Scenario

Caso d'uso: Connessione

Scenario Base

1. L'utente collega, tramite un cavo USB, il dispositivo al PC
2. Il software riconosce il dispositivo collegato
3. L'utente seleziona il dispositivo da connettere
4. Viene aperto un canale di comunicazione con il dispositivo selezionato
5. Il dispositivo riceve una configurazione iniziale

Varianti

2.a Il riconoscimento non avviene per errori nel collegamento, resta in attesa fino a quando non rileva un dispositivo.

4.a Errore nell'apertura del dispositivo. Lo scenario torna al punto 2.

5.a Errore nel caricamento della configurazione. La connessione viene chiusa e si ritorna al punto 2.

Caso d'uso: Configurazione

Scenario Base

1. L'utente, mediante l'interfaccia grafica, seleziona i parametri di configurazione del dispositivo
2. Il dispositivo riceve i parametri di configurazione e si comporta di conseguenza

Varianti

2.a Avviene un errore nella trasmissione della configurazione. I dati letti non sono coerenti alla configurazione selezionata.

Caso d'uso: Lettura dati & Visualizzazione grafici

Scenario Base

1. Il dispositivo produce dati
2. Il visualizzatore riceve e bufferizza dati
3. I dati vengono mostrati graficamente all'utente

Varianti

2.a A frequenze alte molti dati vengono scartati per mantenere buone prestazioni.

Caso d'uso: Personalizzazione

Scenario Base

1. l'utente sceglie la finestra del tempo (in secondi) da visualizzare.
2. l'utente sceglie ampiezza del segnale letto.
3. l'utente effettua la traslazione dei dati per focalizzarsi su determinati eventi.
4. l'utente richiede il trigger del segnale per visualizzare la forma d'onda
5. l'utente effettua sovrapposizione dei segnali letti per confrontare diversi segnali
6. l'utente effettua personalizzazioni grafiche

Varianti

1.a Se la finestra da visualizzare è troppo grande si effettua campionamento dei dati.

1.b Se l'utente non sceglie una finestra, ne verrà selezionata una di default e lo scenario continua.

2.a Se l'utente non sceglie un'ampiezza, ne verrà selezionata una di default e lo scenario continua.

3.a Se l'utente non effettua una traslazione, ne verrà selezionata una di default e lo scenario continua.

4.a Se l'utente non richiede il trigger i dati verranno mostrati come arrivano e lo scenario continua.

5.a Se l'utente non effettua una sovrapposizione, ogni segnale verrà mostrato in un grafico a parte e lo scenario continua.

4.3.4 Diagramma delle attività

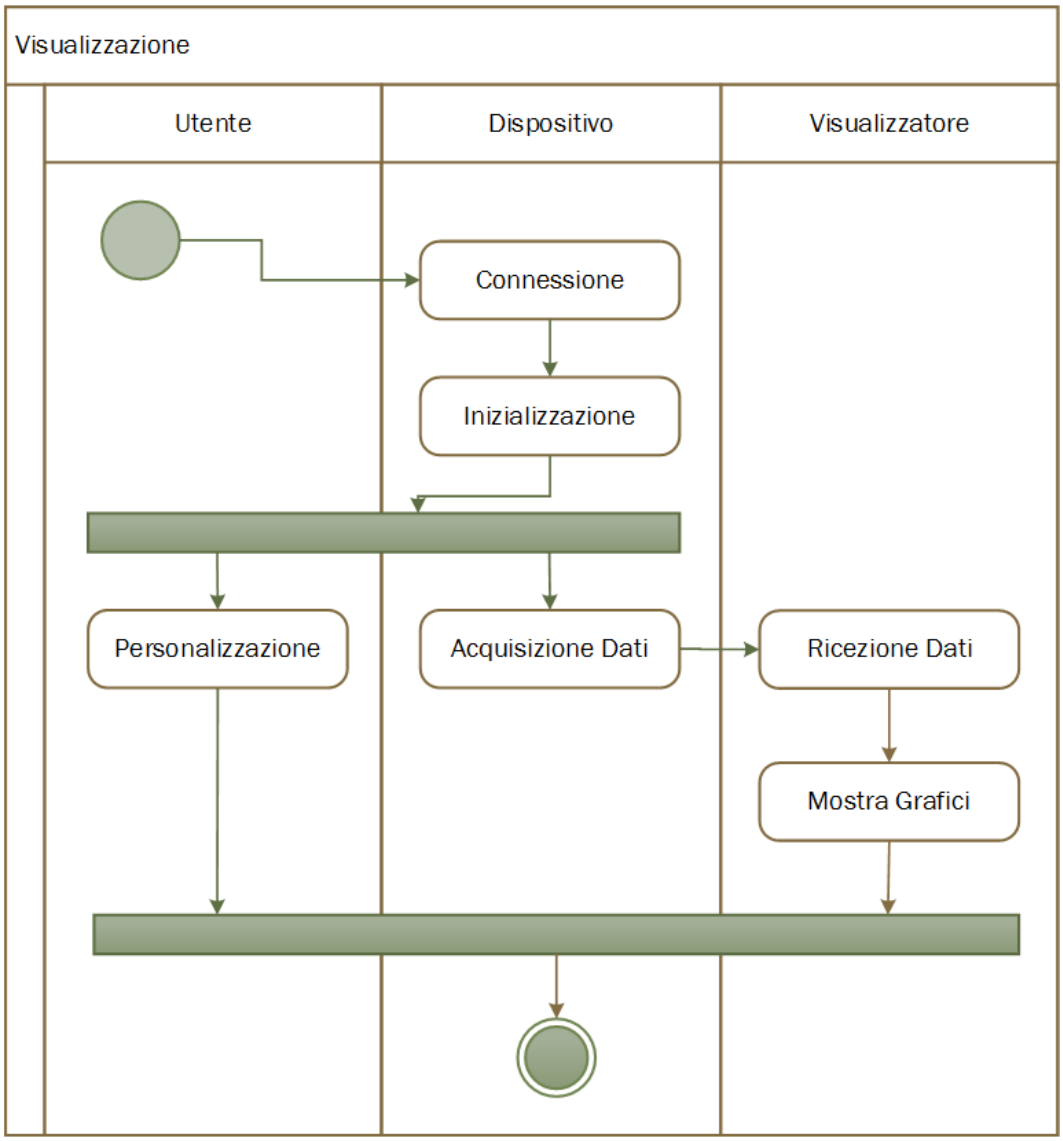


Figura 4.5: Diagramma delle attività che spiega il caso d'uso

4.3.5 Diagramma delle classi

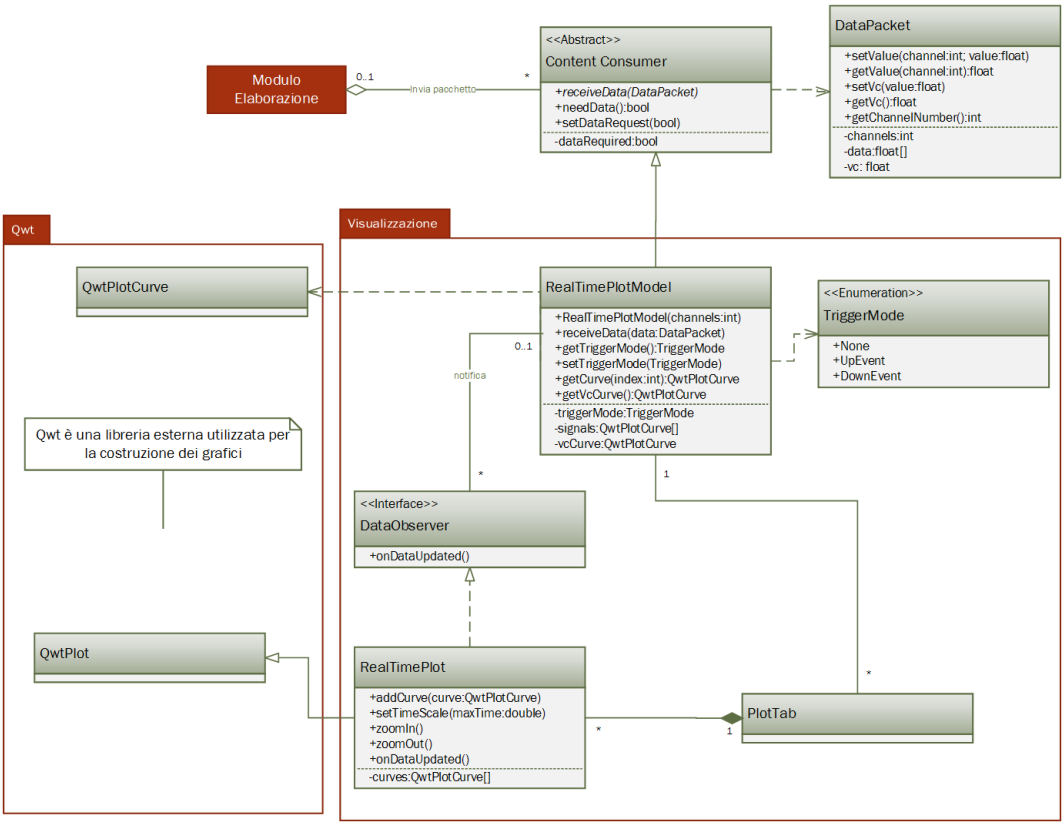


Figura 4.6: Diagramma delle classi del modulo di visualizzazione

4.4 Analisi Modulo di Memorizzazione

4.4.1 Specifiche

Realizzare un modulo che legga dei dati da un dispositivo collegato e salvi i dati su disco in diversi formati. L'utente deve avere la possibilità di impostare diverse preferenze, quali:

- il percorso della cartella dove verranno salvati i dati
- il nome dei file
- il formato del file
- separazione dei dati in più file
- stop automatico in base ai minuti impostati dall'utente

I formati richiesti sono:

* **.dat:** formato binario dove i dati letti dal dispositivo devo essere salvati secondo un formato specifico per mantenere l'interoperabilità con altri software. Il formato richiesto deve mantenere la seguente forma:

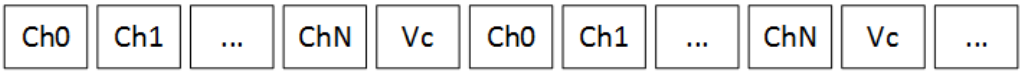


Figura 4.7: Formato file binario, N indica il numero di canali

* **.csv:** formato testuale (comma separed value) che mantiene lo stesso formato dei file binari ma separa i valori tra loro con la virgola. Il formato non è adatto per alte frequenze in quanto la memoria utilizzata è maggiore.

