Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA

Studio del sistema di controllo di un rivelatore di ioni, realizzato in ambiente EPICS, per il progetto SPES

Tesi di Laurea Specialistica

Laureando: Conforto Nicola
Relatore: Prof. F. Bombi
Correlatori: Dott. A. Andrighetto, Dott. G. Bassato
Corso di Laurea Specialistica in Ingegneria Informatica

Dipartimento di Ingegneria dell'Informazione
Anno Accademico 2009-2010

Indice

Introduzione

Il presente lavoro di tesi è rivolto a fornire strumenti software per la misurazione di proprietà fisiche di un fascio di ioni. In particolare, il sistema di rivelazione del Front End del progetto SPES è costituito da un blocco diagnostico composto da un misuratore di corrente (chiamato coppa di Faraday), un profilatore di fascio (beam profiler), ed un misuratore di emittanza.

Per comprendere l'utilità di un sistema di diagnostica del fascio di questo tipo, è necessario capirne innanzitutto la collocazione all'interno dell'intero progetto e lo scopo del progetto stesso.

Dall'inizio del ventesimo secolo la fisica nucleare esplora i confini della natura per produrre materia mai osservata prima in laboratorio; le ricerche di base e le complesse tecnologie appositamente create, hanno spesso portato alla nascita di un gran numero di applicazioni nel campo della medicina, dell'industria e della fisica applicata, arrivando in molti casi ad influenzare usi e costumi della società.

Nel corso degli anni l'Europa ha assunto la leadership nel campo della ricerca nucleare e sta pianificando la costruzione di una nuova generazione di impianti per la produzione di fasci radioattivi, con lo scopo di esplorare la materia esotica e fornire un valido strumento per applicazioni di tipo medico ed industriale. All'interno del programma partecipa attivamente, con il progetto SPES (Selective Production of Exotic Species), anche l'Istituto Nazionale di Fisica Nucleare (INFN) ed i Laboratori Nazionali di Legnaro. Tale progetto prevede la costruzione di un impianto, basato sul metodo ISOL (Isotope Separation On Line), per la produzione di fasci di ioni radioattivi ricchi di neutroni (neutron rich). La produzione di tali fasci richiede l'impiego di tecnologie estremamente complesse ed innovative. In questo versante, il progetto SPES prevede, come verrà descritto più avanti, la generazione e l'estrazione di ioni radioattivi mediante il bombardamento di un bersaglio (target) innovativo, sviluppato interamente dal gruppo di ricerca del progetto.

La complessità di un impianto come quello del progetto SPES si ripercuote sulla complessità del sistema di controllo da sviluppare. Si può dedurre come sia necessario avere un'organizzazione delle informazioni ordinata nei vari livelli della struttura, in modo da poter gestire e coordinare efficacemente ogni sottosistema che la costituisce. Una delle principali problematiche nella progettazione di questo sistema di controllo si è rivelata essere la necessità di integrare soluzioni hardware e software di diversa tipologia; l'adozione di un sistema di controllo unico, capace di implementare uno standard di comunicazione comune, diventa quindi un passo fondamentale per la realizzazione del progetto. Tra le varie offerte disponibili sul mercato spicca il software EPICS, un software di controllo open source sviluppato all'interno della comunità dei laboratori di fisica nucleare; esso fornisce strumenti ed applicazioni per creare un sistema di controllo distribuito indipendente dall'hardware, ed implementare una infrastruttura di comunicazione in grado di realizzare una memoria dati distribuita tra tutti i nodi della rete di controllo.

Attualmente, presso i Laboratori Nazionali di Legnaro, è in fase di test il Front End *offline* del progetto SPES, al fine di effettuare prove di ionizzazione e trasporto di ioni stabili. Per verificare il funzionamento dell'apparato di produzione di ioni e del sistema di controllo, risulta necessario avere un sistema di diagnostica che sia in grado di rilevare determinate proprietà fisiche del fascio di ioni in uscita dal sistema Front End.

Il presente lavoro di tesi, come già accennato, è quindi rivolto a fornire strumenti software per la misurazione di proprietà fisiche di un fascio di ioni, utilizzando tre diversi strumenti di diagnostica.
Oltre allo scopo ed al funzionamento meccanico di questi strumenti, nel presente lavoro viene descritto in dettaglio anche il funzionamento dell'intero sistema di acquisizione, basato su un calcolatore con bus VME e sistema operativo VxWorks.
Parallelamente allo sviluppo del software di acquisizione e di controllo degli strumenti, sono state sviluppate anche le interfacce per gli operatori del sistema.

La tesi si compone di sei capitoli, il cui contenuto viene riassunto qui di seguito.

Il **primo capitolo** introduce alcune applicazioni dei fasci di ioni radioattivi nei campi di interesse della fisica nucleare, della fisica dello stato solido, dell'astrofisica e della medicina. Vengono descritti due metodi di produzione di tali fasci (FRS ed ISOL) e si illustrano le principali caratteristiche del progetto SPES, in fase di sviluppo presso i Laboratori Nazionali di Legnaro.
Il **secondo capitolo** invece descrive la strumentazione del Front End del progetto SPES; vengono quindi trattati il funzionamento del target, del sistema di estrazione, dei metodi di ionizzazione ed anche del sistema di trasporto del fascio. Viene poi descritto in dettaglio il funzionamento del sistema di controllo dell'intero sistema e le motivazioni che hanno portato alla scelta del software EPICS.

Il **terzo capitolo** descrive il funzionamento del software EPICS, soffermandosi in particolare sul funzionamento del protocollo *Channel Access* e sui meccanismi base di funzionamento degli *IOC* (Input Output Controller), dei database, e dei *Device Driver/Device Support*. Infine, un accenno ai principali strumenti per lo sviluppo dei database e delle interfacce per operatori.
Il **quarto capitolo** è dedicato alle misure e al relativo metodo di acquisizione. Viene quindi data una descrizione tecnica delle proprietà fisiche da misurare, lo scopo della misurazione, ed il funzionamento dei vari sistemi di rivelazione. Il particolare, viene descritta l'acquisizione della corrente di fascio tramite una coppa di Faraday, l'acquisizione del profilo del fascio tramite un profilatore, e la misurazione dell'emittanza con un emittance meter di tipo fessura-griglia (slit-grid).
Nel **quinto capitolo** viene invece descritto l'hardware ed il software su cui è realizzato il sistema di diagnostica. Viene descritto il funzionamento del bus VME, elencate le schede elettroniche utilizzate (assieme alle relative caratteristiche), e descritti i meccanismi basilari su cui si basa il funzionamento del sistema operativo real-time VxWorks, con particolare attenzione alle politiche di gestione e comunicazione fra task.
Il **sesto capitolo** è infine dedicato alle applicazioni create per il sistema di diagnostica. Vengono quindi descritte le operazioni preliminari per la creazione dell'applicazione, la configurazione del database, lo script per l'inizializzazione del sistema, il funzionamento dei programmi che gestiscono l'acquisizione ed il calcolo dell'emittanza, e la configurazione delle interfacce create con il software *Control System Studio*.

Capitolo 1 – Introduzione al Progetto SPES

La materia, definita come tutto ciò che ha una massa e che occupa spazio, é costituita da atomi. Essi rappresentano la più piccola unità di un elemento chimico, rompendo la quale l'elemento non è più lo stesso, perde la sua identità. L'ossigeno, ad esempio, è formato da molecole costituite dal legame tra due atomi di ossigeno. Se rompiamo la molecola separando i due atomi, possiamo ancora parlare di ossigeno, ma se dividiamo l'atomo otteniamo pezzi di qualcosa con caratteristiche chimiche totalmente differenti. A loro volta gli atomi presentano un nucleo estremamente piccolo (dimensione approssimativa: 10^{-15} metri) di carica positiva, circondato da una nuvola di elettroni di carica negativa (fig. 1.1). Gli elettroni sono particelle fondamentali ed indivisibili, che orbitano a distanze relativamente grandi, considerando la dimensione del nucleo. Con le dovute proporzioni, se il nucleo fosse grande come una pallina da ping pong, essi dovrebbero trovarsi a circa un chilometro di distanza, con solo il vuoto nel mezzo. [1, 2]

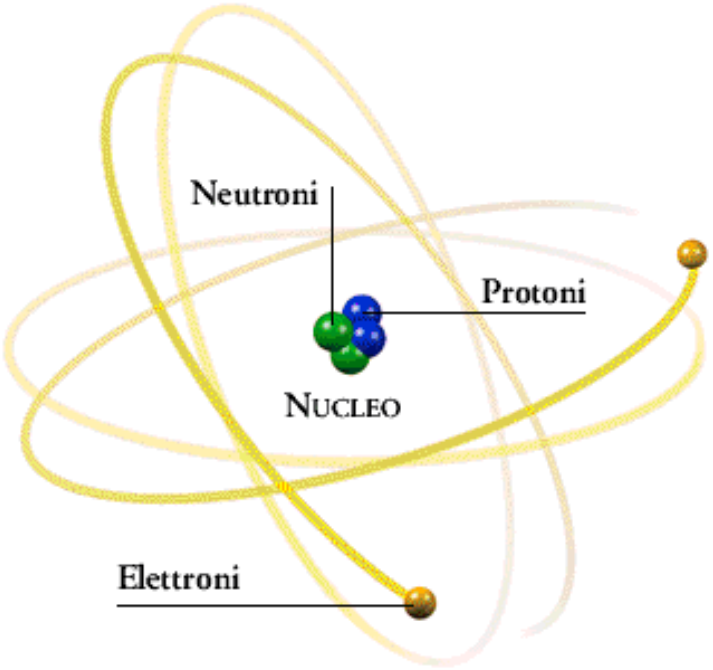

Figura 1.1 – Formazione dell'atomo

All'interno dell'atomo, il nucleo é costituito da protoni carichi positivamente e da neutroni privi di carica, per questo definiti neutri, entrambi denominati nucleoni ed aventi una massa circa 1800 volte più grande degli elettroni. Negli atomi il numero di protoni é uguale al numero di elettroni (carichi negativamente), in modo tale che l'atomo risulti elettricamente neutro.

La maggior parte della massa dell'atomo (più del 99,9%) si concentra proprio nel nucleo, il quale ne occupa la parte centrale e determina, attraverso la sua carica elettrica, la natura chimica dell'elemento. Le particelle del nucleo sono tenute insieme da una intensa forza detta *forza nucleare forte*, che fa parte delle quattro forze fondamentali insieme alla *forza nucleare debole*, alla gravità e all'elettromagnetismo. Non tutti i nuclei però sono legati allo stesso modo. L'interazione favorisce situazioni in cui il numero di neutroni e protoni è uguale, mentre le forze coulombiane repulsive favoriscono le formazione di nuclei in cui il numero di neutroni è maggiore. Ne risulta che i sistemi stabili giacciono in una zona della carta dei nuclidi (fig. 1.2) detta *valle di stabilità*, che, al crescere del numero di massa A, inizia sulla diagonale N=Z per poi curvare verso valori di N maggiori di Z (dove N indica il numero di neutroni e Z indica il numero di protoni nel nucleo).

In questa carta, ogni nuclide messo in evidenza sperimentalmente, è rappresentato da un quadrato che contiene il simbolo dell'elemento ed il numero di nucleoni A. Nella carta, i nuclidi sono disposti in modo che il numero di protoni Z sia indicato in ordinata ed il numero di neutroni N = A-Z in ascissa.

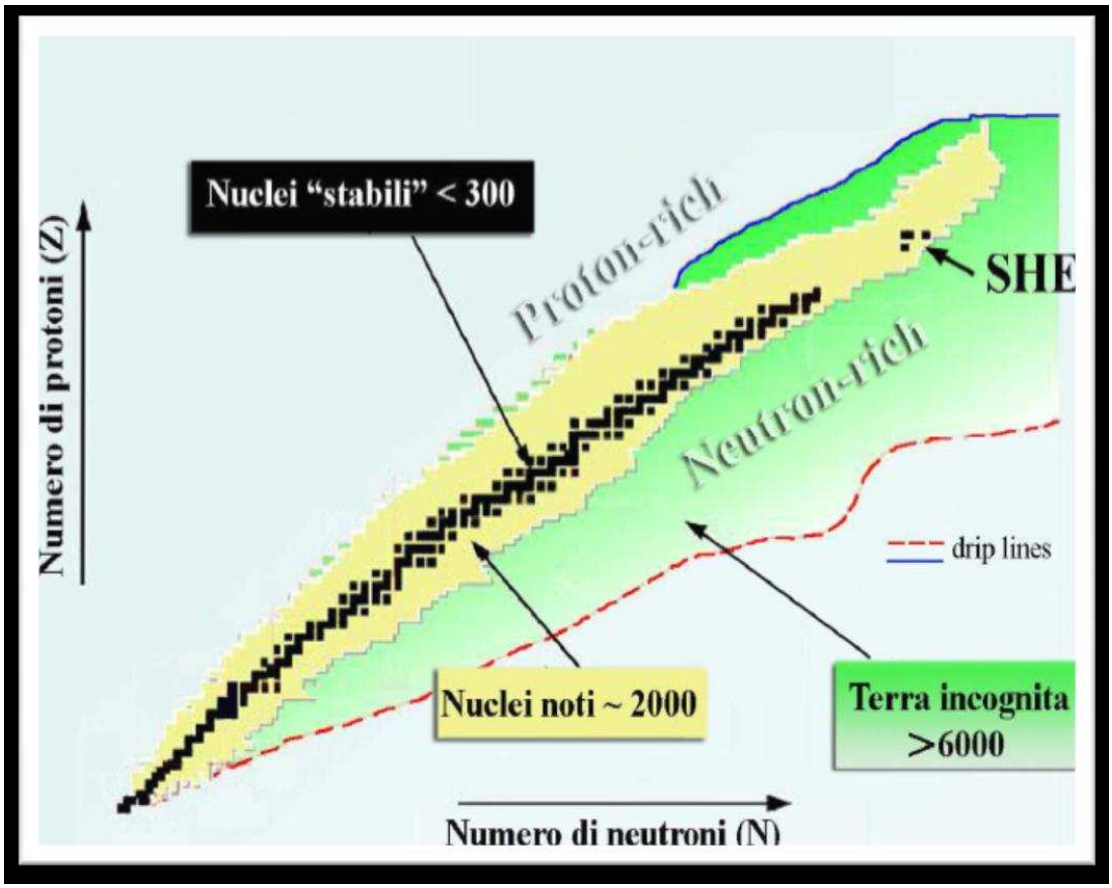

Figura 1.2 – La carta dei nuclidi

Un elemento chimico, oltre al numero fisso di protoni che lo caratterizza, può avere un numero variabile di neutroni: in tal caso si identificano diversi isotopi di uno stesso elemento. Gli isotopi presenti in natura sono quasi tutti stabili. Tuttavia, alcuni isotopi naturali (come ad esempio ^{238}U), e quasi tutti gli isotopi artificiali, presentano nuclei instabili, a causa di un eccesso di protoni e/o di neutroni. Tale instabilità provoca la trasformazione spontanea in altri nuclei, e questa trasformazione si accompagna con l'emissione di radiazioni ionizzanti per cui essi sono chiamati isotopi radioattivi, o anche radioisotopi, o anche radionuclidi.