* **.abf:** l'ABF (AXON BINARY FILE) è un formato proprietario, può quindi essere letto e creato dai sviluppatori di terze parti usando la libreria ABFFIO. Il formato del file ABF è il seguente. [7]

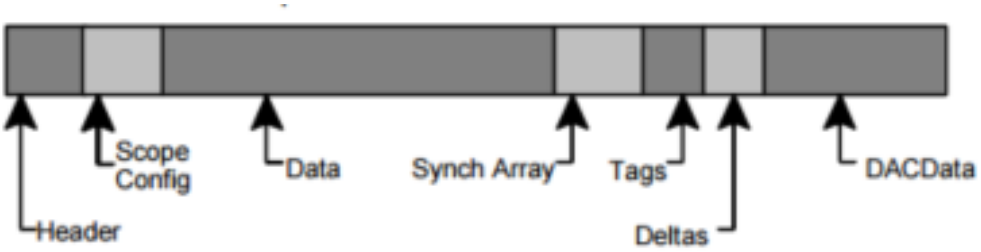


Figura 4.8: Formato ABF

Mantenendo il seguente formato si assicura l'interoperabilità con software di terze parti che sono in grado di leggere i file ABF, come Elements Data Analyzer oppure Mathlab.

La memorizzazione dei dati deve essere fatta in modo affidabile, salvando tutti i dati che arrivano nell'esatto ordine in cui arrivano mantenendo il software sempre reattivo ai comandi degli utenti. E' necessario inoltre un file di intestazione, con formato *.edh* che riporti alcune informazioni sui dati salvati come la frequenza di campionamento, il range selezionato, la data di acquisizione ecc..

4.4.2 Casi d'uso

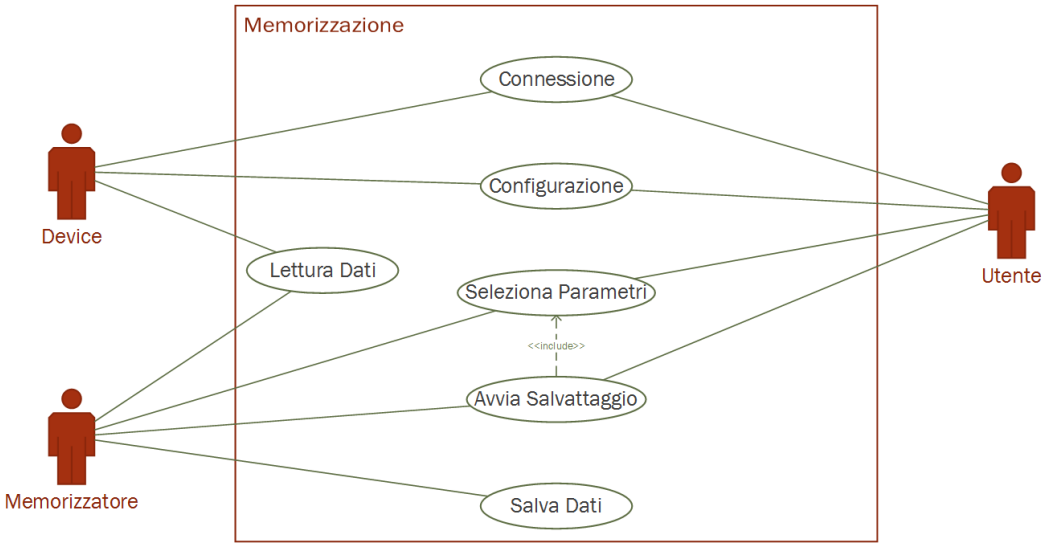


Figura 4.9: Casi d'uso dello scenario di memorizzazione

4.4.3 Descrizione dello Scenario

I casi d'uso di connessione, configurazione e lettura dati non verranno illustrati quanto sono già stati descritti nella sezione precedente.

Scenario Base

1. Connessione dispositivo
2. L'utente richiede il salvataggio dei dati
3. L'utente seleziona i parametri di salvataggio
4. L'utente avvia il salvataggio dei dati
5. il memorizzatore crea il file di intestazione
6. Il memorizzatore bufferizza i dati che riceve
7. Il memorizzatore salva i dati su disco

Varianti

1.a Errore nella connessione del dispositivo, lo scenario termina.

5.a Errore nella scrittura su disco. Lo scenario termina.

7.a Errore nella scrittura su disco. Lo scenario termina.

4.4.4 Diagramma di sequenza

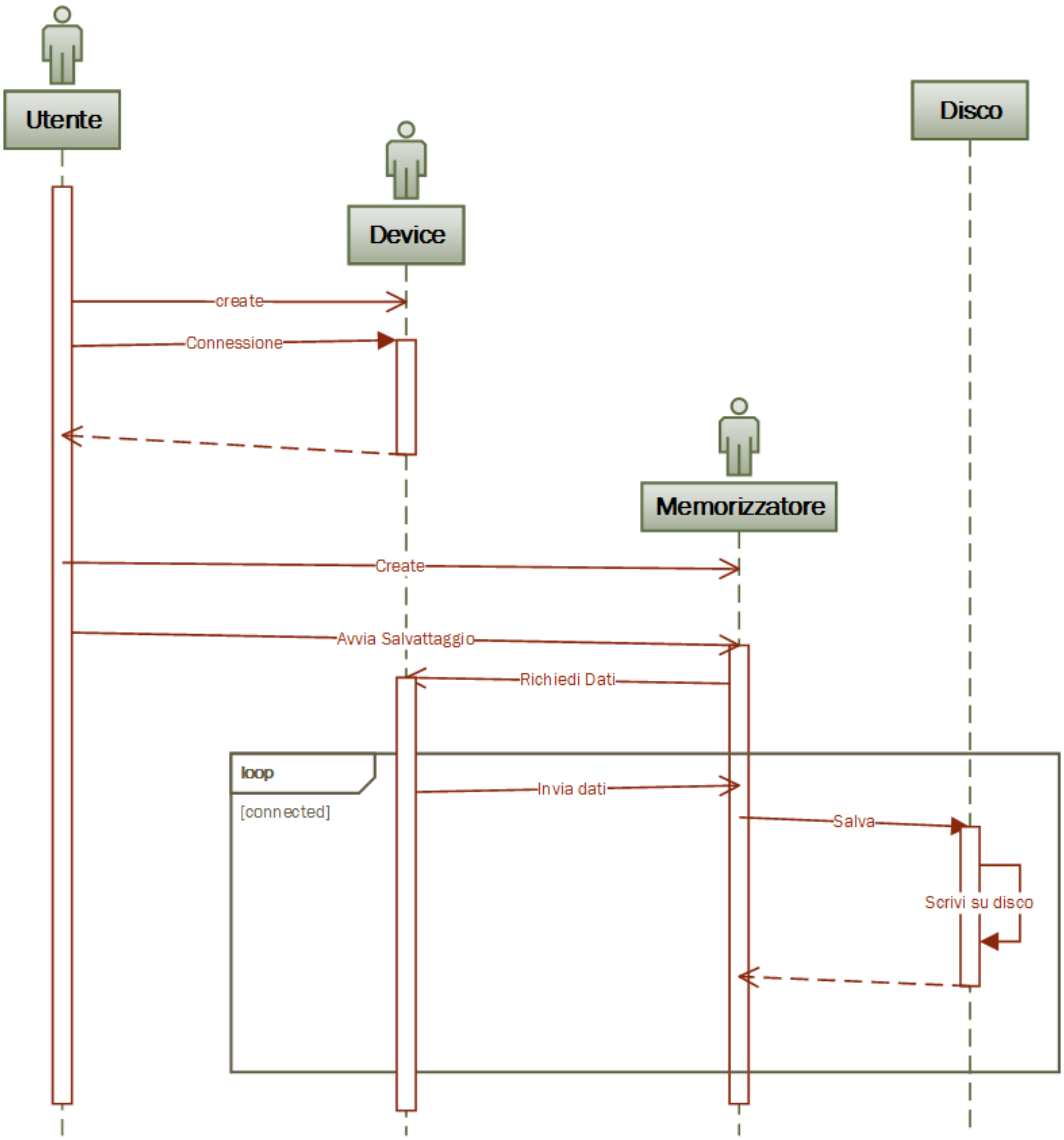


Figura 4.10: Diagramma di sequenza che spiega i casi d'uso

4.4.5 Diagramma delle classi

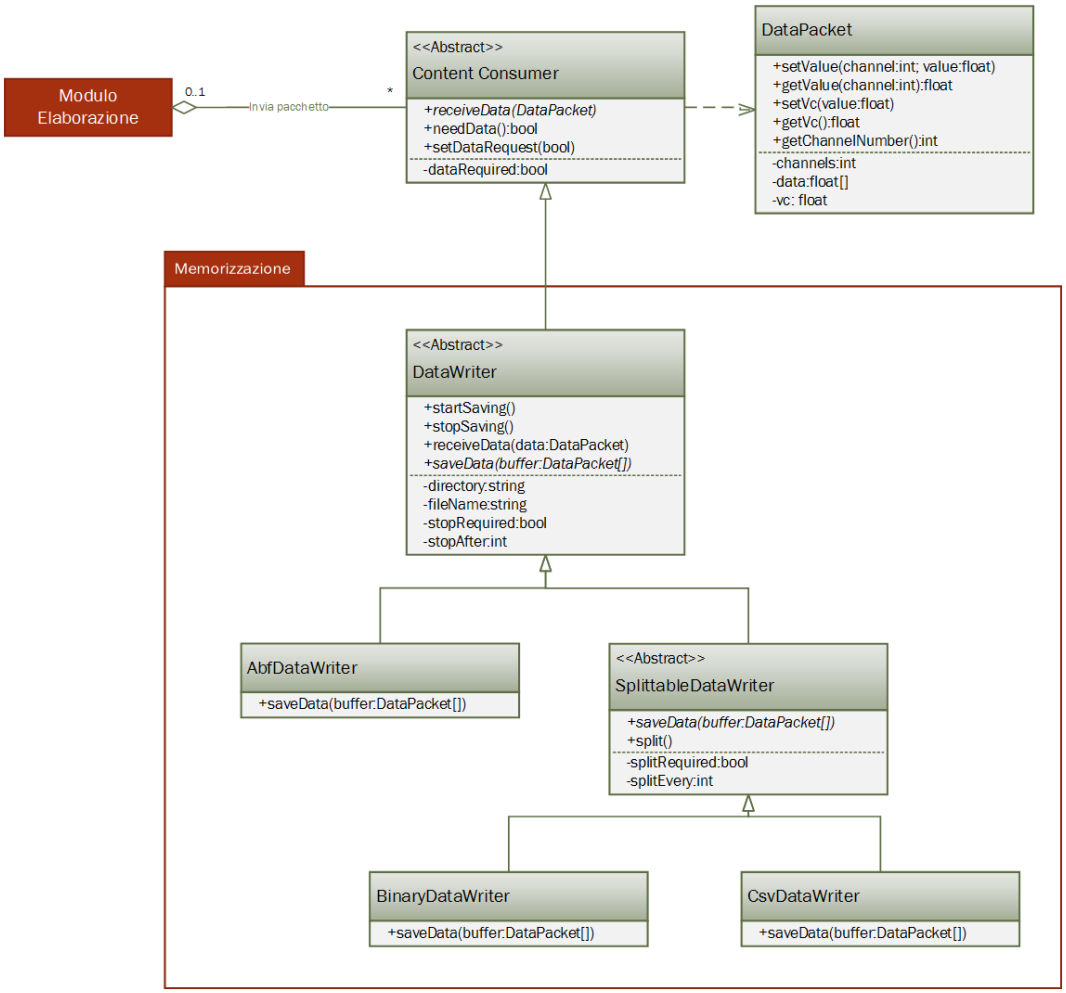


Figura 4.11: Diagramma delle classi del modulo di memorizzazione

4.5 Modulo di Analisi

4.5.1 Specifiche

Realizzare un modulo che legga dei dati da un dispositivo collegato ed effettui analisi in real-time dei dati. Le analisi richieste sono:

- **Statistiche:** vengono calcolate le statistiche sui dati che vengono letti su intervalli di tempo definiti. Tra le statistiche richieste ci sono la media, la deviazione standard e la varianza. I risultati devono essere mostrati all'utente con aggiornamento periodico.
- **Istogramma:** Viene costruito l'istogramma dei dati acquisiti in tempo reale, l'utente deve avere la possibilità di iniziare, resettare e fermare l'analisi. Deve inoltre scegliere la dimensione dei Bin.
- **FFT:** Costruire la trasformata discreta di fourier dei dati acquisiti e mostrare graficamente il modulo.L'utente deve avere la possibilità di seleziona i Bin sui quali fare l'analisi, maggiore è il numero di bin maggiore è l'affidabilità del risultato.