La trasformazione di un atomo radioattivo porta alla produzione di un altro atomo, che può essere anch'esso radioattivo oppure stabile. Essa è chiamata disintegrazione o decadimento ed è accompagnato dall'emissione di particelle (alfa, beta, neutrini...) e/o raggi gamma. A seconda del tipo di atomo e dell'isotopo, si posso avere diversi tipi di decadimento; alcuni esempi sono α, β^+ e β^-.

I nuclei instabili vengono comunemente chiamati *esotici* ed al momento circa 2000 di essi sono stati prodotti e caratterizzati nei laboratori di ricerca di tutto il mondo. Calcoli teorici prevedono tuttavia l'esistenza di un numero di atomi esotici molto più elevato (più di 6000), cosicché è possibile che un gran numero di essi sia presente nella così detta *terra incognita* che comprende la regione ricca di neutroni (n-rich) e quella dei nuclei superpesanti (SHE,

Super Heavy Elements). La stabilità di questi nuclei è una questione fondamentale nella fisica nucleare e studi teorici hanno dimostrato che l'esistenza di nuclei con numero atomico più grande di 102 è interamente affidata agli effetti quantistici che gli specialisti definiscono “di shell”.

Nella carta dei nuclidi di figura 1.2, i nuclei definiti *stabili* (quadratini neri) sono quelli non radioattivi oppure quelli aventi tempo di decadimento comparabile o più lungo dell'età della terra, mentre la regione di colore giallo è quella dei nuclei artificiali che possono avere vita più o meno breve. Aggiungendo neutroni o protoni ad un nucleo ci si allontana dalla valle di stabilità sino a raggiungere i limiti, detti *drip lines*, caratterizzati da una diminuzione della forza di attrazione tra neutroni e protoni tale da non garantire più la stabilità nucleare: calcoli teorici hanno dimostrato che al di fuori delle drip lines i nuclei emettono nucleoni molto rapidamente per formare nuovi nuclei con combinazioni di protoni e neutroni tali da poter rientrare nell’area di potenziale stabilità, nella quale l'interazione forte è nuovamente capace di garantire stabilità all’insieme dei nucleoni. La regione indicata in verde, ancora inesplorata, è definita *terra incognita* ed è caratterizzata dalla presenza di nuclei radioattivi con rapporti N/Z molto piccoli o molto grandi; la figura mostra che l’area proton-rich è relativamente ben definita teoricamente, mentre quella neutron-rich è molto più vasta ed indefinita. Come vedremo nei paragrafi successivi, lo studio dei nuclei instabili, ed in particolare dei nuclei esotici, ha aperto nuovi campi di studio nella fisica nucleare, ha permesso di confermare precedenti ipotesi di fondamentale importanza in fisica atomica (quale, ad esempio, il modello standard) ed ha infine suggerito promettenti applicazioni in fisica dello stato solido ed in medicina. Per l’utilizzo pratico e la produzione di ioni radioattivi di questo tipo è necessaria la costruzione di sistemi acceleratori ed attrezzature capaci di garantire fasci ionici di elevata purezza, intensità ed energia (detti anche facilities).

1.1 Applicazioni degli ioni esotici

Nel paragrafi seguenti verranno brevemente descritte le principali applicazioni di questi fasci di ioni radioattivi nelle aree sopra citate, dedicando maggior spazio alle applicazioni in fisica nucleare e in fisica dello stato solido essendo queste di maggior interesse ai fini di questo lavoro. [3, 4]

1.1.1 Applicazioni alla fisica nucleare

Miglioramento e verifica del modello standard.
Il Modello Standard della fisica delle particelle è una teoria che descrive insieme tre delle quattro forze fondamentali, cioè l'interazione nucleare forte, elettromagnetica e l'interazione nucleare debole (queste ultime due unificate nell'interazione elettrodebole), nonché la funzione e le proprietà di tutte le particelle (note ed osservate) che costituiscono la materia. Nonostante il suo

successo, tale modello non è del tutto soddisfacente, poiché dipende in modo sostanziale da alcune assunzioni fatte ad hoc.
Complessi esperimenti di fisica nucleare, suggeriti da convincenti basi teoriche, sono stati ideati allo scopo di chiarire l'origine di queste assunzioni e pervenire così all'unificazione delle interazioni fondamentali. Tali esperimenti prevedono precise misure delle proprietà di decadimento di alcuni nuclei, che possono essere effettuate proprio utilizzando come sorgente pura di ioni i fasci di ioni radioattivi prodotti dagli impianti.

Studio della struttura di nuclei complessi.
I nucleoni (protoni e neutroni) sono costituiti da subparticelle chiamate quark: esse esercitano un effetto fisico anche oltre i nucleoni nei quali sono confinati e, in particolare, le interazioni tra i nucleoni all'interno del nucleo sono diverse da quelle esistenti tra due nucleoni liberi, in quanto esse dipendono anche dalla densità di protoni e neutroni associata al particolare tipo di nucleo.
Al momento, non esiste una formula generale che consenta di quantificare l'entità delle interazioni nucleari per tutti i nuclei sintetizzati e naturali, in quanto i calcoli quantomeccanici sono applicabili unicamente ai nuclei più leggeri; l'obiettivo della fisica nucleare è di ottenere una trattazione unitaria che:

- permetta di derivare l'effettiva interazione tra le particelle nucleari;
- elimini le incongruenze dei modelli correnti;
- sia applicabile anche ai nuclei aventi rapporto protoni/neutroni elevato.

A questo proposito i fasci di ioni radioattivi possono fornire un prezioso contributo.

Misura della dimensione del nucleo: i nuclei *halo*.
La dimensione del nucleo è legata al numero totale di nucleoni che lo costituiscono. Indicando con A il numero totale di nucleoni, si può determinare il raggio nucleare R con la semplice funzione:
$R=R_0 A^{1/3}$
dove R_0 è una costante pari ad 1,2 fermi. Tuttavia, allontanandosi dalla valle di stabilità della carta dei nuclidi intervengono fenomeni quantistici inusuali che permettono ad uno o più neutroni di valenza debolmente legati di muoversi intorno al cuore del nucleo su orbite molto larghe, aumentandone effettivamente il diametro di parecchie volte. Questo tipo di nuclei sono chiamati nuclei *halo*, e sono oggetto di ricerca per le possibili applicazioni nelle tecnologie nucleari.

Produzione di elementi superpesanti.
I chimici ed i fisici nucleari si sforzano di completare il sistema periodico degli elementi scoprendo elementi sempre più pesanti e studiando le loro proprietà chimiche e fisiche.
La disponibilità di fasci radioattivi di alta intensità, costituiti da nuclei instabili ricchi di neutroni (n-rich), accoppiati a target stabili, anch'essi ricchi di

neutroni permetterebbe la sintesi e lo studio di nuovi elementi chimici lontani dalla zona di stabilità nucleare, e possibilmente alla scoperta degli sfuggenti nuclei magici superpesanti, che sono stati predetti fin dagli anni '70 ma non sono ancora stati trovati. La scoperta di questa *isola di stabilità* circondata da una zona popolata esclusivamente da isotopi instabili costituisce una nuova speranza per la fisica nucleare. [3, 5]

1.1.2 Applicazioni in fisica dello stato solido

Una delle principali applicazioni dei fasci di ioni esotici alla fisica dello stato solido è rappresentata dalla tecnica *radio tracer diffusion*, nata nel 1920. Questa procedura consiste nell'impiantare all'interno di un sistema solido dei nuclei radioattivi e di studiarne il decadimento, rilevando le particelle o la radiazione gamma da essi emessa. Tale tecnica permette di captare segnali anche da pochissimi atomi e rappresenta uno dei metodi più comuni per studiare i processi di diffusione atomica nei solidi.
Il sistema ospitante può essere drogato con i radioisotopi sonda per diffusione, oppure tramite un impianto ionico; la scelta dell'atomo radioattivo da utilizzare per un determinato esperimento viene fatta in base alla natura chimica e alle proprietà nucleari di quest'ultimo.
L'uso della tecnica radio tracer diffusion consente:

- di osservare, tramite i prodotti di decadimento, l'interazione tra l'atomo sonda e il reticolo che lo circonda;
- di ottenere informazioni riguardanti il campo elettrico e magnetico all'interno del cristallo;
- di studiare i processi diffusivi e le interazioni tra gli atomi sonda;
- di individuare i tipi di difetti presenti nel cristallo.

Drogaggio dei semiconduttori.
Lo sviluppo di semiconduttori di piccole dimensioni aventi caratteristiche ottiche ed elettriche ottimali richiede un controllo completo dei difetti che governano tali proprietà, sia intrinseci (come le vacanze interstiziali) che estrinseci (come i droganti e le impurità atomiche); per tale motivo si stanno dedicando notevoli risorse nelle tecnologie per lo studio dei difetti e dell'attivazione elettrica dei droganti in diversi semiconduttori.

Gli isotopi radioattivi, analogamente a quelli stabili, in base alla natura chimica del semiconduttore ed alla posizione che assumono all'interno del reticolo cristallino, sono in grado di influenzare le proprietà elettroniche ed ottiche dei semiconduttori. Si è dimostrato che in semiconduttori molto piccoli tali proprietà possono essere sensibilmente alterate dalla presenza di un difetto avente una concentrazione minore di 10^{12} atomi/cm^3 (veramente esigua in termici atomici).
Per controllare in maniera affidabile le prestazioni dei semiconduttori sono quindi necessarie tecniche sperimentali che combinino un'alta sensibilità chimica con un'alta sensibilità per la determinazione di basse concentrazioni di difetti.

Per decenni la principale tecnica di rilevazione delle impurezze all'interno di un cristallo è stata il channeling (fig. 1.3), in cui un fascio di ioni (ad esempio He+ a molti MeV) viene guidato lungo le righe atomiche o lungo i piani del cristallo (canali). Tuttavia questo metodo non è sensibile a concentrazioni di difetti inferiori a 10^{18} atomi/cm^3.

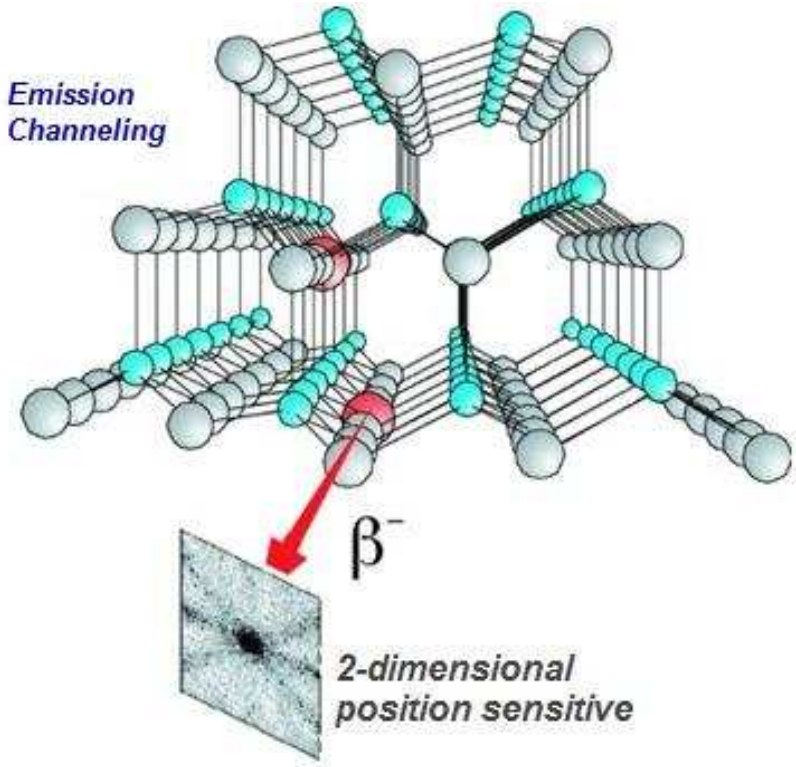

Figura 1.3 Rappresentazione schematica del channeling

La sensibilità può essere aumentata di diversi ordini di grandezza impiantando all'interno del cristallo impurezze radioattive che emettono particelle cariche (emission channeling): rilevare l'emissione di tali particelle lungo le direzioni cristallografiche principali della sostanza porta ad ottenere infatti rese di emissione diverse rispetto a quelle misurate lungo una direzione cristallografica casuale. Nel caso in cui le particelle emesse siano elettroni, l'aumento della resa di emissione lungo una certa direzione cristallografica implica che gli atomi emettitori si trovano lungo la riga cristallina oppure in vicinanza di essa (fig. 1.3); viceversa, l'assenza di un aumento oppure una diminuzione di resa lungo un asse principale suggerisce che l'atomo emettitore occupa un sito interstiziale. La misura dell'emissione lungo differenti direzioni cristalline permette la determinazione del sito cristallografico dell'atomo emittente con un'accuratezza di poche decime di picometro.

Studio delle proprietà magnetiche di superfici e monostrati atomici metallici.

Le superfici e le interfacce dei solidi costituiscono un campo di ricerca di notevole interesse in molte aree della fisica. In particolare è possibile

studiare le proprietà magnetiche degli strati metallici ultrasottili e le variazioni che queste possono subire passando da uno strato atomico all'altro. La possibilità di utilizzare diversi atomi radioattivi sonda permette di testare sperimentalmente predizioni teoriche molto dettagliate.
Un esempio applicativo dei fasci di ioni radioattivi riguarda la determinazione delle proprietà magnetiche e strutturali di film atomici di Palladio su bulk di Nichel. In questo ambito lo studio può essere condotto impiantando isotopi radioattivi di Pd (Palladio) o Cd (Cadmio) a differenti distanze dall'interfaccia (fig. 1.4).

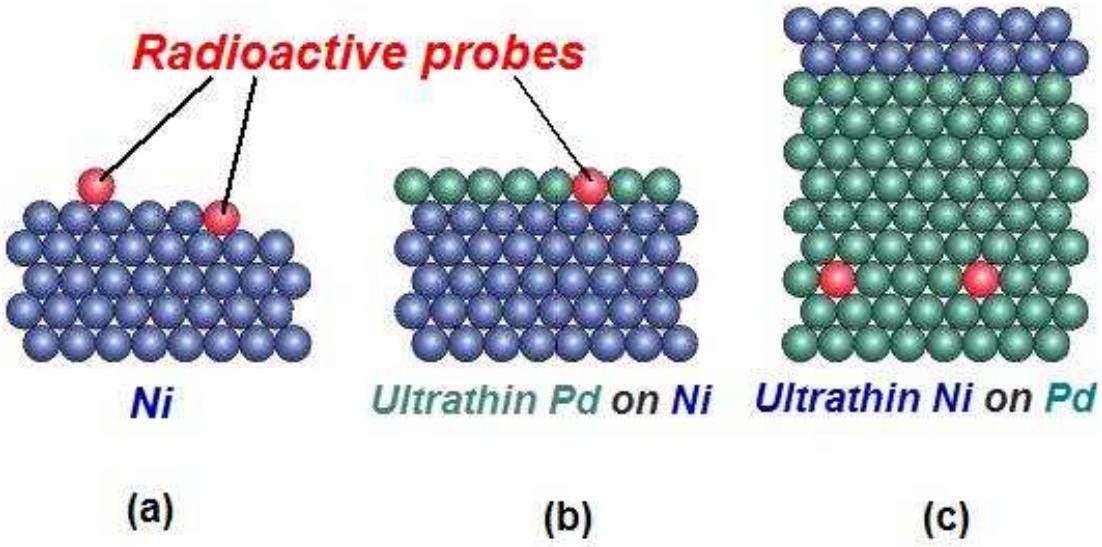

Figura 1.4 Esempio di analisi di proprietà magnetiche

In questo modo è stato possibile determinare, con una risoluzione strutturale mai raggiunta, il comportamento ferromagnetico di uno strato ultrasottile di Palladio cresciuto su Nichel e le proprietà magnetiche indotte sul Palladio da uno strato ultrasottile di Nichel. [3, 4]

1.1.3 Applicazioni mediche: la Tomografia ad Emissione di Positroni (PET)

Prima di procedere si forniscono alcune fondamentali definizioni:
1. l'antimateria è la materia composta da antiparticelle, particelle aventi la stessa massa e caratteristiche opposte a quelle che costituiscono la materia ordinaria;
2. il positrone (detto anche antielettrone) è l'equivalente di antimateria dell'elettrone ed ha carica elettrica pari a +1. Quando un positrone si annichila con un elettrone, la loro massa viene convertita in energia, sotto forma di due fotoni ad altissima energia nella banda dei raggi gamma. Un positrone può essere generato dal decadimento radioattivo con emissione di positroni o dall'interazione con la materia di fotoni con energia superiore a 1,022 MeV .

Sebbene utilizzata principalmente per studiare le interazioni tra particelle elementari, l'antimateria ha anche un'applicazione tecnologica: la Tomografia ad Emissione Positronica (PET, Positron Emission Tomography), una tecnica di medicina nucleare e diagnostica medica che utilizza l'emissione di positroni per realizzare immagini tridimensionali o mappe ad alta risoluzione degli organi interni dei pazienti. Essa fornisce immagini di sezioni dell'organismo umano, analogamente alla tomografia assiale computerizzata a raggi X (TAC), ed appartiene alla categoria delle metodiche dette di *Tomografia ad emissione*.

Nella tomografia ad emissione le immagini tomografiche descrivono la distribuzione di una sostanza radioattiva (radiotracciante) somministrata in quantità molto piccole, di norma per via endovenosa. Un radiotracciante è ottenuto legando un isotopo radioattivo ad una molecola che traccia (segue) un determinato processo biochimico o fisiologico nell'organismo. La misura della distribuzione del radiotracciante nell'organismo, in accordo al processo biochimico o fisiologico in studio, fornisce informazioni sulla funzionalità regionale dell'organo in esame. In funzione del radiotracciante utilizzato, si possono ottenere immagini rappresentative di funzioni diverse, per esempio della pressione sanguigna (cerebrale o miocardica) o del livello metabolico di un dato tessuto (valutato come consumo di glucosio).

Pertanto, mentre nella TAC si misura l'attenuazione di un fascio di raggi X emessi dall'apparecchio radiologico ad opera dei tessuti corporei, nella tomografia ad emissione si rivelano le radiazioni emesse direttamente dal corpo del paziente dopo la somministrazione di un apposito radiotracciante. Pertanto con la TAC le patologie vengono rilevate da una variazione dell'attenuazione dei raggi X che corrisponde ad una variazione della densità dei tessuti; al contrario la tomografia ad emissione fornisce immagini funzionali ed evidenzia la patologia come una regione con anomala captazione del radiotracciante a causa di un'alterazione della funzione ad esso associata.

Dopo un certo tempo di attesa dalla somministrazione del tracciante, durante il quale la molecola metabolicamente attiva (spesso uno zucchero) raggiunge una determinata concentrazione all'interno dei tessuti organici da analizzare, il soggetto viene posizionato nello scanner.
L'isotopo di breve vita media decade, emettendo un positrone. Dopo un percorso che può raggiungere al massimo pochi millimentri, il positrone si annichila con un elettrone, producendo una coppia di fotoni (di energia paragonabile a quella dei raggi gamma) emessi in direzioni opposte tra loro (sfasate di 180◦ lungo la stessa retta). Tali fotoni vengono successivamente rilevati dal dispositivo di scansione, grazie anche all'impiego di speciali tubi fotomoltiplicatori. Punto cruciale della tecnica è la rilevazione simultanea di coppie di fotoni: i fotoni che non raggiungono il rilevatore in coppia, cioè entro un intervallo di pochi nanosecondi, non sono presi in considerazione. Dalla misurazione della posizione in cui i fotoni colpiscono il rilevatore (ogni coppia di fotoni individua una retta) si può ricostruire la posizione del corpo da cui sono stati emessi (teoricamente con due coppie di fotoni, e dunque con due

rette, è possibile individuare il punto di emissione dei fotoni), permettendo la determinazione dell'attività o dell'utilizzo chimico all'interno delle parti del corpo investigate.
Lo scanner utilizza la rilevazione delle coppie di fotoni per mappare la den-densità dell'isotopo nel corpo; la mappa risultante rappresenta i tessuti in cui la molecola campione si è maggiormente concentrata e viene letta ed interpretata da uno specialista in medicina nucleare o in radiologia al fine di determinare una diagnosi ed il conseguente trattamento. Spesso, e sempre più frequentemente, le scansioni della Tomografia a Emissione di Positroni sono raffrontate con le scansioni a Risonanza Magnetica nucleare, fornendo informazioni sia morfologiche che metaboliche, cioè sull'anatomia del tessuto o dell'organo di interesse e su cosa stiano facendo. La PET è usata estensivamente in oncologia clinica per avere rappresentazioni dei tumori e per la ricerca di metastasi e nelle ricerche cardiologiche e neurologiche.
I radionuclidi utilizzati nella scansione PET sono generalmente isotopi con breve tempo di dimezzamento, come ^{11}C (~ 20min), ^{13}N (~ 10min), ^{15}O (~ 2min) e ^{18}F (~ 110min). Per via del loro basso tempo di dimezzamento, i radioisotopi devono essere prodotti da un ciclotrone posizionato in prossimità dello scansionatore PET.

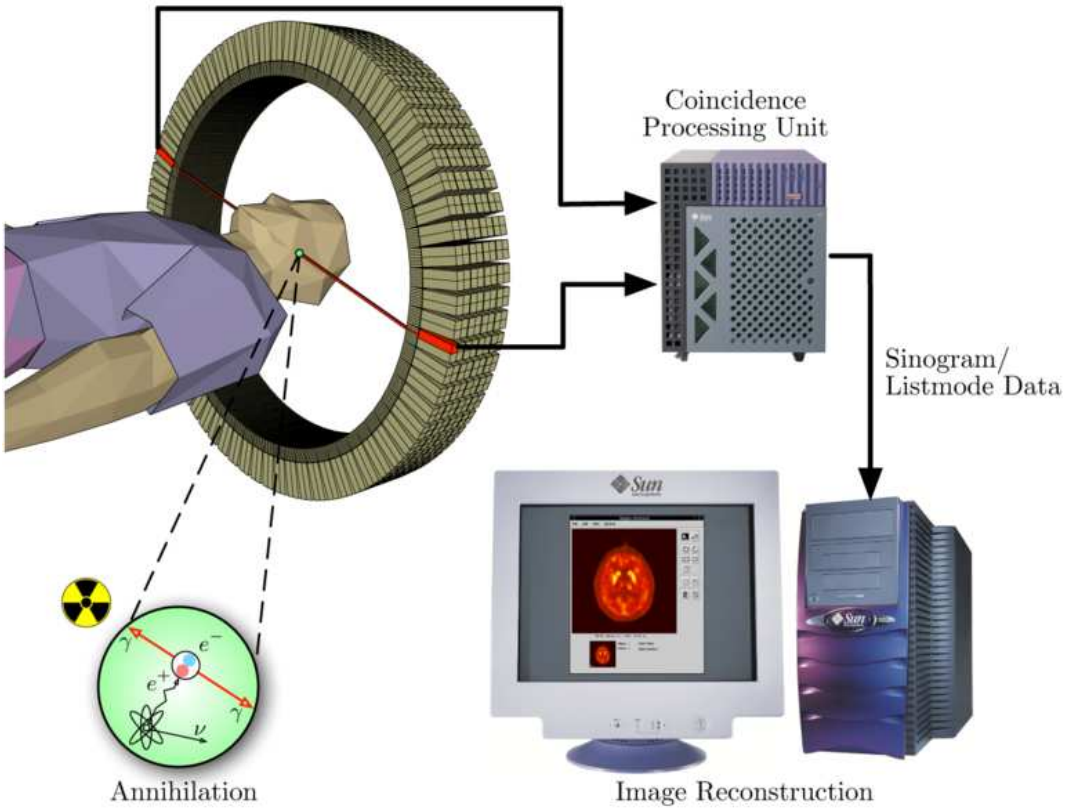

Figura 1.5 Tomografia ad emissione di positroni

Una alternativa alla tomografia ad emissione di positroni (PET) è la tomografia ad emissione di singolo fotone (Single Photon Emission Tomography, SPET). In entrambe le tecniche SPET e PET viene somministrato al paziente un tracciante radioattivo e successivamente si rivelano le radiazioni emesse dal radiotracciante ottenendo in questo modo le

immagini tomografiche delle sezioni corporee. Le due tecniche SPET e PET si differenziano, tuttavia, per le caratteristiche fisiche dei radiotraccianti utilizzati e, di conseguenza, per i sistemi di rivelazione delle radiazioni emesse.

Ad ogni modo, mentre gli altri metodi di scansione, come la TAC e la RMN (risonanza magnetica nucleare) permettono di identificare alterazioni organiche e anatomiche nel corpo umano, le scansioni PET sono in grado di rilevare alterazioni a livello biomolecolare che spesso precedono l'alterazione anatomica, attraverso l'uso di marcatori molecolari che presentano un diverso ritmo di assorbimento a seconda del tessuto interessato.