Il modulo di analisi deve prevedere nuovi tipi di analisi che verranno aggiunti in seguito e mantenere deve essere quindi progettato in modo tale da essere scalare e mantenere un buon utilizzo delle risorse quanto gli algoritmi possono diventare dispendiosi.

4.5.2 Diagramma delle classi

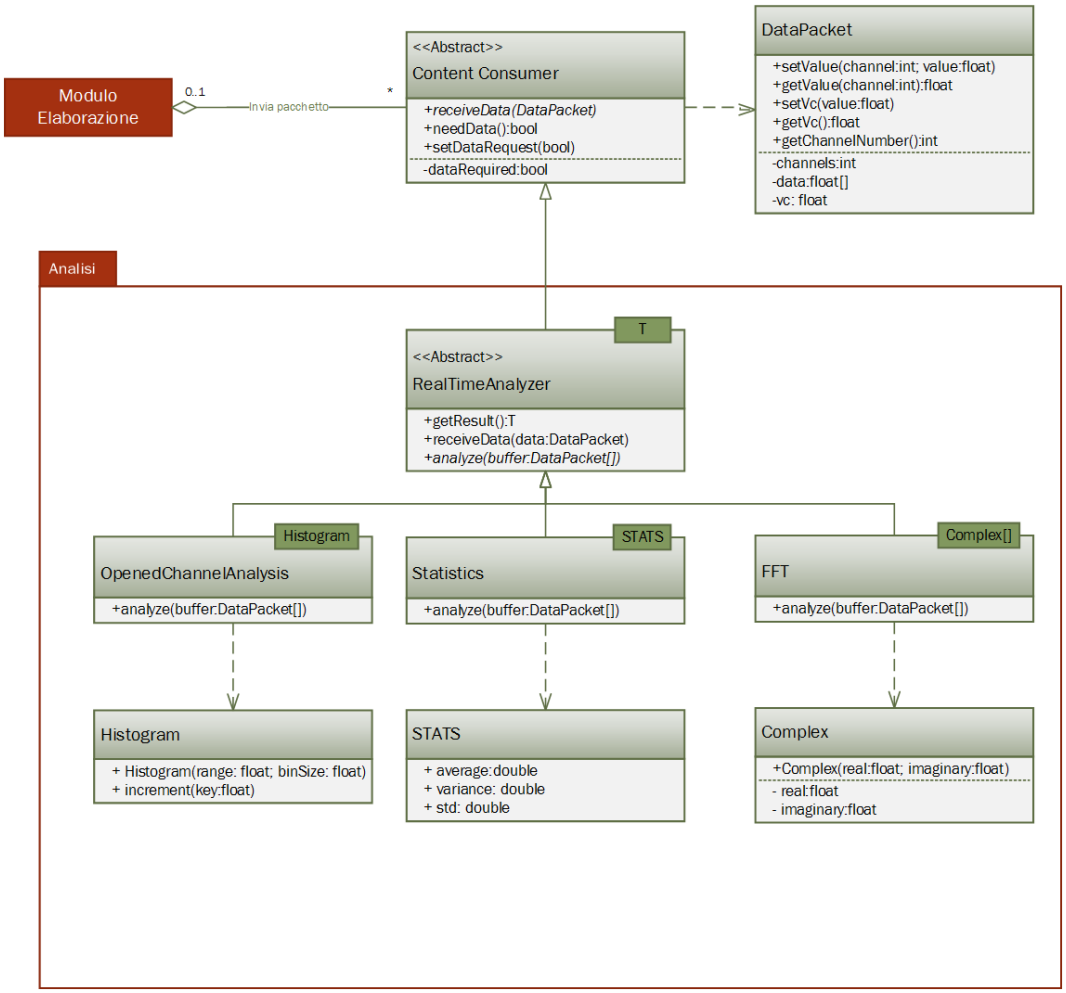


Figura 4.12: Diagramma delle classi del modulo di memorizzazione

Tutte le classi di analisi estendono la classe generica *RealTimeAnalyzer* e dovranno fornire l'output col metodo *getResult():T*. Chi chiama questo metodo avrà il risultato delle analisi fino a quel momento.

4.6 Il file di configurazione

Il file di configurazione è un file che contiene tutti i parametri di configurazione necessari per l'interfacciamento del dispositivo col software. E' diverso per ogni versione e sotto-versione dei dispositivi e mantenuto criptato perché contiene informazioni sensibili. Viene ereditato dalla versione precedente del software e leggermente ottimizzato per adattarlo alle specifiche richieste. Le informazioni principali contenuti in esso sono:

- Informazioni sul dispositivo, come numero dei canali.
- Protocollo di comunicazione col dispositivo.
- Sequenza iniziale di configurazione del dispositivo.
- Definizione delle maschere di configurazione.
- Definizione di componenti di interfaccia grafica necessari per la costruzione della view, necessario per il cambiamento dei parametri del dispositivo da parte dell'utente.

L'utilizzo del file di configurazione permette l'utilizzo di nuovi dispositivi senza necessariamente cambiare il codice sorgente del software.

4.7 Qualità del software richieste

Il software dovrà mantenere alcune qualità in quanto sarà distribuito ai clienti. Le qualità richieste sono:

- **Affidabilità:** Il software dovrà mantenere un comportamento affidabile data la sensibilità dei dati.
- **Robustezza:** Il software dovrà mantenere un comportamento corretto anche in circostanze esterne del software stesso (es. rimozione fisica del dispositivo).
- **Efficienza:** Il software dovrà fare un utilizzo intelligente delle risorse, dato il continuo flusso di dati.
- **Facilità d'uso:** Il software dovrà presentare un interfaccia utente che permetta di esprimersi in modo intuitivo.
- **Portabilità:** Il software sarà distribuito su diverse piattaforme.
- **Facilità di manutenzione:** Il software dovrà essere progettato in modo tale da permettere la manutenzione a prodotto finito dato che saranno necessarie modifiche perfettive per l'aggiunta di nuove funzionalità.
- **Interoperabilità:** I dati prodotti dal software saranno poi utilizzati da altri software per analisi in post processing, dovranno quindi avere un formato compatibile.

Capitolo 5

Progettazione

In questo capitolo verranno descritti i vari moduli per quanto riguarda le scelte progettuali che hanno portato alla loro definizione. Verrà inoltre descritta la progettazione dell'*interfaccia utente*.

5.1 Multithreading

Per soddisfare i requisiti di qualità del software, già presentati nella sezione 4.7, è stato necessario l'utilizzo del multithreading. Questo permette l'esecuzione parallela dei vari moduli migliorando in modo significativo le prestazioni dell'applicazione. Questo approccio inoltre permette di mantenere un'interfaccia grafica reattiva ai comandi dell'utente, nonostante l'esecuzione degli altri moduli di memorizzazione e analisi.

Concettualmente il modulo di acquisizione, rilevazione ed elaborazione girano su thread diversi e ogni **ContentConsumer**, sezione 4.2.3, è un thread a parte. Questa suddivisione permette alte prestazioni e comunicazione asincrona tra i moduli.

Concorrenza

L'utilizzo dei thread presenta i soliti problemi di concorrenza. Per mantenere l'affidabilità dovranno essere evitati con l'utilizzo della mutua esclusione e l'utilizzo di opportune strutture dati, come quella del *Producer and Consumers* utilizzata per la comunicazione tra il modulo di acquisizione ed elaborazione.

5.2 Progettazione Interfaccia Grafica

L'interfaccia grafica deve essere progettata in modo tale da essere di facile utilizzo all'utente, consentendo di utilizzare funzionalità avanzate in modo semplice e intuitivo. Deve essere inoltre incorporato in un unica finestra ma con la possibilità di suddividerla in più finestre per dare maggiore flessibilità a chi usa più di uno schermo.

5.2.1 Connessione del dispositivo

L'interfaccia di connessione è gestita con il pattern MVC (Model View Controller), il model mantiene lo stato in cui è il dispositivo e notifica la view quando si presenta uno cambio di stato, la view si adatta in base allo stato e alle informazioni presenti nel model, il controller cattura gli eventi dell'utente e agisce sul model. La gestione degli eventi verrà fatta mediante l'opportuno utilizzo delle signal e slot presenti in Qt (sezione 2.2).

MVC Pattern

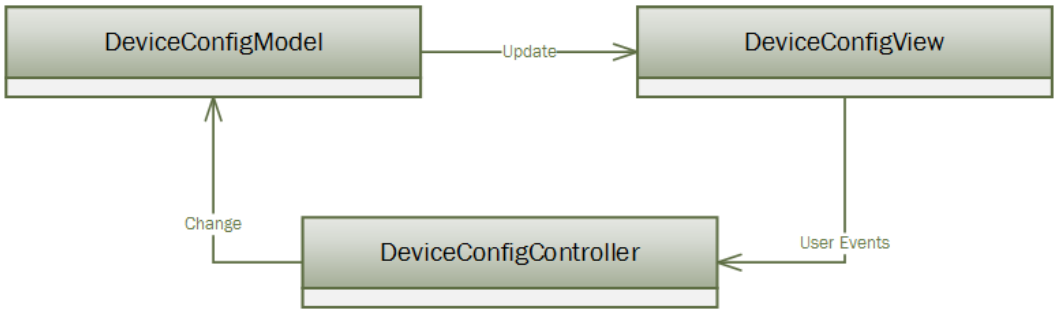


Figura 5.1: Pattern MVC

Model

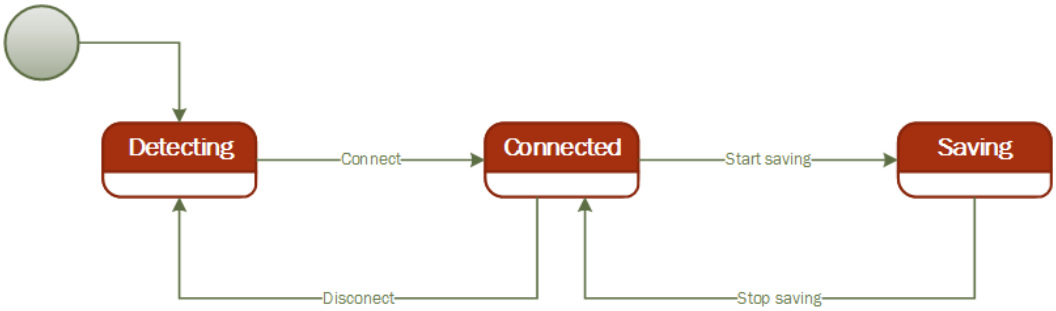


Figura 5.2: Macchina a stati finiti utilizzata

Descrizione degli stati:

- **Detecting**: deve essere possibile selezionare uno dei dispositivi collegati al pc, cambiare file di configurazione e connettere il dispositivo.
- **Connected**: è stato connesso uno specifico dispositivo dove si potrà interagire, si avrà la possibilità di disconnettere il dispositivo oppure di avviare il salvataggio dei dati.
- **Saving**: si avvia il salvattaggio dei dati su disco in base a preferenze definite dall'utente, verrà mostrato un timer per indicare il tempo di salvataggio dei dati acquisiti dal dispositivo connesso. Sarà possibile interrompere il salvataggio in qualsiasi momento.

View

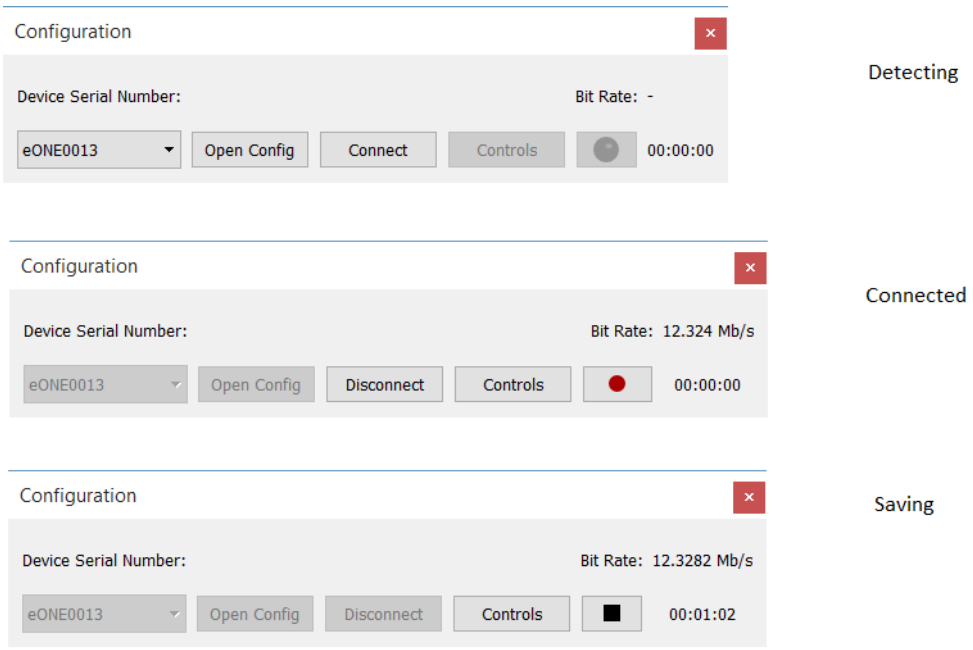


Figura 5.3: Comportamento della view rispetto al model

Come mostrata nella figura 5.3 la view abilita e disabilita controlli in base allo stato in cui si trova il model.

5.2.2 Configurazione

Una volta connesso un dispositivo viene costruita la view relativa alla configurazione del dispositivo, prendendo i parametri di configurazione dal *config file* citato nella sezione 4.6. I controlli creati sono legati a una maschera di configurazione che andrà a modificare la configurare il dispositivo.

Protocolli di tensione

Un'altra parte di configurazione del dispositivo fa riferimento ai protocolli di tensione, che comprende una serie di parametri per le varie configurazioni descritti nelle immagini di riferimento del protocollo selezionato. Nel file di configurazione sono definite anche le regole che i parametri devono rispettare, le regole sono definite con gli operatori elementari (somma, differenza, moltiplicazione, divisione e potenza) e operatori di confronto (maggiore, minore,

uguale ecc..), es $p1 + p2 < p3$. E' necessario quindi effettuare un parse di queste regole e controllare se sono rispettate.

Inoltre deve essere possibile effettuare il salvataggio e il caricamento dei valori selezionati dagli utenti in modo tale da poter riutilizzare la stessa configurazione in futuro.

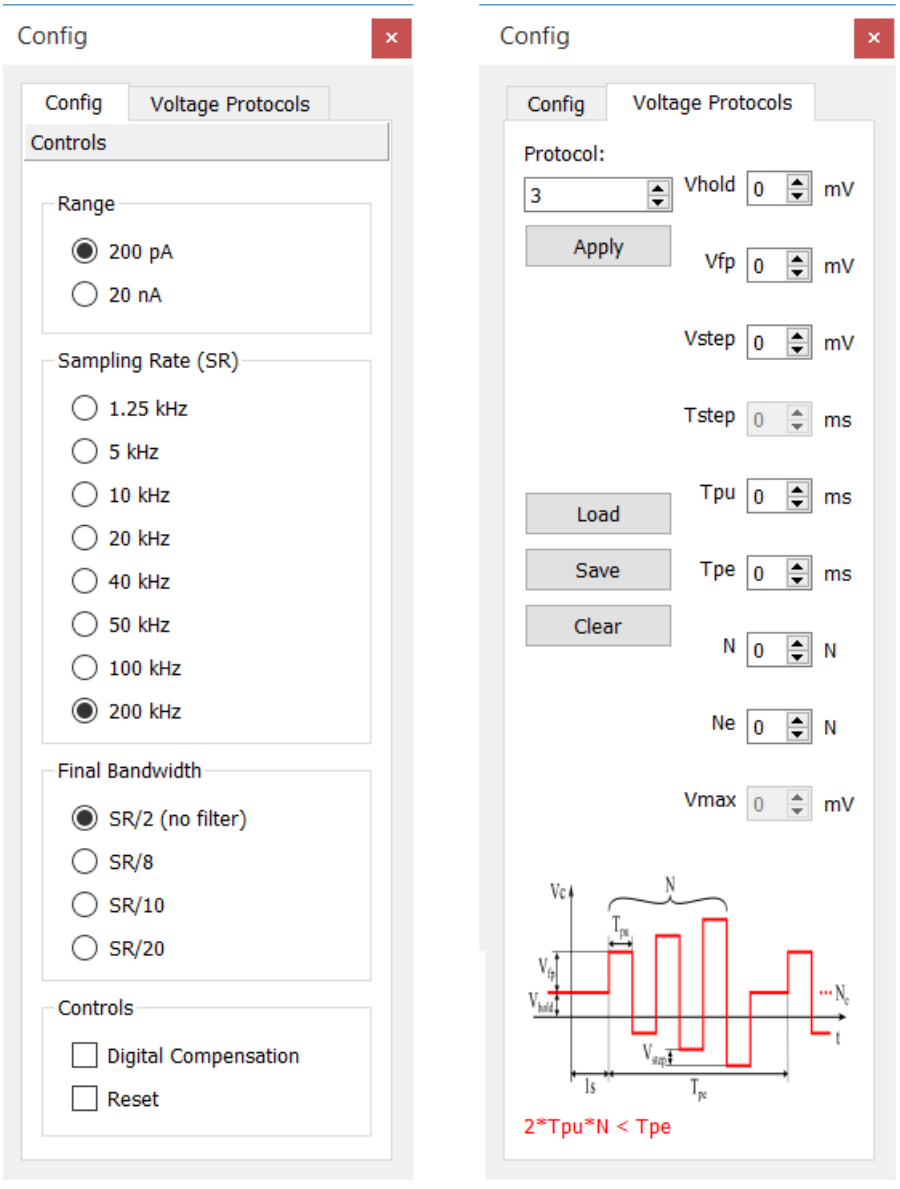


Figura 5.4: Configurazione dispositivo

5.3 Modulo di Visualizzazione

5.3.1 Progetto delle associazioni

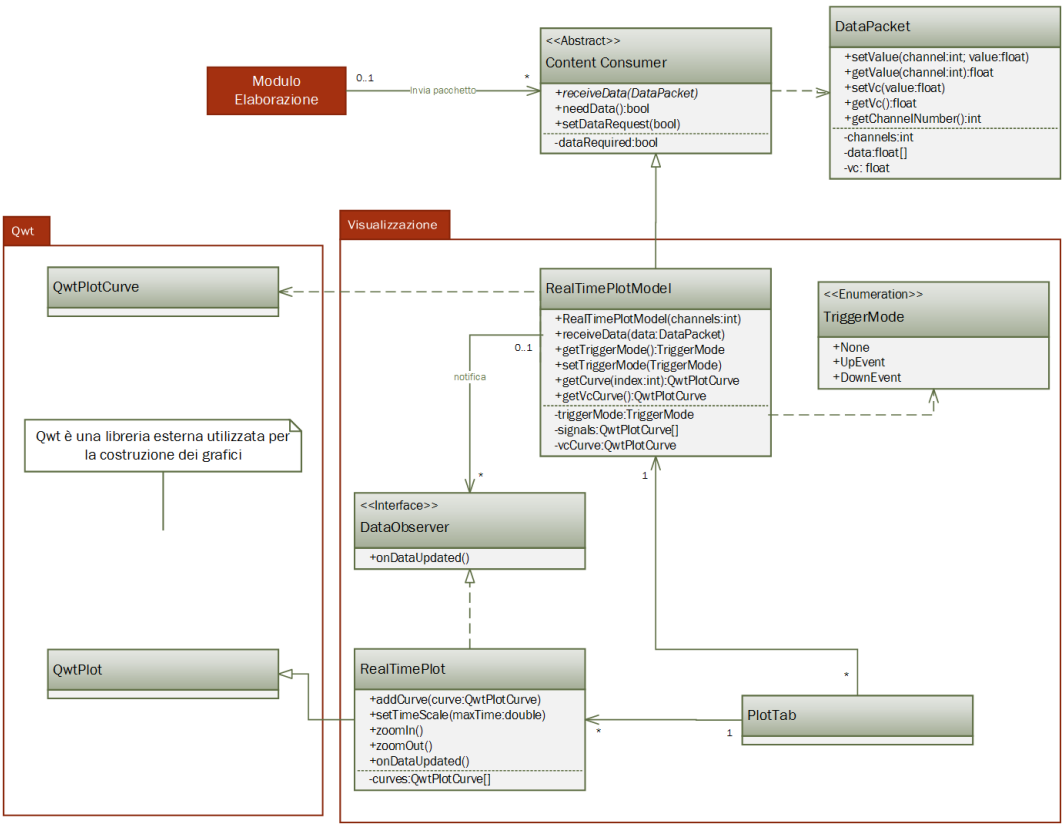


Figura 5.5: Progettazione delle associazioni

5.3.2 Pattern utilizzati

Il modulo rispetta il pattern MVC per mantenere separati i dati dalla view, in modo tale da poter creare nuove view senza bisogno di cambiare il modello di riferimento.