La popolarità di questa tecnica è dovuta alla possibilità che offre di verificare la risposta alla terapia, specialmente in particolari terapie anti-cancro; per tali ragioni si prospettano dunque per essa sempre maggiori applicazioni e sviluppi. [6]

1.1.4 Applicazioni nel campo astrofisico

L’astrofisica nucleare gioca un ruolo fondamentale nella comprensione della struttura, evoluzione e composizione dell’Universo e dei suoi costituenti, sia per quanto riguarda la produzione di energia che la formazione di elementi chimici (nucleosintesi).
Le stelle generano energia attraverso reazioni nucleari coinvolgenti sia nuclei stabili che radioattivi. A volte il consumo del carburante nucleare procede stabilmente e dura bilioni di anni, altre volte è esplosivo e dura pochi minuti o secondi.
Nelle differenti fasi della consunzione delle stelle vengono sintetizzati nuovi elementi chimici sia tramite processi di nucleosintesi che seguono strettamente la valle di stabilità, sia attraverso processi che si svolgono in un territorio sconosciuto. Per sviluppare un modello che descriva il meccanismo di nucleosintesi, è necessario misurare le rese delle reazioni nucleari relative ai principali cicli astrofisici e le caratteristiche di decadimento di molti nuclei tuttora sconosciuti. Queste essenziali informazioni includono i tempi di vita, le masse ed i principali canali di decadimento di un numero di nuclei chiave lontani dalla stabilità.
Numerosi modelli astronomici prevedono che in alcune sorgenti astrofisiche, come nuclei galattici attivi o i lampi di raggi gamma, avvengono fenomeni ancora sconosciuti che si crede siano molto simili a quelli prodotti dagli scienziati nei laboratori di fisica nucleare e subnucleare. Alcune osservazioni indicano che queste sorgenti possano accelerare particelle (protoni e nuclei) fino ad energie macroscopiche: 10 Joule (100 miliardi di miliardi di eV). Queste sorgenti si trovano nello spazio remoto e soltanto la componente di bassa energia delle loro emissioni è osservabile dalla Terra. Infatti i protoni ed i fotoni (raggi gamma) di alta energia provenienti da queste sorgenti sono assorbiti nello spazio. Al contrario i neutrini, che hanno una bassissima probabilità di essere assorbiti, possono giungere sulla Terra da regioni lontanissime dell'Universo.

Tutte queste reazioni nucleari coinvolgenti nuclei instabili possono essere misurate unicamente con un fascio radioattivo: per tale motivo si prevede che la nuova generazione di facility per la produzione di fasci radioattivi risulterà di fondamentale importanza per la comprensione della sintesi elementare nell'Universo. [4, 6]

1.2 La produzione dei fasci RIB

La produzione dei fasci RIB può essere effettuata con diversi metodi, a seconda del tipo di isotopi che si vogliono ottenere. Per ottenere nuclei ricchi ricchi di protoni (p-rich) esistono metodi come la frammentazione, la spallazione, o le reazioni di trasferimento; nuclei ricchi di neutroni (n-rich) invece sono solitamente ottenuti con reazioni di fissione nucleare, indotte da nucleoni (protoni o peutroni) o attraverso fotofissione. La produzione degli isotopi all'interno del bersaglio è ottenuta, principalmente, tramite il processo di fissione dell'uranio. La fissione nucleare consiste nella rottura del nucleo di un elemento in due o più nuclei di elementi più leggeri (fig. 1.6).

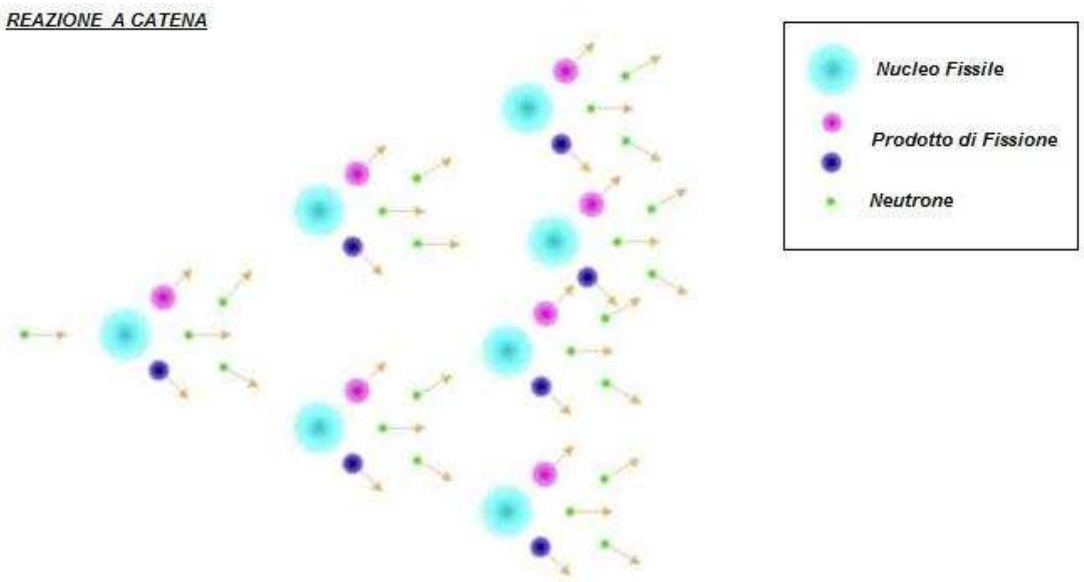

Figura 1.6 Il processo di fissione

In figura 1.6 viene rappresentato schematicamente il processo di fissione ottenuto tramite il bombardamento del nucleo fissile con un neutrone, tuttavia la fissione può avvenire anche mediante un bombardamento protonico, come nel caso del progetto SPES.
Questa reazione interessa prevalentemente elementi con un numero di massa superiore a 100, ma si osserva più facilmente in elementi aventi una massa intorno a 230. I protoni di fissione derivanti da questi elementi, infatti, possiedono una massa totale leggermente inferiore a quella del nucleo di partenza; questa differenza di massa è la causa dell'energia prodotta nella reazione perché la massa persa si trasforma in energia, secondo l'equazione di Einstein $E=mc^2$. La fissione, in genere, avviene a causa della cattura di un

neutrone da parte del nucleo di un elemento pesante, più facilmente da parte di un isotopo di un elemento stabile. Negli elementi chimici, il rapporto tra neutroni e protoni cresce con l'aumentare del numero atomico, partendo da un valore di 1 per l'idrogeno; questo causa instabilità negli elementi pesanti e, in maniera maggiore, nei loro isotopi, visto che entrambi possiedono energie di legame molto basse. Basta, quindi, che un nucleo di tali elementi catturi un neutrone perché l'energia rilasciata dal riassestamento dei nucleoni ecceda l'energia di legame. A questo punto, il nucleo acquista un'anomala forma allungata e, nel giro di qualche frazione di secondo (meno di 10-12 s), il nucleo si divide in due nuclei instabili a numero atomico notevolmente più basso, emettendo, inoltre, alcuni neutroni (in media 2 o 3) ed energia sotto forma di radiazioni elettromagnetiche. Gli isotopi così generati all'interno del bersaglio andranno successivamente a comporre dei fasci di ioni radioattivi denominati RIB (Radioactive Ions Beams).
La produzione di questi RIB sia nelle facilities già operanti che in quelle in via di costruzione viene effettuata principalmente per mezzo di due metodi (fig. 1.7 e 1.8):

- il metodo IN-FLIGHT o FRS (Frangment Recoil Separator), basato sulla frammentazione dell'isotopo in volo;
- il metodo ISOL (Isotope Separation On-Line), che utilizza la separazione degli isotopi in linea.

I due metodi si distinguono principalmente in base alla dimensione del target di produzione richiedendo ognuno una tecnologia di estrazione degli isotopi radioattivi prodotti differente. [3]

Nei paragrafi successivi verrà presentata una breve descrizione dei due metodi citati evidenziandone pregi e difetti.

1.2.1 Il metodo In-Flight o FRS

Il metodo FRS sfrutta la frammentazione degli ioni del fascio primario causata dalla loro collisione con un bersaglio sottile (target di produzione). Tale processo viene illustrato nella figura seguente (fig. 1.7).

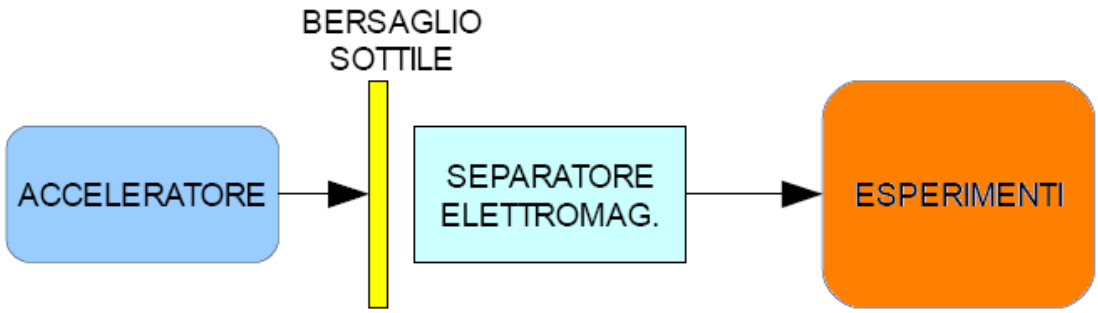

Figura 1.7 Rappresentazione schematica del metodo FRS

Come si osserva in figura 1.7 un separatore elettromagnetico seleziona in base alla massa e alla carica gli ioni esotici prodotti e li convoglia alle sale sperimentali.

Grazie alla natura del target i prodotti di reazione sono generati in volo e non necessitano di un sistema di post-accelerazione, anche se la facility può venire potenziata grazie all'utilizzo di un anello di accumulazione. Il fascio radioattivo prodotto con il metodo IN-FLIGHT presenta due caratteristiche fondamentali:

- elevata energia dei frammenti radioattivi, derivanti direttamente dagli ioni ad alta energia del fascio primario; questi attraversano il target sottile senza sostanziali attenuazioni e non necessitano quindi di un'ulteriore accelerazione per giungere al rilevatore secondario;
- piccolo ritardo tra la produzione dei frammenti ed il loro utilizzo: il ritardo dipende unicamente dal tempo impiegato dai frammenti a percorrere il tragitto target di produzione-target secondario passando attraverso il separatore elettromagnetico.

Si è dimostrato che la frammentazione degli ioni del fascio primario e la fissione in volo, consentono la produzione di fasci relativistici di nuclei esotici appartenenti all'intera carta dei nuclidi. [3]

1.2.2 Il metodo ISOL

La tecnica ISOL (fig. 1.8) viene associata all'utilizzo di bersagli spessi, all'interno dei quali i nuclei radioattivi sono prodotti praticamente a riposo in un bersaglio spesso, bombardato con particelle provenienti da una sorgente primaria (o driver accelerator). I prodotti della reazione sono a temperatura molto alta e successivamente sono portati ad una sorgente di ionizzazione.

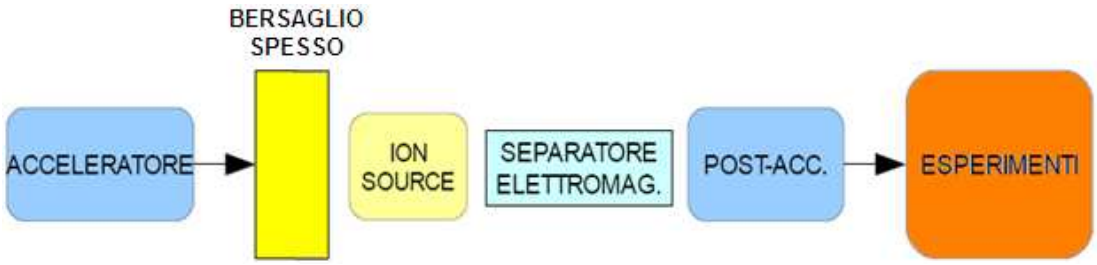

Figura 1.8 Rappresentazione schematica del metodo ISOL

Gli ioni radioattivi così ottenuti vengono estratti, selezionati con metodi elettromagnetici, accelerati una prima volta verso il separatore elettromagnetico e quindi post-accelerati fino al target di reazione per gli esperimenti. Si ottengono con tale tecnica fasci intensi e di alta qualità fino ad energie di 25 MeV/u. I principali costituenti di una facility di tipo ISOL sono:

- l'acceleratore primario (driver);
- il complesso formato dal bersaglio più la sorgente (TIS, Target Ion Source);
- il complesso di *manipolazione del fascio* (separatore e charge breeder);
- il post-acceleratore;

L'acceleratore primario è costituito da una sorgente che inietta protoni oppure atomi ionizzati a bassa energia (alcune decine di KeV) nell'acceleratore il quale li accelera fino a raggiungere un'energia massime pari a 1 GeV e qualche mA di intensità di corrente. Il fascio primario è

successivamente convogliato nel complesso bersaglio-sorgente nel quale avviene la produzione di isotopi radioattivi. Prima di essere inviato alle utenze, il fascio viene selezionato e purificato mediante separatori elettromagnetici e post-accelerato fino all'energia richiesta dall'esperimento.
La tecnica ISOL permette di ottenere fasci di ioni radioattivi di qualità migliore rispetto a quelli ottenuti con il metodo IN-FLIGHT, con caratteristiche più adatte agli esperimenti di fisica e astrofisica nucleare. Il principale svantaggio della tecnica ISOL è il ritardo dovuto alla relativa lentezza dei processi di diffusione, effusione ed estrazione dei nuclidi tra la sorgente di ionizzazione ed il complesso bersaglio-sorgente: risulta perciò problematica la gestione di nuclidi con tempo di decadimento inferiore a qualche decina di millisecondi. È necessaria un'accurata progettazione del gruppo bersaglio-sorgente affinché un sistema di tipo ISOL sia efficiente, vale a dire la riduzione del ritardo tra la produzione e l'arrivo delle specie esotiche e l'incremento della produzione senza dover rinunciare alla purezza del fascio. L'intensità degli ioni radioattivi prodotti da una facility di tipo ISOL è data dalla seguente relazione:

$$I = \sigma \cdot \varphi \cdot \tau \cdot \varepsilon_1 \cdot \varepsilon_2 \cdot \varepsilon_3$$

dove:

- σ è la sezione d'urto per le reazioni nucleari d'interesse;
- φ è l'intensità di corrente del fascio primario;
- τ è lo spessore sfruttabile della targhetta;
- ε_1 è l'efficienza di rilascio (dovuta agli effetti di diffusione e di effusione all'interno del target);
- ε_2 è l'efficienza di ionizzazione della sorgente;
- ε_3 è l'efficienza di trasmissione dell'acceleratore finale (separatore di massa isobarico, charge breeder, linee di trasporto fino agli apparati di rivelazione nelle sale sperimentali).