Per l'aggiornamento periodico della view viene utilizzato il pattern Observer.

5.3.3 Descrizione delle Classi

RealTimePlotModel

La classe gestisce i dati ricevuti dal modulo di elaborazione, costruendo le curve e l'asse dei tempi in base alla frequenza che viene settato. Gestisce inoltre il sotto-campionamento dei dati per mantenere alte prestazioni nonostante la grande mole di dati. Gestisce le funzionalità di triggering notificando la view quando è necessario un refresh del grafico.

PlotTab

Questa classe fa da controller nel pattern MVC e gestisce tutti i comandi dell'utente. Gestisce inoltre l'interazione tra il model e la view.

RealTimePlot

La classe mostra i grafici relativi ai dati ricevuti. Implementa l'interfaccia *DataObserver* per ricevere la notifica di refresh dal model. E' la view nel pattern MVC e costruisce i controlli di gestione dei grafici.

5.4 Modulo di Memorizzazione

Il modulo di memorizzazione deve essere progettato in modo tale da garantire una struttura facilmente scalabile per dare la possibilità di aggiungere nuovi formati di salvataggio, in modo tale da poter garantire maggiore interoperabilità con software di terze parti. Viene gestito con il pattern *Template*, creando la classe astratta, *DataWriter*, che estende *ContentConsumer* (sezione 4.2.3) e implementa i metodi comuni a tutti i formati, come la costruzione del file di intestazione. Implementa il metodo astratto *receiveData(DataPacket)* e riempe un buffer di pacchetti. Quando il buffer è pieno chiama in modo asincrono il metodo astratto *saveBuffer(DataPacket[])* (template method) per delegare il compito di salvataggio alle classi figlie. Questa struttura garantisce una facile scalabilità in quanto se si vuole aggiungere un nuovo formato basta solo implementare il metodo astratto. Fig. 4.10

5.4.1 Template Method

Come già descritto sopra per lo sviluppo di questo modulo è stato utilizzato il template method, ecco quindi una breve descrizione di che cos'è il template method.

[6]Il template method è un pattern comportamentale basato su classi, utilizzato in informatica nell'ambito della programmazione orientata agli oggetti. Questo pattern permette di definire la struttura di un algoritmo lasciando alle sottoclassi il compito di implementarne alcuni passi come preferiscono. In questo modo si può ridefinire e personalizzare parte del comportamento nelle varie sottoclassi senza dover riscrivere più volte il codice in comune. Template method è uno dei design pattern fondamentali della programmazione orientata agli oggetti definiti originariamente dalla cosiddetta gang of four, ovvero gli autori del libro Design Patterns.

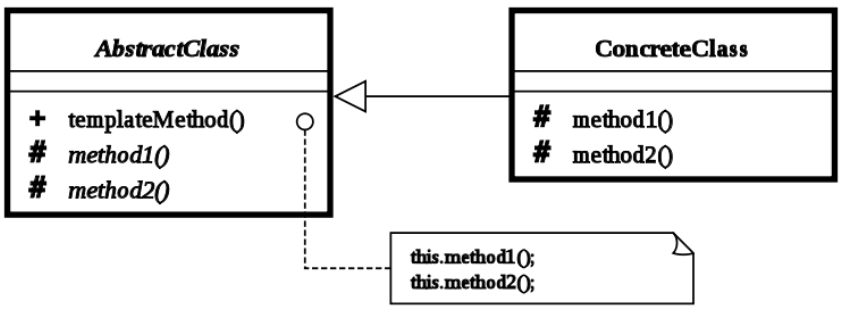


Figura 5.6: Diagramma delle classi che spiega il template method

5.5 Modulo di Analisi

Il modulo di analisi deve prevedere diversi tipi di analisi, però mantenendo la stessa interfaccia esterna. L'analisi viene fatta su buffer di determinata dimensione e restituisce un risultato che varia in base al tipo di analisi. E' stato quindi appropriato costruire una classe astratta, **RealTimeAnalyzer**, che abbia un risultato generico e un metodo *analyze(DataPacket[])* astratto, che verranno poi definita dalle classi concrete. Le classi figlie dovranno impostare il risultato nella concretizzazione del metodo *analyze(...)* in modo tale da garantire il corretto funzionamento della struttura progettata. Fig. 4.12

5.5.1 Statistiche

Le statistiche vengono calcolate applicando le formule matematiche.
Sia b un vettore di N elementi si calcola la media μ:

$$\mu = \sum_{i=1}^{N} b_i$$

La Varianza σ^2:

$$\sigma^2 = \sum_{i=1}^{N} (\mu - b_i)$$

La Deviazione Standard σ:

$$\sigma = \sqrt{\sigma^2}$$

5.5.2 OpenChannelAnalysis

Questo tipo di analisi costruisce l'istogramma dei dati ricevuti, siccome si tratta di numeri con la virgola è necessario definire gli intervalli dei valori di dimensione prefissata. Il risultato di questa analisi è un oggetto di tipo **Histogram**, che per motivi di performance viene gestito da un array, dove l'indice corrisponde a un intervallo e il valore corrisponde al numero di volte in cui è stato riscontrato l'intervallo. E' quindi necessaria una mappatura dai valori ricevuti di tipo *float* agli indici del vettore di tipo *int*.
Sia r il range dei valori che va da -r a r e d la dimensione degli intervalli, per la gestione dell'istogramma sarà quindi necessario un vettore di dimensione $\frac{2r}{d}$. Sia v un nuovo valore da aggiungere all'istogramma di tipo float, la mappatura nel vettore sarà fatto secondo la formula: $||\frac{v}{d}|| + \frac{r}{d}$, dove $||..||$ indica il valore arrotondato all'intero più vicino.

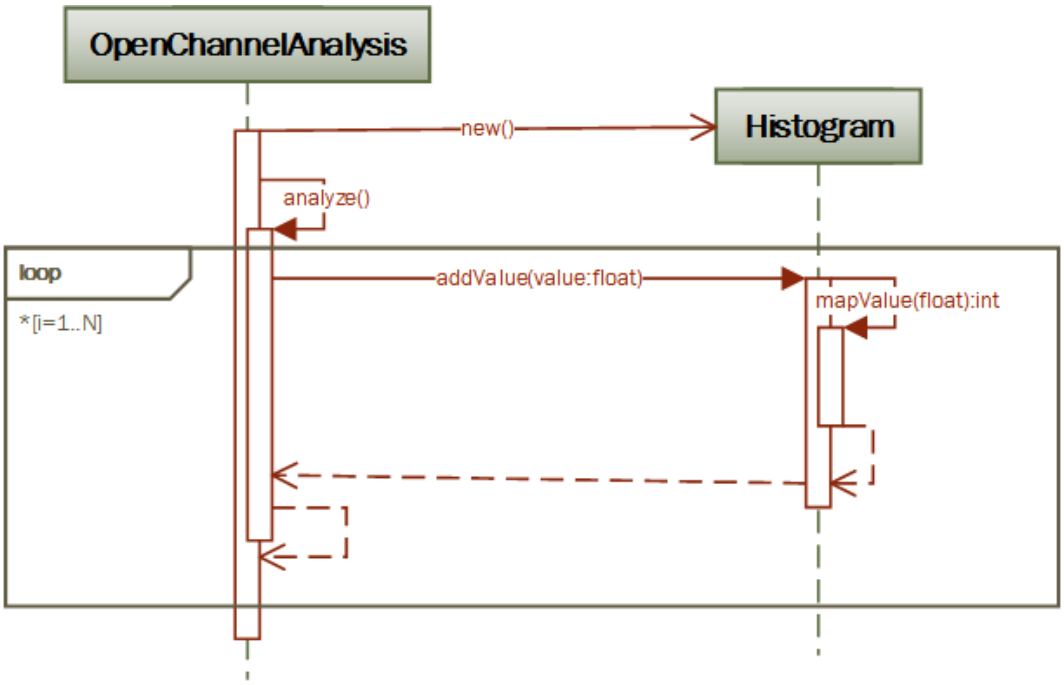


Figura 5.7: Sequenza di istruzioni per l'implementazione dell'analisi

5.5.3 Trasformata di Fourier Discreta (DFT)

Si considera la sequenza x[n], n = 0,1,..,N
posto:

$$W_N = \exp(-j\frac{2\pi}{N})$$

La DFT è definita come:

$$X[k] = DFT\{x[n]\} = \sum_{n=0}^{N-1} x[n]W_N^{kn}$$

La periodicità dei coefficienti $W_N^{kn} = \exp\{-j\frac{2\pi}{N}kn\}$ rende possibile la riduzione del numero di prodotti dell'algoritmo DFT raccogliendo dei termini. L'algoritmo efficiente utilizzato per calcolare la trasformata di Fourier è la **FFT** (**F**ast **F**urier **T**rasform) che permette di calcolare una DFT (in generale, complessa) di N campioni mediante un numero di moltiplicazioni complesse dell'ordine di $Nlog_2(n)$ contro le N^2 della DFT. La FFT è idonea al calcolo della DFT di sequenza la cui lunghezza è una potenza di 2. Come citato, il risultato della FFT è un vettore di numeri complessi si dovrà quindi effettuare un ulteriore operazione per il calcolo del modulo:

$$||X[k]|| = \sqrt[2]{real(X[k])^2 + Im(X[k])^2}$$

con $k = 0..N-1$.

Capitolo 6

Implementazione

In questo capitolo verranno descritte alcune scelte implementative di maggiore rilievo. Come già citato in precedenza il linguaggio di riferimento sarà C++.

6.1 Uso dei Workers

Come indicato nei capitoli precedenti, il progetto è basato sul multi-threading. Verrà introdotto brevemente come sono stati implementati i thread.

6.1.1 Cosa sono i Worker

Un *Worker* è un thread che è sempre attivo e rimane in attesa di nuovi compiti. I worker sono gestiti con i thread di Qt, per mantenere la portabilità verso altri sistemi. Essi permettono il fork di un nuovo thread e di mantenerlo in esecuzione indefinitivamente. Mediante il metodo *moveToThread(QThread)*, presente in tutte le classi che estendono *QObject*, è possibile registrare un oggetto a uno specifico thread in modo tale che tutte le *slot* presenti nella classe vengano chiamati sul thread impostato dal metodo.