La separazione dei prodotti radioattivi dal substrato del target ed il loro trasferimento alla sorgente per la ionizzazione sono processi che seguono le leggi del moto browniano, quindi fortemente dipendenti dalla temperatura. Chiaramente, più breve è la vita media degli atomi radioattivi, più rapido deve essere il tempo di rilascio e di conseguenza il bersaglio deve essere mantenuto alla più alta temperatura possibile.
I metodi FRS ed ISOL consentono la realizzazione di esperimenti complementari tra loro: i fasci radioattivi prodotti da una facility FRS vengono usati in esperimenti che richiedono alte energie (generalmente 50÷100 MeV/nucleone) mentre quelli prodotti da una facility ISOL (che consente accelerazioni variabili da bassissime energie fino a circa 10÷20 MeV/nucleone) sono più adatti a studi della struttura nucleare e di astrofisica. Il range di energie riproducibile da una facility di tipo ISOL è comparabile con quello dei nucleoni nel nucleo alle elevate temperature in ambiente stellare. Esiste un'area di sovrapposizione dei due metodi nel campo delle energie intermedie, dove lo stesso fenomeno può essere studiato con tecniche sperimentali alternative; i metodi FRS ed ISOL in queste circostanze possono essere infatti utilizzati insieme, in modo da sfruttare i vantaggi che ciascuno di essi offre. In tal modo è possibile combinare la produzione universale garantita dal metodo FRS con l'eccellente qualità del fascio di tipo ISOL. [3, 6]

1.3 Il progetto SPES presso i LNL

In ambito europeo, le opportunità scientifiche offerte dai RIB ed i notevoli problemi tecnologici associati hanno suggerito la costruzione di una rete di impianti complementari, definite di *intermedia generazione*, fondamentali per arrivare alla costruzione di un'unica grande facility europea di tipo ISOL, chiamata EURISOL: il progetto rappresenta un'iniziativa che vede coinvolti i principali laboratori nucleari europei ed è dedicato allo studio ed alla progettazione di una struttura per la produzione di fasci radioattivi di qualità significativamente superiore a quella attualmente disponibile. Il progetto SPES (Search and Production of Exotic Species) nasce con l'intento di usare un sistema ISOL per lo studio degli ioni esotici prodotti da bersagli di carburo di uranio colpiti da un fascio protonico da 40MeV e 200 μA (P = 8kW).
Tale programma, coordinato a livello nazionale, prevede la collaborazione di sei sezioni dell'Istituto Nazionale di Fisica Nucleare (LNL, LNS, sezione di Napoli, sezione di Milano, sezione di Bologna, sezione di Pavia), dei Dipartimenti di Ingegneria e di Scienze Chimiche dell'Università degli Studi di Trento e Padova, dell'ENEA (Bologna), e, a livello internazionale, strette collaborazioni con il CERN (Ginevra), i Laboratori di Oak Ridge (Tennessee, USA), e con il High Energy Accelerator Research Organization (Giappone).
I Laboratori Nazionali di Legnaro (LNL) partecipano attivamente nell'ambito del progetto SPES che prevede la costruzione di un impianto di tipo ISOL per la produzione di fasci di isotopi radioattivi ricchi di neutroni (n-rich) e di alta qualità, nel range di massa compreso tra 80 e 160.

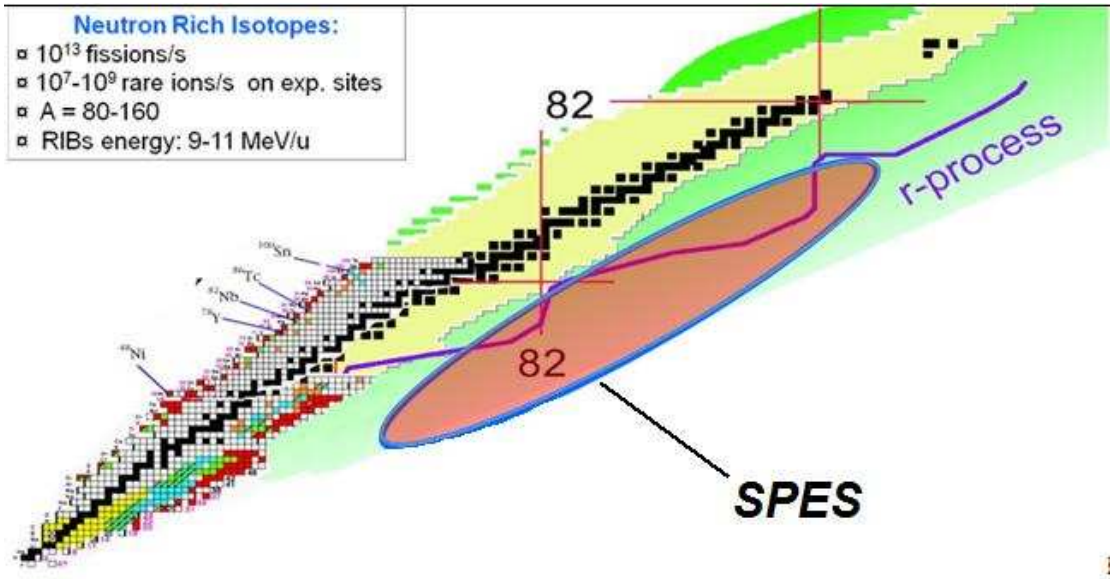

Figura 1.9 Area di studio del progetto SPES (carta dei nuclidi)

L'acceleratore SPES, come tutte le facility di tipo ISOL, è composto da 4 distinte sezioni:

- l'acceleratore primario che fornisce il fascio di protoni necessario per la produzione degli isotopi attraverso la collusione con il target;

- il complesso costituito dal target e dal sistema di ionizzazione;
- i sistemi di manipolazione del fascio (separatori di massa e charge breeder);
- il sistema di post-accelerazione.

Il fascio primario viene prodotto da un ciclotrone che produce protoni con energia di 40MeV e corrente di 200 μA.
Il fascio protonico incide quindi sul bersaglio, causando fino a 10^{13} fissioni al secondo; i radioisotopi prodotti nel target vengono ionizzati ed estratti all'energia di circa 60 KeV per essere poi separati in massa nella sezione successiva dell'apparato.
La separazione di massa avviene attraverso un separatore elettromagnetico (filtro di Wien) in grado di filtrare una grande quantità di contaminanti. La risoluzione di tale dispositivo è tale da permettere di separare in base alla massa i vari elementi. A seguire vi è uno spettrofotometro ad alta risoluzione, che permette di separate gli isotopi (ad esempio 132Cs e 132Sn). Per migliorare la capacità di selezione dei separatori isobari è necessario operare con una energia di ingresso dell'ordine dei 200 keV, e ciò significa che sia il target che il primo separatore vengono montati su una piattaforma di alto voltaggio a 260 kV.

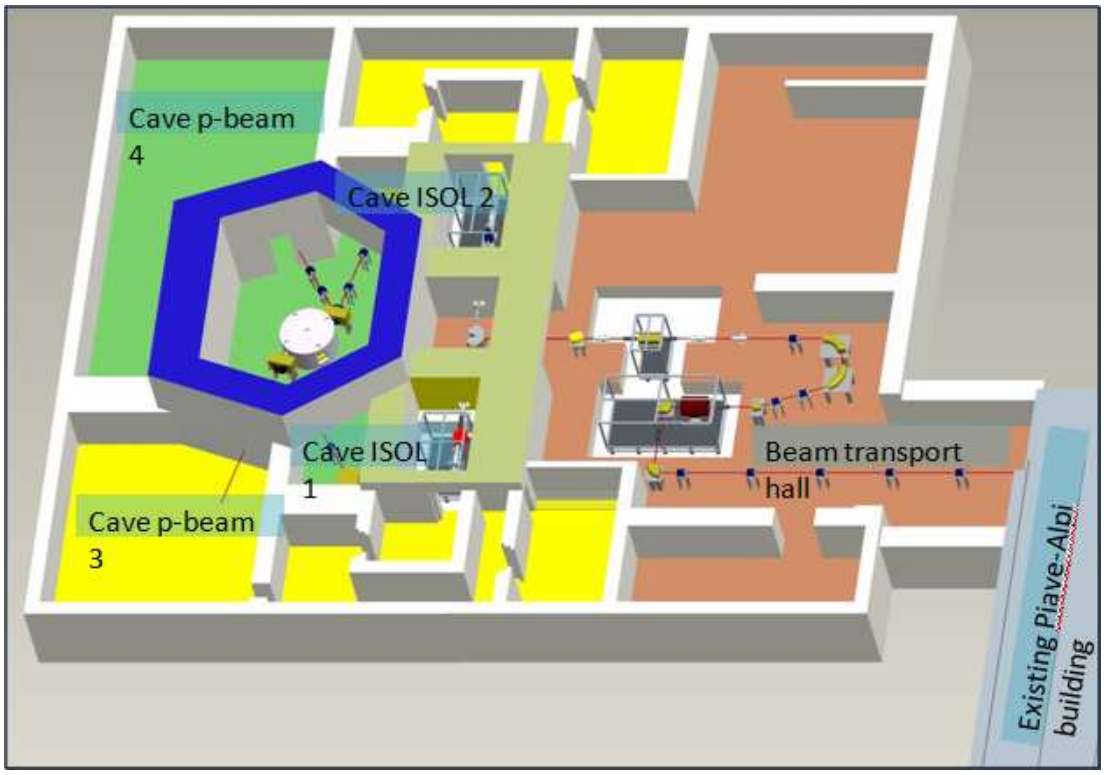

Figura 1.10 Struttura dell'impianto SPES

A questo punto il fascio radioattivo può essere direttamente convogliato alle sale sperimentali (ed utilizzato in esperimenti che richiedono fasci radioattivi di bassissima energia), oppure post-accelerato fino a diversi MeV dall'acceleratore ALPI preesistente presso i Laboratori Nazionali di Legnaro

(acceleratore LINAC superconduttore a cavità risonanti) ed infine utilizzato per le sperimentazioni.
Tuttavia, prima dell'iniezione nel complesso PIAVE-ALPI, esso può venire diretto verso un charge breeder, dispositivo che serve ad incrementare la carica degli ioni (e quindi anche la loro successiva accelerazione). [3, 6]

1.4 Conclusioni

La produzione di fasci di particelle esotiche è di importanza rilevante per applicazioni che vanno dallo studio della materia fino ad alcune applicazioni mediche.
Il progetto EURISOL, fondato dalla Comunità Europea, prevede lo studio e la realizzazione di un grande impianto di tipo ISOL; i Laboratori Nazionali di Legnaro dell'Istituto Nazionale di Fisica Nucleare partecipano attivamente all'attività di ricerca e sviluppo in questo ambito con il progetto SPES.
Utilizzando un sistema target innovativo, esso si propone di produrre isotopi radioattivi ricchi di neutroni come prodotti di fissione dell'uranio; questi fasci esotici verranno poi iniettati nel sistema di post-accelerazione già esistente, al fine di essere studiati.

Capitolo 2 – Strumentazione del Front-End

Il progetto SPES, come ogni struttura di tipo ISOL, si compone di 4 parti: un acceleratore primario, un sistema target/ionizzatore, un sistema di manipolazione ed un post-acceleratore. Il complesso target/ionizzatore è il cuore del Front End del progetto SPES; esso è un sistema complesso, composto dal target, dal sistema di ionizzazione ed estrazione, dai relativi sistemi di vuoto, riscaldamento e raffreddamento, dai primi metri del sistema di manipolazione e dalla diagnostica. Non meno complesso è il sistema di controllo di tutta la strumentazione.
I seguenti paragrafi descrivono le caratteristiche ed il funzionamento di tutta la strumentazione, soffermandosi infine su quali siano le esigenze dal punto di vista dei controlli.

2.1 Il target di produzione ed il sistema di estrazione

Il target di produzione è composto principalmente da sette dischi coassiali in UCx opportunamente distanziati al fine di dissipare, attraverso radiazione termica, la potenza di 8kW sviluppata dal fascio di protoni (40MeV, 200μA) al fine di ottenere le 10^{13} fissioni/secondo imposte al progetto.

Figura 2.1 Prototipo di bersaglio diretto del progetto SPES, e del relativo riscaldatore.