6.1.2 Buffered Consumer

Tipicamente, ogni *ContentConsumer* (sezione 4.2.3) è un worker. E' stato quindi necessario estendere il concetto del ContentConsumer e implementarlo come worker. In fase implementativa è stata quindi aggiunta la classe *BufferedConsumer* che estende *ContentConsumer* e viene registrato su un nuovo thread. La classe si limita ad implementare il metodo *receiveData(DataPacket)*, riempire un buffer e chiamare un metodo astratto sul nuovo thread.

```
class BufferedConsumer: public ContentConsumer{
    Q_OBJECT
public:
    BufferedConsumer(unsigned bufferSize);
    ~BufferedConsumer();

public slots:
    virtual void receiveData(DataPacket);
    virtual void receiveBuffer(DataPacket*) = 0;

signals:
    void onBufferFilled(DataPacket*);

protected:
    unsigned getBufferSize();

private:
    unsigned buffersize;
    unsigned bufferedData;
    ConsumerThread* thread;
    DataPacket* buffer;
};
```

Listato 6.1: Intestazione della classe

Questa classe permette l'utilizzo trasparente dei worker, dato che chi andrà ad estendere questa classe non deve gestire i thread, non incorre nemmeno in corse critiche in quanto non ci sono dati condivisi con altri thread.

```
BufferedConsumer::BufferedConsumer(unsigned bufferSize){
   //crea il nuovo thread
    this->thread = new QThread();
    this->moveToThread(this->thread);
    this->thread->start();
    //gestisce il buffer
    this->buffersize = bufferSize;
    buffer = new DataPacket[buffersize];
    bufferedData = 0;
    //collega la signal alla slot, le chiamate verranno eseguite nel
       "thread"
    connect(this, SIGNAL(onBufferFilled(DataPacket*)), this,
       SLOT(receiveBuffer(DataPacket*)));
}


BufferedConsumer::~BufferedConsumer(){
    //termina il thread in esecuzione
    this->thread->quit();
    this->thread->wait();
    delete this->thread;
}


void BufferedConsumer::receiveData(DataPacket d){
    buffer[bufferedData++] = d;

    if(bufferedData == buffersize){
        //emmette la signal, verra' eseguita la slot connessa ad essa
        //nel thread in attesa
        emit onBufferFilled(buffer);
        bufferedData = 0;
        buffer = new DataPacket[buffersize];
    }
}

unsigned BufferedConsumer::getBufferSize(){
    return buffersize;
}
```

Listato 6.2: Implementazione della classe

6.2 Bit Rate

Siccome non è possibile ottenere dal dispositivo la frequenza effettiva con cui esso sta inviando dati è stato opportuno creare un *ContentConsumer* apposito per la stima della frequenza.

```
[...]

void BitRateCalculator::receiveData(DataPacket d){
    dataCount++;

   //entra nell'if ogni secondo
    if((int)(clock()-prevTime) >= seconds){
      //calcolo della frequenza
        rate = actualBitRate * //numero campioni
            (d.getChannelNumber() //numero di canali
            + 1) // l'uno indica la tensione
            * sizeof(float) * 8 // e' il numero di bit del tipo float
            / 1024.0; // converte in Kb/s

      //seleziona l'unita' di misura
        QString rateLabel = " Kb/s";
        if(rate > 1024){
            rate /= 1024.0;
            rateLabel = " Mb/s";
        }
      //notifica la frequenza
        emit onBitrateCalculated( QString::number(rate) + rateLabel);
        prevTime=clock();
      //resetta il conteggio
        dataCount = 0;
    }
}

[...]
```

Listato 6.3: Implementazione della classe BitRateCalculator

Capitolo 7

Testing

In questo capitolo verranno mostrati vari test effettuati con i dispositivi oppure con dispositivi simulati. I test verranno eseguiti variando il carico di lavoro del computer in uso.

Computer In uso

Il PC su cui è stato effettuato il test è un pc portatile Asus prodotto nell'anno 2015, con processore Intel core i7-5500U, 2.2 to 3.0 GHz, con 12 GB di memoria RAM DDR3, con sistema operativo Microsoft Windows 10, connesso ad alimentazione elettrica e con il profilo di risparmio energia impostato a - bilanciato. Questa configurazione rispetta la richiesta dell'azienda di fornire buone prestazione su PC di fascia media di mercato.

Verranno eseguiti test su una macchina virtuale con 8.1 con un dual-core e 4 GB di memoria RAM per vedere il comportamento su macchine con prestazioni inferiori.

7.1 eONE

Il dispositivo *eONE* effettua la lettura su un solo canale. I dati sono di tipo *float*, con dimensione di quattro byte per dato. Ogni campione contiene due dati: quello del campione in corrente e la tensione applicata. Nella versione *HS*, la versione high speed del dispositivo, viene raggiunta una frequenza di campionamento pari a 200 kHz, con una bit-rate di 12.20 Mb/s.

7.1.1 Situazione di base

Per Situazione di base si intende il caso in cui il computer dove l'applicazione sta girando non mantiene in contemporanea altri processi che posso

occupare la CPU, con un utilizzo di CPU inferiore al 10% e un consumo della RAM inferiore al 30%. Questa situazione è attesa per l'utilizzo dall'utente finale.

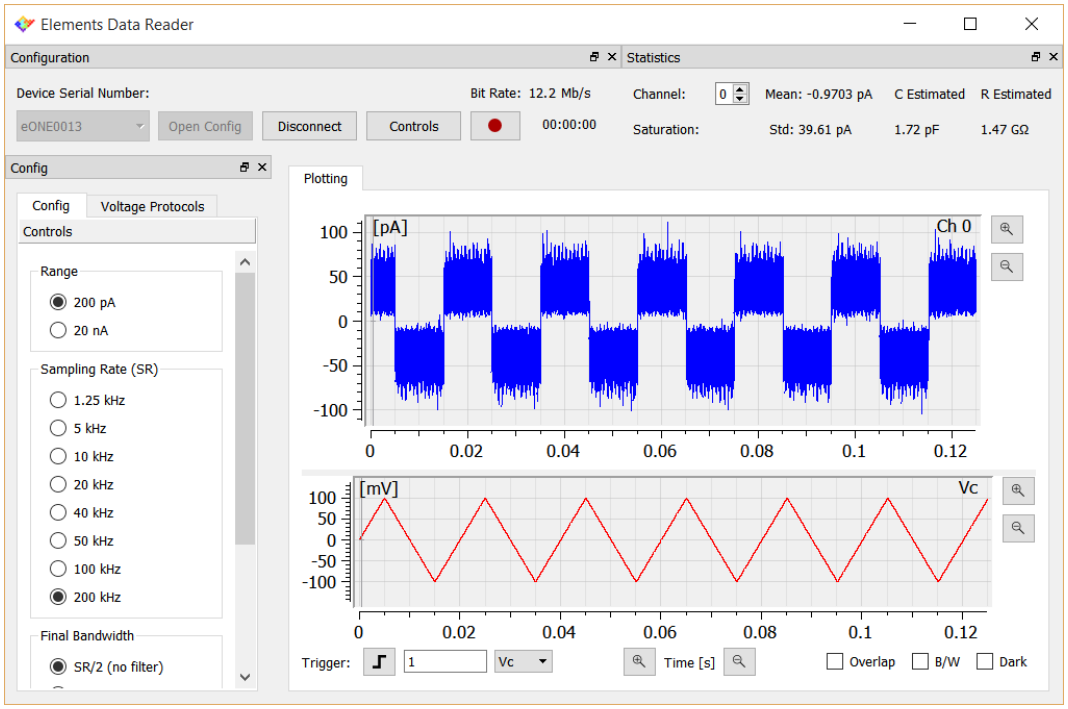


Figura 7.1: Screenshot applicazione: acquisizione da eONE in condizioni di base

Scenari di testing

Verranno considerati diversi scenari di utilizzo dell'applicazione, considerando sempre il caso peggiore, cioè mantenendo la frequenza di campionamento pari a 200kHz.

Acquisendo dati

In questo scenario si rappresenta lo stato dell'applicazione in cui si effettua l'acquisizione dei dati, la visualizzazione di essi e il calcolo delle statistiche, come mostrato in figura 7.1, la bit-rate mantenuta è circa uguale a quella richiesta, ossia 12.2 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione EDR si attesta intorno al 15%, con un'occupazione della memoria RAM intorno ai 90MB.

Acquisendo e Memorizzando dati

In questo scenario si rappresenta lo stato dell'applicazione che mantiene le funzionalità dello scenario precedente e contemporaneamente effettua la memorizzazione dei dati su disco. La bit-rate mantenuta è circa uguale a quella richiesta, ossia 12.2 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione *EDR* si attesta intorno al 17%, con un'occupazione della memoria RAM intorno ai 90 MB e una scrittura su disco di 1.5 MB/s. Confrontando con lo scenario si mostra che la memorizzazione non influisce sulle prestazioni.

Acquisendo, Memorizzando e Analizzando dati

In questo scenario si rappresenta lo stato dell'applicazione che mantiene le funzionalità dello scenario precedente e contemporaneamente effettua il calcolo dell'istogramma con un intervallo pari a 0.01. La bit-rate mantenuta è circa uguale a quella richiesta, ossia 12.2 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione *EDR* si attesta intorno al 21%, con un'occupazione della memoria RAM intorno ai 95 MB e una scrittura su disco di 1.5 MB/s. Avviando inoltre un ulteriore analisi, cioè il calcolo della trasformata di Fourier su 32768 campioni, l'utilizzo della CPU sale a 24% e la RAM a 110 MB. Confrontando con lo scenario si mostra che le analisi su dati tendono ad essere operazioni pesanti, influiscono quindi significativamente sulle prestazioni.

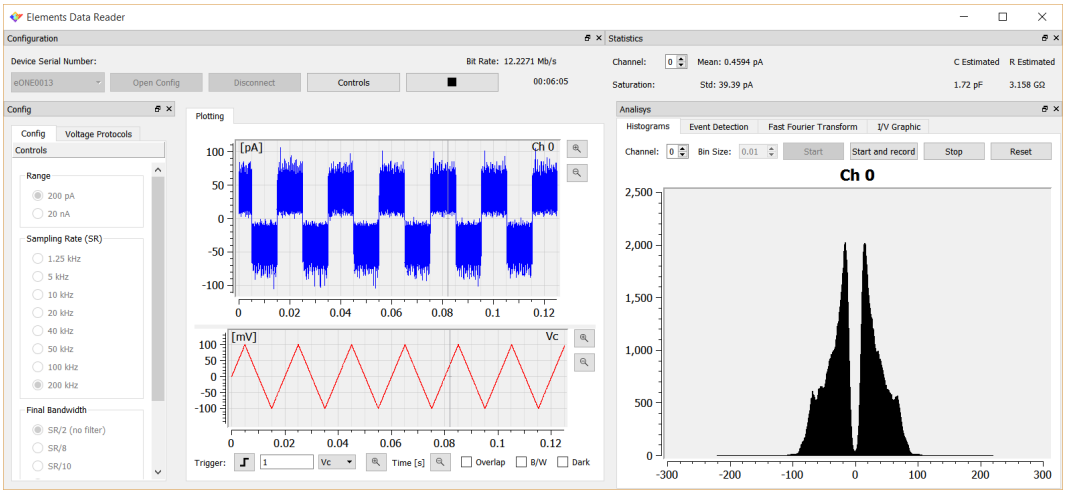


Figura 7.2: Screenshot applicazione: acquisizione, memorizzando e analizzando dati da eONE

7.1.2 Situazione di alto carico di lavoro

In questa situazione si testa il software nel caso in cui il computer sul quale gira l'applicazione sia presente un alto carico di lavoro, si simula questa situazione mantenendo aperti diversi software che richiedo un uso continuo della CPU, come la riproduzione di video, macchine virtuali ecc... In questa circostanza si ha un consumo della CPU superiore a 40% e 7 GB di RAM occupati. Le prestazioni, come atteso, sono rimaste invariate e anche la bit-rate di acquisizione, non perdendo quindi dati.