Essi sono contenuti in una scatola (box) costituita da un tubo cavo in grafite. Il fascio di protoni attraversa due finestre in grafite, interagisce con i bersagli di produzione in UCx ed infine viene assorbito da un sistema di dumper in grafite.
Nella scatola e nei dischi contenuti in essa si deve mantenere la temperatura di esercizio di circa 2000℃, in modo da ottimizzar e l'estrazione dei prodotti di fissione. Essendo la potenza del fascio di protoni non sufficiente a mantenere il target al livello di temperatura richiesto, è necessario introdurre un dispositivo indipendente avente la funzione di riscaldare e schermare il target. Il sistema di riscaldamento supplementare deve essere in grado di sostenere il target durante i transitori, evitando improvvisi sbalzi di temperatura molto pericolosi per l'integrità strutturale dei dischi. Il riscaldatore (heater), rappresentato in Figura 2.1, è composto da un tubo molto sottile di tantalio saldato ai bordi a due ali (wings) direttamente collegate ai morsetti in rame (clamps), attraverso i quali è possibile far dissipare per effetto Joule il desiderato quantitativo di potenza al riscaldatore. Per la struttura del riscaldatore è stato scelto il tantalio, essendo questo un metallo altamente resistente alla corrosione, in grado di condurre energia elettrica e termica e di raggiungere temperature molto elevate.
Il processo di fissione nucleare produce nuclei aventi massa compresa tra gli 80 ed i 160 uma. Per la produzione di un RIB, gli isotopi dopo la fase di produzione nel target devono essere estratti e ionizzati.

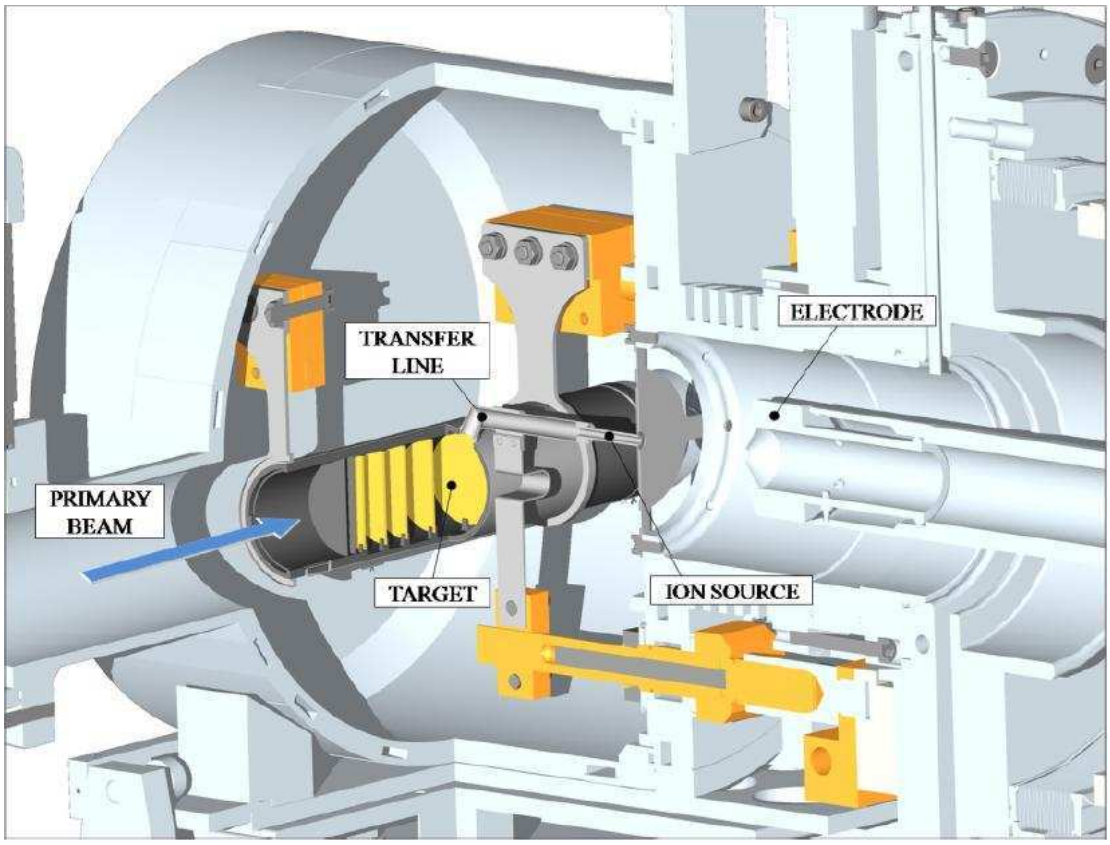

Figura 2.2 Rappresentazione del sistema di estrazione e ionizzazione del progetto SPES

La linea di trasferimento, che collega il target con la sorgente di ionizzazione, è interessata dal processo di estrazione degli isotopi. Nell'attuale configurazione, la linea di trasferimento, indicata in Figura 2.2, è un tubo sottile di tantalio saldato al riscaldatore ad un'estremità e connesso meccanicamente alla sorgente di ionizzazione.
Tra la camera target ed il sistema di trasporto del fascio è presente una differenza di potenziale ($V_{camera} - V_{transport}$) massima di 60 kV : è quindi necessario, al fine di evitare il contatto diretto, interporre un isolante elettrico (electrical insulator) come rappresentato in Figura 2.3. La differenza di potenziale presente permette quindi l'accelerazione degli ioni, formando in questo modo il fascio che verrà inviato alle sale sperimentali o post-accelerato. [6]

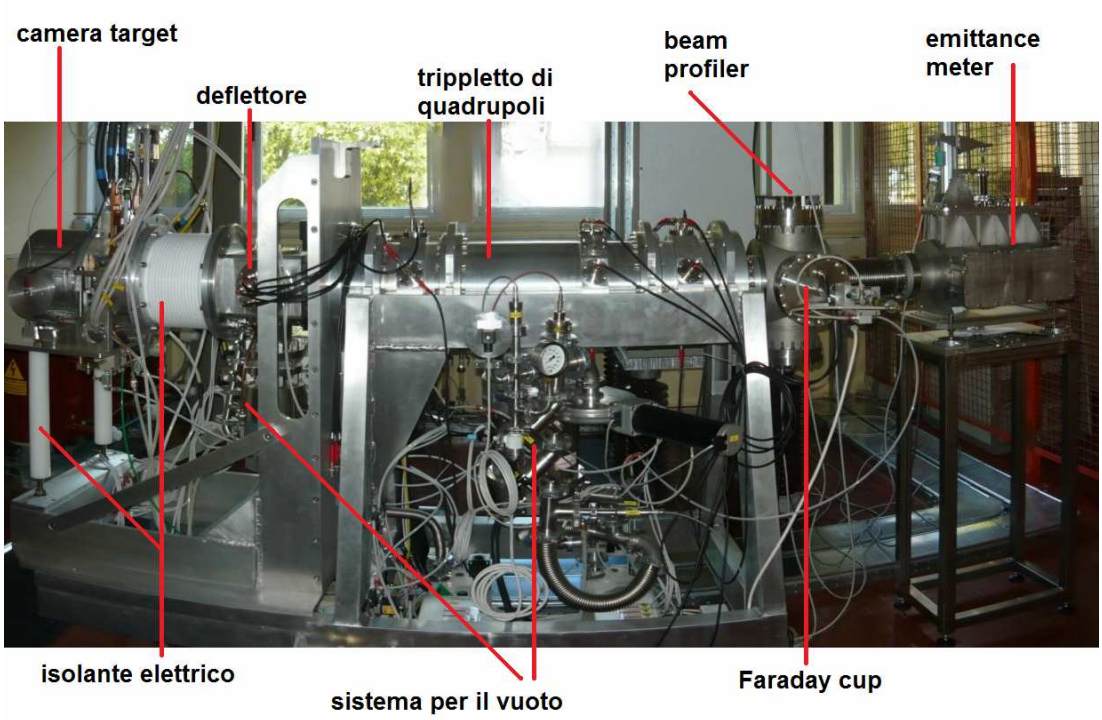

Figura 2.3 Front End offline del progetto SPES

2.2 Il processo di ionizzazione

La camera target precedentemente descritta trova alloggio all'interno del front end. Gli isotopi prodotti nel target, per divenire fasci di nuclei accelerati, devono essere necessariamente ionizzati. Il processo di ionizzazione è regolato dalle caratteristiche chimiche dell'elemento utilizzato: in generale l'energia di ionizzazione decresce lungo un gruppo della tavola periodica e incrementa da sinistra a destra attraverso il periodo. E' quindi possibile classificare gli elementi prodotti nel target in quattro gruppi distinti:

- elementi che non sono estratti dal target a causa della loro scarsa volatilità;
- elementi con bassa energia di prima ionizzazione (≤ 5eV);
- elementi con energia di prima ionizzazione compresa tra 5eV e 10eV ;
- elementi con alto valore di energia di prima ionizzazione (≤ 15eV).

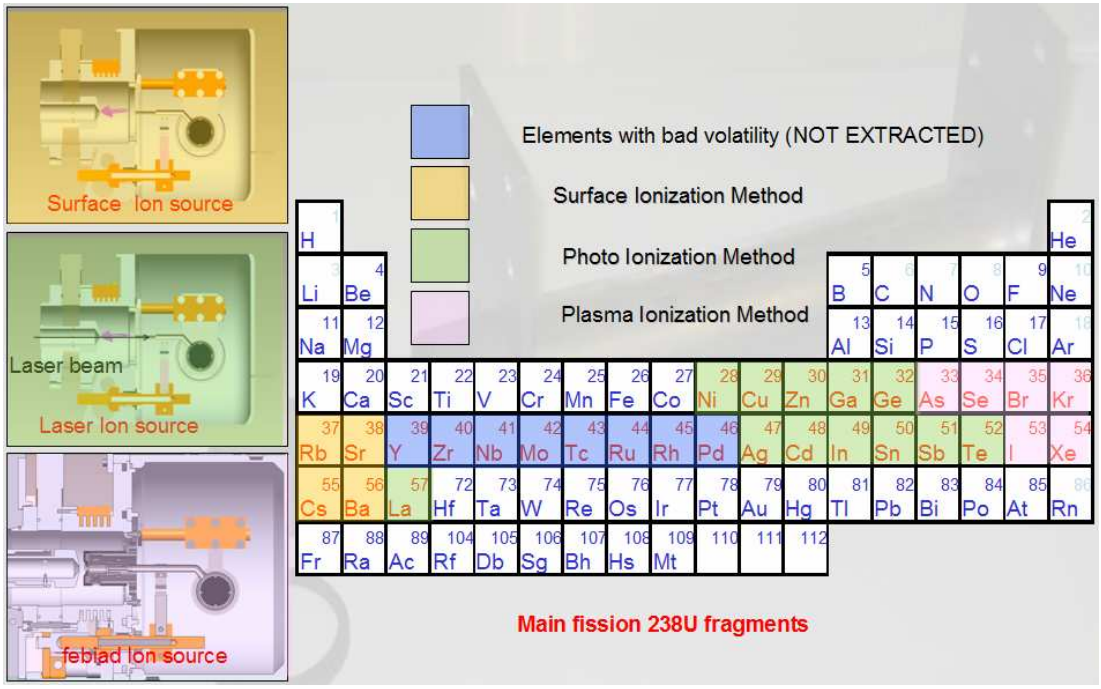

Figura 2.4 Tabella con i prodotti di fissione dell'238U, e relativo metodo di ionizzazione.

Vi sono quindi tre differenti metodi di ionizzazione e, per ognuno di essi, una particolare sorgente di ionizzazione per la produzione del fascio desiderato. Qui di seguito verranno prese in esame le tre diverse metodologie e le rispettive sorgenti di ionizzazione impiegate all'interno del progetto SPES. [6]

2.2.1 Metodo di ionizzazione superficiale

Attraverso la Surface Ion Source (SIS), la ionizzazione all'interno del complesso front end è causata dall'urto degli isotopi provenienti dal target sulla superficie interna di un tubo di tungsteno (hot cavity). A causa di tali urti, gli isotopi cedono un elettrone e vengono quindi ionizzati positivamente (ioni +1). Questo è possibile se la minima energia necessaria per rimuovere un elettrone da una superficie (funzione di lavoro) è maggiore del potenziale di ionizzazione dell'isotopo.

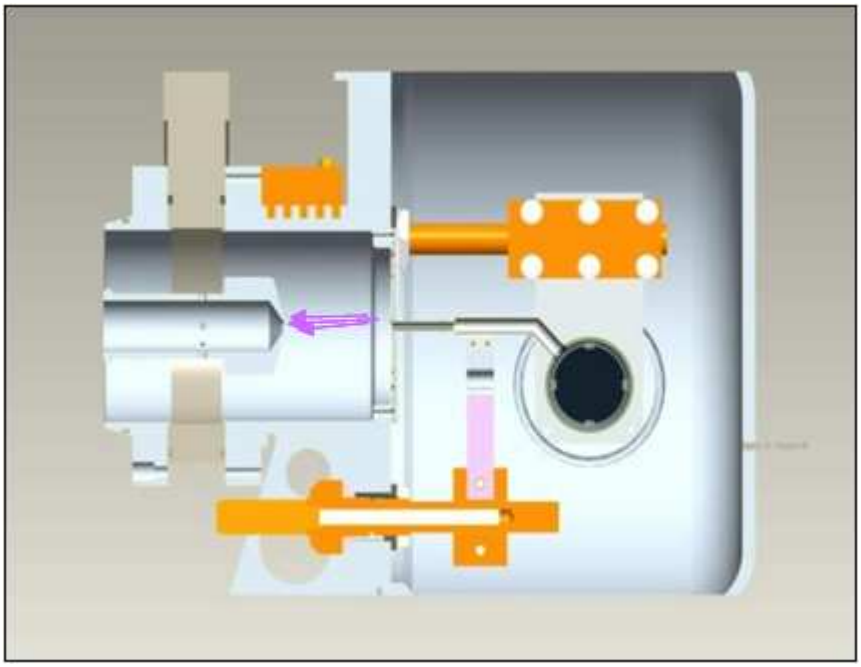

Figura 2.5 Metodo di ionizzazione superficiale

Nel caso in esame si riescono a produrre con elevata efficienza ioni positivi per elementi con potenziale di ionizzazione inferiore alla funzione di lavoro del tugsteno (pari a circa 5 eV). Per mezzo della ionizzazione superficiale è quindi possibile produrre un fascio di ioni utilizzando gli elementi caratterizzati da una bassa energia di ionizzazione, come gli elementi alcalini ed alcalini terrosi. [6]

2.2.2 Metodo di fotoionizzazione

La fotoionizzazione (LIS, Laser Ion Source), il cui principio di funzionamento è rappresentato in Figura 2.4, è oggi il più potente strumento per la produzione di fasci di ioni per le facility di tipo ISOL. Come sorgente di ionizzazione viene impiegato un fascio laser, ottenuto come sovrapposizione di 3 diversi fasci laser aventi diverse lunghezza d'onda.
Tale fascio viene diretto all'interno della hot cavity in modo da interagire con gli isotopi e fornire loro l'energia necessaria affinchè vi sia una ionizzazione selettiva. La metodologia di fotoionizzazione permette quindi di produrre soltanto gli ioni della specie di interesse, riuscendo ad ottenere un fascio nel quale le contaminazioni sono minime.