7.1.3 Testing su macchina virtuale

Il test su macchina virtuale è stato eseguito per verificare il comportamento del software su sistemi con prestazioni inferiori. Come indicato sopra la macchina virtuale presenta le seguenti caratteristiche: processore i7 dual core con 4GB di RAM e con sistema operativo a 64bit Windows 8.1.

Acquisendo dati

In questo scenario si rappresenta lo stato dell'applicazione in cui si effettua l'acquisizione dei dati, la visualizzazione di essi e il calcolo delle statistiche, come mostrato in figura 7.1, la bit-rate mantenuta è circa uguale a quella richiesta, ossia 12.2 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione EDR si attesta intorno al 30%, con un'occupazione della memoria RAM intorno a 85MB.

Acquisendo e Memorizzando dati

In questo scenario si rappresenta lo stato dell'applicazione che mantiene le funzionalità dello scenario precedente e contemporaneamente effettua la memorizzazione dei dati su disco. La bit-rate mantenuta è circa uguale a quella richiesta, ossia 12.2 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione *EDR* si attesta intorno al 33%, con un'occupazione della memoria RAM intorno ai 90 MB e una scrittura su disco di 1.5 MB/s. Confrontando con lo scenario precedente si mostra che la memorizzazione non influisce sulle prestazioni.

Acquisendo, Memorizzando e Analizzando dati

In questo scenario si rappresenta lo stato dell'applicazione che mantiene le funzionalità dello scenario precedente e inoltre effettua il calcolo dell'istogramma con un intervallo pari a 0.01. La bit-rate mantenuta è circa uguale a quella

richiesta, ossia 12.2 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione *EDR* si attesta intorno al 40%, con un'occupazione della memoria RAM intorno ai 91 MB e una scrittura su disco di 1.5 MB/s. Avviando inoltre un ulteriore analisi, cioè il calcolo della trasformata di Fourier su 32768 campioni, l'utilizzo della CPU sale a 44% e la RAM a 110 MB. Confrontando con gli scenari precedenti si mostra che le analisi su dati tendono ad essere operazioni pesanti, anche in questo caso influiscono significativamente sulle prestazioni.

Alto carico di lavoro

In situazione di alto carico di lavoro il software non riesce a gestire tutti i dati, si riscontra quindi un calo della bit-rate e una conseguente perdita dei dati.

7.2 eFOUR

Il dispositivo *eFOUR* effettua la lettura su quattro canali distinti. I dati sono di tipo *float*, con dimensione di quattro byte per dato. Ogni campione contiene cinque dati: quattro campioni in corrente e la tensione applicata. Nella versione *HS*, la versione estesa del dispositivo, raggiunge una frequenza di campionamento massima a 200 kHz, con una bit-rate di 30.5 Mb/s.

7.2.1 Situazione di base

Acquisendo dati

In questo scenario si rappresenta lo stato dell'applicazione in cui si effettua l'acquisizione dei dati, la visualizzazione di essi e il calcolo delle statistiche, come mostrato in figura 7.1, la bit-rate mantenuta è circa uguale a quella richiesta, ossia 30.5 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione EDR si attesta intorno al 19%, con un'occupazione della memoria RAM intorno ai 80MB.

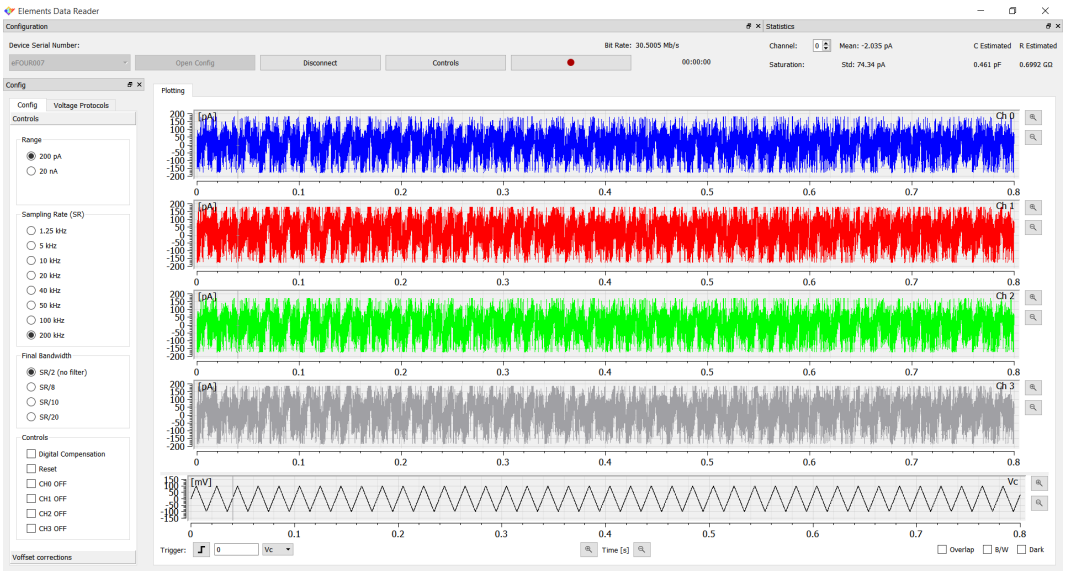


Figura 7.3: Screenshot applicazione: acquisizione da eFOUR in condizioni di base

Per situazione di base si intende la stessa descritta in precedenza.

Acquisendo e Memorizzando dati

In questo scenario si rappresenta lo stato dell'applicazione che mantiene le funzionalità dello scenario precedente e inoltre effettua la memorizzazione dei dati su disco. La bit-rate mantenuta è circa uguale a quella richiesta, ossia 30.5 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione *EDR* si attesta intorno al 20%, con un'occupazione della memoria RAM intorno a 85 MB e una scrittura su disco di 3.8 MB/s. Confrontando con lo scenario si mostra che la memorizzazione non influisce sulle prestazioni.

Acquisendo, Memorizzando e Analizzando dati

In questo scenario si rappresenta lo stato dell'applicazione che mantiene le funzionalità dello scenario precedente e inoltre effettua il calcolo dell'istogramma con un intervallo pari a 0.01. La bit-rate mantenuta è circa uguale a quella richiesta, ossia 30.5 Mb/s, rispondendo completamente ai requisiti. L'utilizzo percentuale della CPU dell'applicazione *EDR* si attesta intorno al 27%, con un'occupazione della memoria RAM intorno ai 90 MB e una scrittura su disco di 3.8 MB/s. Avviando inoltre un ulteriore analisi, cioè il calcolo della trasformata di Fourier su 32768 campioni, l'utilizzo della CPU sale a 29% e

la RAM a 110 MB. Confrontando con lo scenario si mostra che le analisi su dati tendono ad essere operazioni pesanti, influiscono quindi significativamente sulle prestazioni.

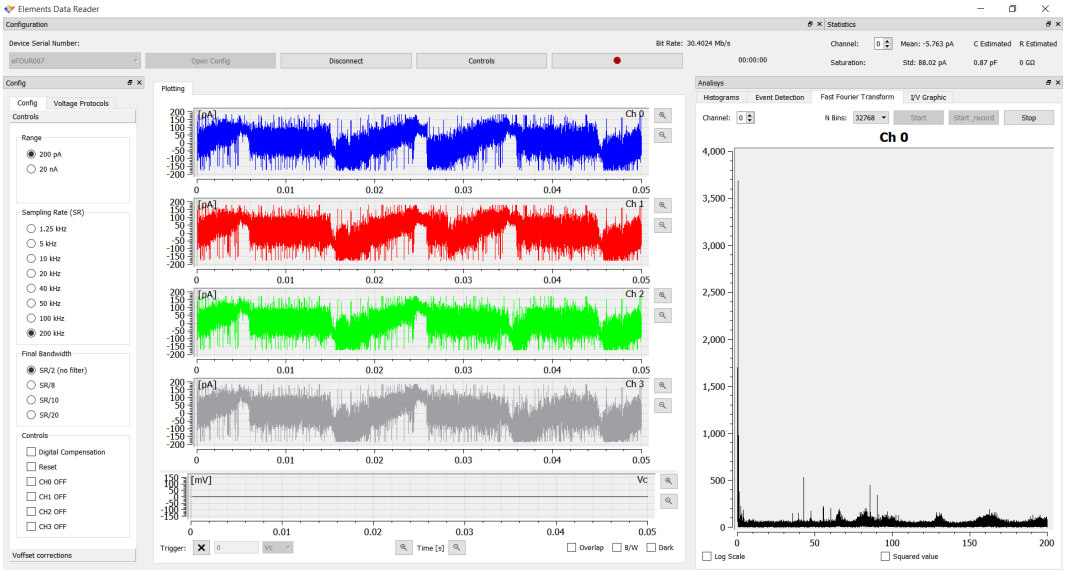


Figura 7.4: Screenshot applicazione: acquisendo, memorizzando e analizzando dati da eFOUR

7.3 eSIXTEEN

Sono Stati effettuati anche alcuni test con il dispositivo ancora in fase di prototipazione a 16 canali. Si prevede che alla massima frequenza, 200 kHz, l'eSIXTEEN mantenga una bit-rate pari a 103 Mbit/s.

La bit-rate mantenuta è intorno a 103 Mbit/s nonostante stia memorizzando, effettuando l'istogramma di tutti e sedici i canali e calcolando la FFT su 32768 campioni. La CPU utilizzata è pari al 50%, con 110 MB di RAM utilizzata e una scrittura su disco di 12 MB/s.

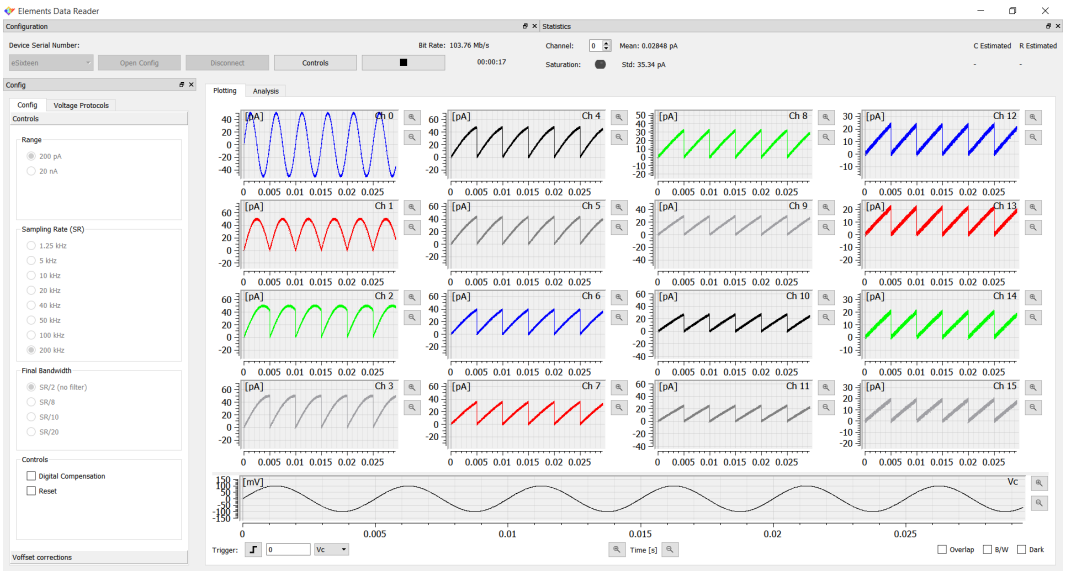


Figura 7.5: Screenshot applicazione: acquisendo, memorizzando e analizzando dati da eSIXTEEN simulato

7.4 Tabella riassuntiva

Device	SO	Freq.	M*	A**	Bitrate	CPU	RAM
eONE	Win 10	200 kHz	NO	NO	12.20 Mbit/s	15 %	90 MB
eONE	Win 10	200 kHz	SI	NO	12.20 Mbit/s	17 %	90 MB
eONE	Win 10	200 kHz	SI	SI	12.20 Mbit/s	24 %	110 MB
eONE	Win 8.1	200 kHz	NO	NO	12.20 Mbit/s	30 %	85 MB
eONE	Win 8.1	200 kHz	SI	NO	12.20 Mbit/s	33 %	90 MB
eONE	Win 8.1	200 kHz	SI	SI	12.20 Mbit/s	44 %	110 MB
eFOUR	Win 10	200 kHz	NO	NO	30.50 Mbit/s	19 %	80 MB
eFOUR	Win 10	200 kHz	SI	NO	30.50 Mbit/s	20 %	85 MB
eFOUR	Win 10	200 kHz	SI	SI	30.50 Mbit/s	29 %	110 MB
e16	Win 10	200 kHz	SI	SI	103 Mbit/s	50 %	110 MB

* = Memorizzando, ** = Analizzando, CPU, Bitrate e RAM si intendono come valori medi

7.5 Affidabilità del software

Un ulteriore test è stato effettuato per verificare l'affidabilità dell'applicazione. Il software è stato mantenuto attivo per oltre 2 ore senza presentare

problemi di alcun tipo, rendendo superfluo il proseguimento del test. In questo tempo il software mostrava i dati in real-time, salvava su disco ed effettuava le analisi descritte precedentemente. Questo test garantisce l'affidabilità del software anche in lunghe sessioni di registrazione dati. Il test è stato effettuato anche dai tecnici di Elements, su diversi sistemi operativi e con macchine diverse, riscontrando esiti positivi.

7.6 Consistenza dei Dati e Interoperabilità

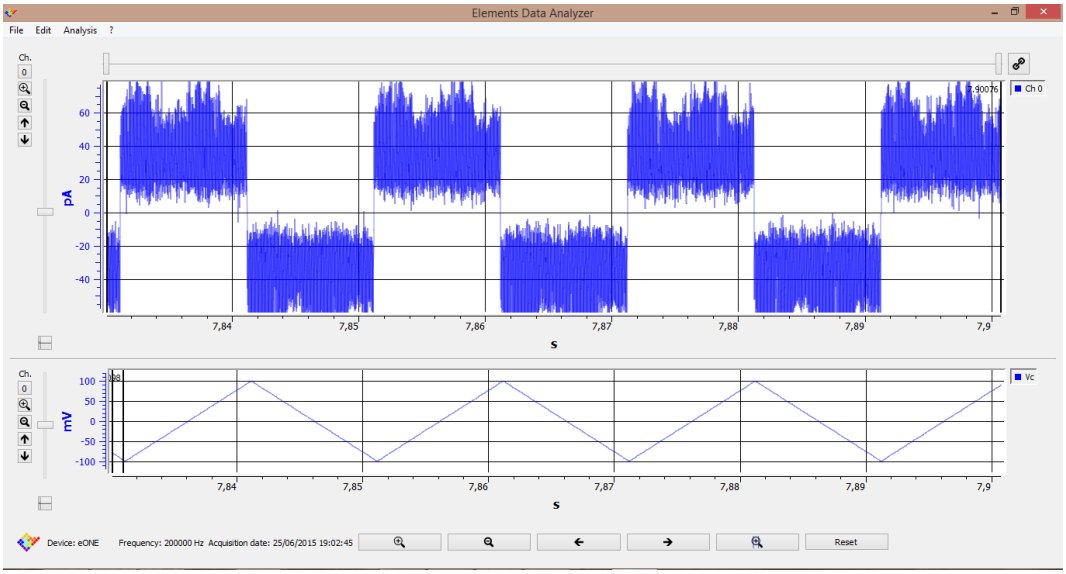


Figura 7.6: Elements Data Analyzer con il risultato di una registrazione su eONE

Avendo come modello la versione precedente del software, si è cercato di verificare la coerenza dei dati. Purtroppo però non è stato possibile verificare contemporaneamente il comportamento delle due applicazioni leggendo dallo stesso dispositivo, ma effettuando registrazioni dei dati, si è potuto notare che i dati letti, per valori e forma d'onda, risultano assolutamente corretti. Inoltre è stato utilizzata come verifica della consistenza il software *Elements Data Analyzer*, che effettua analisi in post-processing. L'utilizzo del software esterno verifica inoltre l'interoperabilità dei dati salvati.

7.7 Soddisfazione dei requisiti

I requisiti richiesti inizialmente sono stati tutti rispettati, presentando tutte le funzionalità richieste mantenendo buone prestazioni, come dimostrato dai vari test. L'applicazione sviluppata mantiene un'interfaccia utente fluida nonostante la grande mole di operazioni da svolgere periodicamente. Mediante i vari test e la progettazione del software si è dimostrato di aver soddisfatto tutte le specifiche del software richieste.

Conclusioni

Il progetto di tesi conseguito è stato molto formativo, in quanto ho avuto la possibilità di lavorare all'interno di un'azienda, potendo così notare l'ambiente lavorativo. Il progetto si è svolto con un una grande cooperazione di gruppo, collaborando sulle componenti in comune e risolvendo le problematiche più importanti che si riscontravano nel cammino dello sviluppo del software.

L'appoggio e la fiducia che il team dell'azienda ci ha concesso sono stati di fondamentale importanza per lo sviluppo del software, illustrandoci dettagli tecnici necessari per la gestione dei dati, valutando le nostre idee e concedendoci la massima libertà in tutte le fasi di sviluppo del software.

Il processo di sviluppo si è svolto come previsto nei primi colloqui, procedendo nello sviluppo dei moduli e analizzando e adattando il loro comportamento. Al termine del processo il software sviluppato soddisfa i requisiti richiesti, ma è ancora in fase di beta testing in quanto non è disponibile a tutti gli utenti ma solo alcuni clienti dell'azienda che effettuano prove e forniscono feedback necessari per il miglioramento del software.

I due punti di forza del software finale sono sicuramente le prestazioni e la scalabilità, poiché sono le caratteristiche su cui, entrambi gli elementi del team, ci siamo concentrati maggiormente in fase di analisi e progettazione. In effetti le prestazioni mostrate nei test sono completamente soddisfacenti, superando addirittura le aspettative che avevamo.

Le conoscenze acquisite durante le lezioni universitarie hanno svolto un ruolo importante per lo sviluppo del sistema, in particolare *Sistemi Operativi* per la gestione del multithreading e l'utilizzo di strutture dati apposite, *Algoritmi* per i calcoli dei vari costi computazionali e *Programmazione a Oggetti* per il paradigma a oggetti su cui si basa l'intero software e tecniche di programmazione avanzate come l'utilizzo dei vari *design pattern*.

In confronto alla vecchia vecchia versione, il software presenta un interfaccia utente più intuitiva mantenendo tutte le funzionalità ma automatizzando alcune di esse, è più fluida anche campionando ad alte frequenze. Presenta inoltre una struttura modulare che permette l'ampliamento di nuove funzionalità modificando il minimo indispensabile il codice sorgente.

Sviluppi Futuri

Il software progettato è una versione base, del quale è previsto un ampliamento continuo. Una dei possibili ampliamenti riguarda il modulo di visualizzazione che deve prevedere parallelismi elevati con possibilità di selezionare o deselezionare i canali da visualizzare, adattando l'interfaccia utente rispetto ai feedback degli utenti finali. Il software è in fase di beta testing dove aziende collaboratici di Elements s.r.l. usano e denotano eventuali BUG. Il software dovrà gestire anche nuovi dispositivi prodotti da Elements s.r.l. che potranno utilizzare moduli hardwaredi comunicazione diversi dal FTDI. Il software sviluppato è stato progettato in modo tale da prevedere nuovi dispositivi e aggiungerli in modo semplice. Sono previsti ulteriori tipi di analisi in real-time da aggiungere in quanto le analisi rendono il software all'avanguardia nel settore. E' previsto una collaborazione con l'azienda Elements per l'ampliamento del software e sviluppo di nuovi progetti.

Ringraziamenti

Ringrazio innanzitutto i tecnici di Elements s.r.l, Federico, Marco e Michele per la bella opportunità che mi hanno offerto e per tutto l'aiuto necessario per lo svolgimento del software. Un particolare grazie va al collega Matteo Marra che si è impegnato al massimo per la riuscita del progetto in tempi ragionevoli. Ringrazio inoltre il prof. Viroli per gli ottimi consigli e tutti gli amici e parenti che mi hanno sostenuto nel mio percorso di studi.

Elenco delle figure

Bibliografia

[1] Thei Federico, Phd thesis: *A HYBRID TECHNOLOGY FOR PARALLEL RECORDING OF SINGLE ION CHANNELS, Universit'a di Bologna, 2011.*

[2] Elements `http://elements-ic.com/products/eone/`

[3] hwupgrade `http://www.hwupgrade.it/articoli/storage/2630/kingston-hyperx-max-nuova-interfaccia-nuovo-disco_2.html`

[4] QT `https://wiki.qt.io/`

[5] Qwt `http://qwt.sourceforge.net/`

[6] Wikipedia `https://www.wikipedia.org/`

[7] Axon Instruments Inc. *ABF Help*