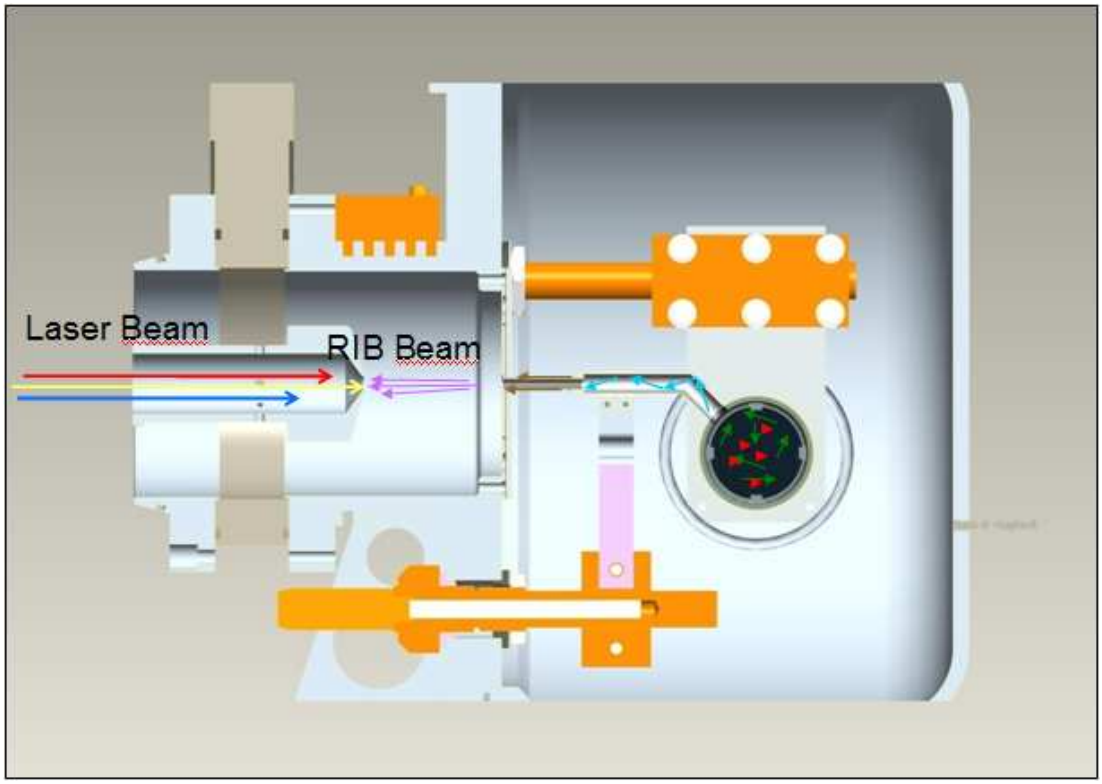

Figura 2.6 Sistema di fotoionizzazione

Per avere un'efficienza di ionizzazione elevata è di fondamentale importanza limitare e controllare il disallineamento della hot cavity causato dall'espansione termica: se la hot cavity si disallinea viene a ridursi la zona di azione del laser e di conseguenza si riduce anche l'efficienza di ionizzazione.

2.2.3 Metodo di ionizzazione al plasma

La Plasma Ion Source (PIS) permette di ionizzare anche gli elementi riportati nelle ultime due colonne della tavola periodica e caratterizzati da un alto potenziale di ionizzazione. Il principale problema di questa metodologia è la pessima selettività: l'energia media fornita (100eV), infatti, è tale da ionizzare tutti gli elementi prodotti nel target, con la conseguenza che il fascio di ioni prodotto non risulta sufficientemente puro per poter essere impiegato, ottenendo quindi efficienze minori rispetto alle tecniche laser e superficiale.
Attualmente ai Laboratori Nazionali di Legnaro è in fase di studio e progettazione una nuova sorgente di ionizzazione che permetta di utilizzare questa metodologia. [6]

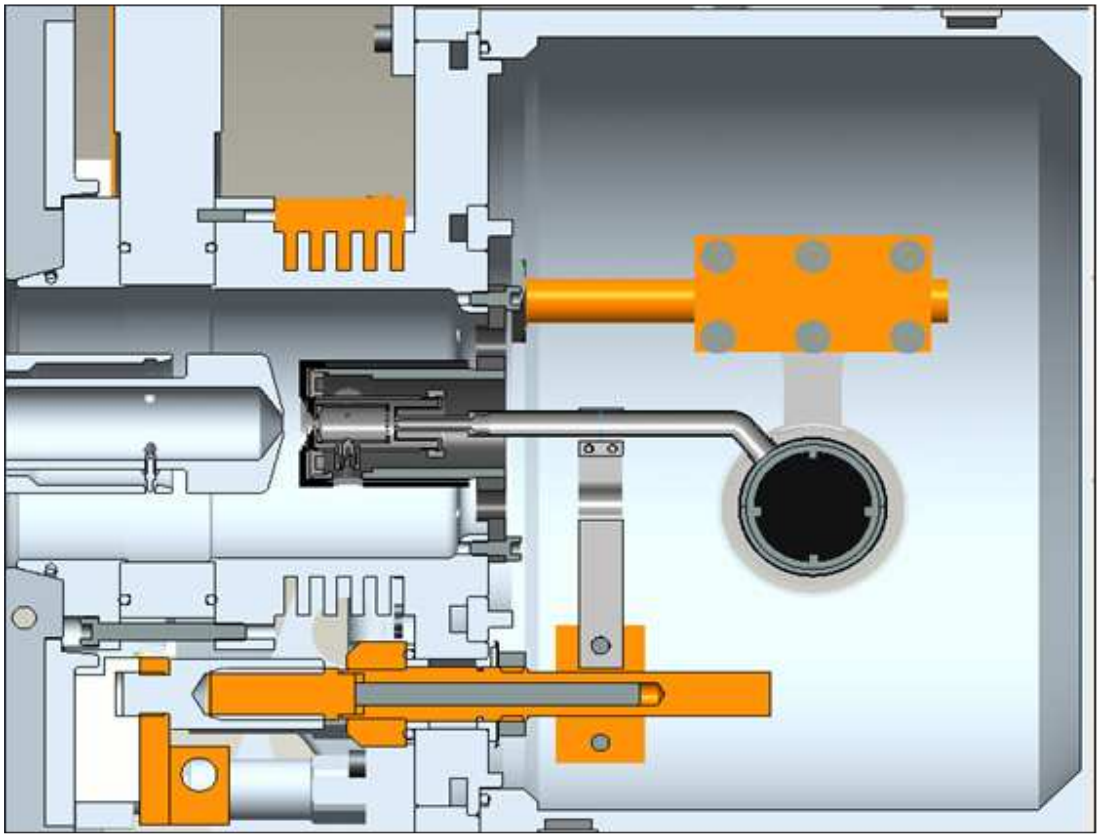
Figura 2.7 Metodo di ionizzazione al plasma

2.3 Estrazione e trasporto

I principali dispositivi che permettono il controllo e l'interazione con il fascio ionizzato sono il blocco estrattore, il sistema di deflessione ed il sistema di focalizzazione.
Il blocco estrattore, costituito dall'alimentatore di alta tensione, fornisce la differenza di potenziale necessaria ad introdurre la prima accelerazione agli ioni in uscita dalla sorgente di ionizzazione. Essi vengono risucchiati da un estrattore mobile, a potenziale di terra, la cui posizione (e quindi la distanza dalla sorgente) viene impostata tramite un PLC ed attuata da un motorino pneumatico con aggiunta di un potenziometro per il controllo della posizione. L'intero blocco contenente la camera target e tutti i dispositivi ad esso connessi si trovano ad alta tensione (30kV per la versione offline, 60kV nella versione definitiva).
Il campo elettrico, fornito dall'alimentatore di alta tensione e applicato nella camera target, permette di modificare il comportamento delle linee di forza del fascio di ioni; quindi all'aumentare del potenziale fornito dall'alimentatore si ha un progressivo assottigliamento delle linee di forza e conseguentemente un miglioramento delle caratteristiche del fascio. Per questo tipo di regolazione viene quindi richiesta una notevole precisione (e le specifiche tecniche prevedono un errore massimo di +/- 1V).

Il sistema di deflessione invece permette di correggere gli errori di disallineamento del fascio, ed è composto da 4 coppie di placche metalliche (steerer) a cui viene applicata una differenza di potenziale. La correzione dell'allineamento avviene attraverso la gestione del campo elettrostatico e si rende necessaria poichè, con l'utilizzo del front end, la dilatazione termica dei componenti della sorgente di ionizzazione incide sull'efficienza di trasporto del fascio.

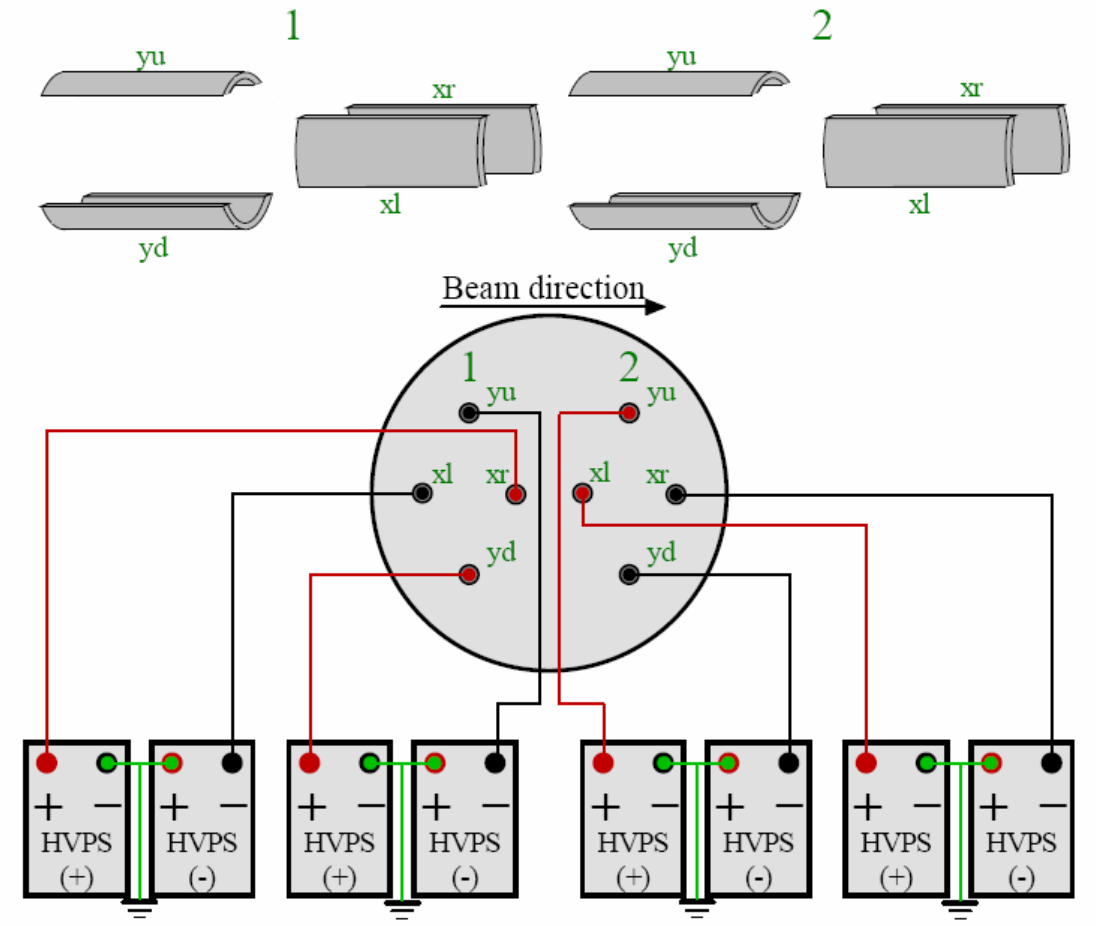

Figura 2.8 Schema del sistema di deflessione del fascio

Il sistema di focalizzazione permette infine di controllare la geometria del fascio; esso è composto da un trippletto di quadrupoli elettrostatici, ciascuno dei quali è formato da due coppie di placche. Analogamente al deflettore, applicando delle tensioni si permette di variare le caratteristiche del fascio, ma applicando una stessa tensione alle placche di ogni coppia. In questo modo non vado a variare l'allineamento del fascio, bensì solo la sua focalizzazione.

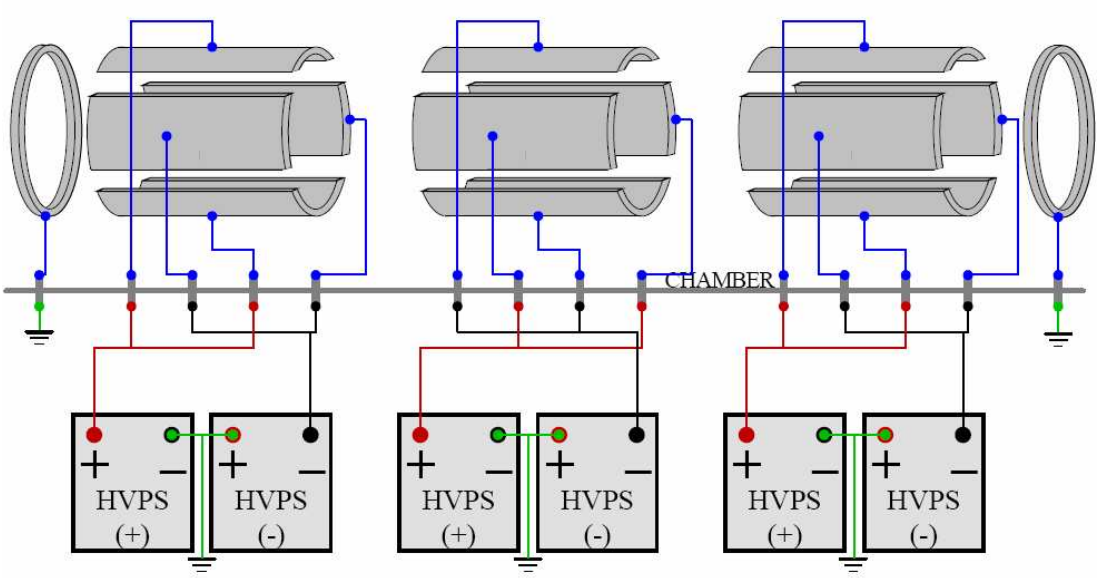

Figura 2.9 Schema del sistema di focalizzazione

Il blocco target (costituito dalla camera target e dall'estrattore) è connesso al resto della struttura attraverso degli elementi isolatori in teflon caricati con fibre di vetro; tale isolamento è necessario poichè il blocco target è soggetto all'azione della differenza di potenziale di 30kV impiegata per fornire la prima accelerazione. Tali supporti di isolamento in teflon saranno comunque sostituiti da dei supporti ceramici quando il sistema sarà messo in produzione. Per garantire l'isolamento elettrico dell'intero blocco, si rende inoltre necessario l'utilizzo di acqua demineralizzata per il sistema di raffreddamento.

Il sistema di controllo dell'intero sistema deve permettere di controllare in modo rapido e preciso le configurazioni di tensione dei sistemi per l'estrazione ed il trasporto del fascio. Si rende quindi necessario sviluppare opportuni sistemi per garantire delle caratteristiche di linearità e geometria del fascio di ioni soddisfacenti.
Il dispositivo impiegato per fornire la differenza di potenziale al sistema di estrazione degli ioni è un alimentatore Fugrack avente le seguenti caratteristiche tecniche:
- tensione massima: 65kV
- corrente massima: 2mA
- potenza erogata: 140W.

Una opportuna unità rack contiene l'alimentatore e fornisce l'interfaccia hardware per il controllo in remoto, nonchè i sistemi di abilitazione manuale e di sicurezza dei canali di alimentazione.

Figura 2.10 Il rack contenente gli alimentatori di alta tensione per deflettore, trippletto e camera target.

I dispositivi per fornire le tensioni al sistema di deflessione ed al sistema di focalizzazione del fascio di ioni sono alimentatori prodotti dalla Ultravolt. In particolare vengono utilizzati gli alimentatori serie 4A12-P4 (alimentatori unipolari positivi) e 4A12-N4 (alimentatori unipolari negativi) aventi le seguenti caratteristiche:
- tensione massima (in modulo): 4kV
- corrente massima: 1mA
- potenza erogata: 4W
- tensione di input (nominale): 12V DC.

Come per l'hardware utilizzato nel sistema di estrazione, anche gli alimentatori Ultravolt dispongono di una interfaccia hardware per il controllo in remoto. [6]

2.4 Strumenti di Diagnostica

Collegato al dispositivo di front-end offline, subito dopo al trippletto di quadrupoli, vi sono allocati alcuni strumenti di diagnostica del fascio.
Questi consistono in un primo blocco meccanico composto da una Coppa di Faraday (Faraday Cup) e da un Profilatore di Fascio (Beam Profiler), seguito da un Misuratore di Emittanza.
Tuttavia, essendo queste le apparecchiature su cui si concentra la maggior parte del presente lavoro di tesi presso i Laboratori Nazionali di Legnaro, l'argomento verrà trattato su un capitolo dedicato.

2.5 Il sistema di controllo

Il sistema di controllo dell'intero front-end è molto complesso e comprende diversi aspetti ed attrezzature.
Nell'area di lavoro sono presenti i seguenti dispositivi:
- il sistema front-end
- il rack sulla piattaforma in alta tensione
- il rack per i dispositivi posti a massa
- il trasformatore di isolamento
- il sistema di laser per la fotoionizzazione
- la griglia di protezione per isolare il sistema front-end
- il sistema per la produzione del vuoto nel blocco front-end
- il sistema per il raffreddamento della camera target
- il rack per il sistema di diagnostica del fascio.

I dispositivi di controllo e di alimentazione del sistema front end sono stati raccolti all'interno di rack opportunamente organizzati. In particolare:
1. una coppia di rack raccoglie tutti i dispositivi connessi al target aventi un potenziale fino a 30kV rispetto a terra (per la versione offline). In essi sono presenti tutti gli alimentatori di corrente impiegati per il sistema di riscaldamento della camera target, per l'anodo, e per la linea di produzione nonché la strumentazione per il monitoraggio delle temperature della camera target;
2. un ulteriore rack organizza tutti i dispositivi connessi alle parti del blocco front end aventi potenziale nullo rispetto a terra. In questo gruppo fanno parte gli alimentatori di alta tensione per i sistemi di deflessione e focalizzazione, l'alimentatore per la piattaforma di alto voltaggio ed i dispositivi embedded impiegati per la gestione dei controlli sui sistemi appena citati.
3. I PLC (Programmable Logic Controller) con relativi cablaggi ed interfacce touchscreen sono invece contenuti in parte in un terzo rack, ed in parte in un armadietto di sicurezza a muro. Essi sono dedicati alla gestione del sistema pneumatico e del sistema di sicurezza del test bench. Il sistema di sicurezza comprende l'accesso alla gabbia, i sistemi per il vuoto, lo stato degli alimentatori di alta tensione, delle valvole, e del sistema di raffreddamento.

4. un quarto rack invece è dedicato alla strumentazione per la diagnostica, e contiene tutti gli strumenti necessari ad acquisire i dati dai dispositivi sul front-end che vanno a rilevare proprietà fisiche del fascio.

Vi è però la necessità di isolare i rack contenenti i dispositivi che lavorano al potenziale di piattaforma dalle strumentazioni poste a potenziale zero e dalla rete elettrica del laboratorio. A tale scopo vengono impiegate delle connessioni in fibra ottica per la trasmissione dei dati ed un trasformatore di isolamento per disaccoppiare la linea di alimentazione della piattaforma ad alta tensione dalla rete elettrica.
L'area dove è posto il target offline è delimitata da una griglia di protezione necessaria per impedire l'accesso al blocco front end durante lo svolgimento dei test. In particolare dentro l'area di lavoro sono stati posizionati, oltre al test bench, i rack contenenti i dispositivi operanti al potenziale di 30kV ed il trasformatore di isolamento; al di fuori della zona operativa sono invece presenti i rimanenti rack contenenti i dispositivi per la gestione del sistema pneumatico, del sistema di sicurezza, del sistema di deflessione, del sistema di focalizzazione, del sistema di diagnostica e l'alimentatore che fornisce la differenza di potenziale di 30kV al blocco target.

Come lascia intuire il precedente paragrafo, il sistema di controllo e gestione della facility SPES deve presentare particolari caratteristiche, quali la capacità di integrare diverse soluzioni tecnologiche e coordinare diversi sottosistemi tra loro connessi, al fine di garantire la funzionalità dell'impianto.
L'impiego di un sistema di automatizzazione integrata realizzato attraverso un'opportuna architettura informatica, similmente ad un processo automatizzato industriale, permette di avere un'ottimizzazione nell'utilizzo delle risorse ed una flessibilità nella realizzazione degli esperimenti.
Una rete Ethernet è la via più efficace per connettere dispositivi eterogenei e distribuiti su un'ampia area; tale rete viene efficacemente pianificata seguendo una organizzazione gerarchica composta da più livelli. Ai livelli superiori vi devono essere le sale di controllo preposte alla supervisione dell'intero impianto tramite interfacce grafiche ed alla gestione delle moli di dati provenienti dai livelli sottostanti. Ad ogni livello intermedio vi deve essere il sistema di controllo per ogni struttura che costituisce l'impianto; in particolare esso ha il compito di realizzare un'architettura di controllo in grado di unificare le diverse scelte tecnologiche adottate nel livello sottostante. Al livello inferiore vi sono tutti i dispositivi che si interfacciano con le diverse strumentazioni e parti dell'acceleratore; in particolare, questo livello è caratterizzato da una eterogeneità di soluzioni hardware e software che, in base all'applicazione, possono essere sia semplici sistemi embedded oppure anche complessi sistemi VME.

Grafica e gestione dei database.
I computer, posti al livello più alto dell'architettura di controllo, realizzano due funzioni principali: l'interfaccia operatore e la gestione dei database. L'interfaccia operatore viene fornita da diversi PC, situati principalmente nella sala di controllo, ma anche dislocati in diversi punti dell'impianto per scopi diagnostici.

I pc dedicati alla sorveglianza della rete devono essere invece equivalenti ed intercambiabili, nel senso che tutte le informazioni collegate ad ogni sottosistema della facility devono essere accessibili da ogni nodo della rete di controllo. Allo stesso tempo, la capacità di cambiare i parametri di funzionamento su un determinato sottosistema deve essere concesso solamente ad un personale autorizzato. Alcune tipologie di informazioni saranno poi disponibili, solamente in modalità di sola lettura ed attraverso un web server, al di fuori della rete locale.
Un'altra operazione essenziale per il funzionamento del sistema di controllo è la gestione dei database. Tale mansione viene fornita da diversi server in configurazione ridondante, in modo da garantire operazioni fault tollerant. I database hanno il compito di memorizzare i parametri per il setup di tutti i sottosistemi costituenti l'impianto, di collezionare ed archiviare i dati utili per l'analisi delle operazioni, compresi gli allarmi e tutti i tipi di eventi che possono essere significativi per i problemi di correlazioni e di diagnosi.

Hardware per il controllo di basso livello
Il controllo di ogni sottosistema viene basato su dispositivi dedicati: ad esempio sistemi di PLC opportunamente configurati vengono utilizzati in sistemi dove l'affidabilità ha una notevole rilevanza. Molti dei modelli di PLC attualmente in commercio possiedono una interfaccia Ethernet; di conseguenza l'integrazione, dal punto di vista hardware, risulta semplice. Comunque, a meno che i PLC non vengano usati assieme ai software di supervisione forniti dal proprio produttore, risulta necessario impiegare un gateway per poterli interfacciare al sistema di controllo, poichè non vi è ancora disponibile un protocollo di comunicazione standard.
Nei casi in cui il sistema di controllo richieda la realizzazione di funzioni complesse, oppure vi siano severe limitazioni in termini di tempi di risposta, lo standard VME rappresenta la soluzione migliore, grazie alla disponibilità di moduli con processori ad alte prestazioni ed un gran numero di schede I/O provenienti da diversi produttori.
La scelta del sistema operativo da impiegare influenza fortemente le prestazioni globali dei sistemi di controllo. Una possibile soluzione può essere l'impiego di un sistema operativo real-time come VxWorks, poichè le caratteristiche che lo contraddistinguono gli permettono di essere un'eccellente base per implementare task critiche.

Il software di controllo
Sebbene possono essere usate diverse tecnologie hardware nel controllo dei sottosistemi di SPES, è obbligatorio adottare un modello comune per lo scambio dei dati. Da studi effettuati e dall'esperienza riportata da numerosi istituti di ricerca sparsi in tutto il mondo, è stato deciso di basare il sistema di controllo sul software EPICS. Il concetto base dell'architettura EPICS è l'avere una memoria dati distribuita; le variabili di processo, definite mediante una particolare struttura di record, sono raccolte dal campo (o trasferite al campo) secondo un meccanismo (scansione periodica, interlock, ecc..) configurabile dall'utente. Una volta che la variabile di processo è stata memorizzata nel database, essa diventa accessibile, attraverso il suo nome, da ogni nodo della rete.

L’infrastruttura che localizza le variabili e le rende disponibili alle richieste delle applicazioni client viene chiamata Channel Access.
Sul lato client, esistono molte utility dedicate alla creazione di un sistema di controllo pienamente operativo, senza avere la necessità di sviluppare alcun codice applicativo. Ad esempio vi sono programmi per la gestione degli allarmi e tool che forniscono servizi di backup&restore. Il progettista del sistema può sviluppare facilmente la propria applicazione client; l’unica regola che deve osservare è il rispetto del protocollo di comunicazione del Channel Access. In Figura 2.4 è rappresentato un esempio di possibile architettura di un sistema di controllo basato su EPICS.

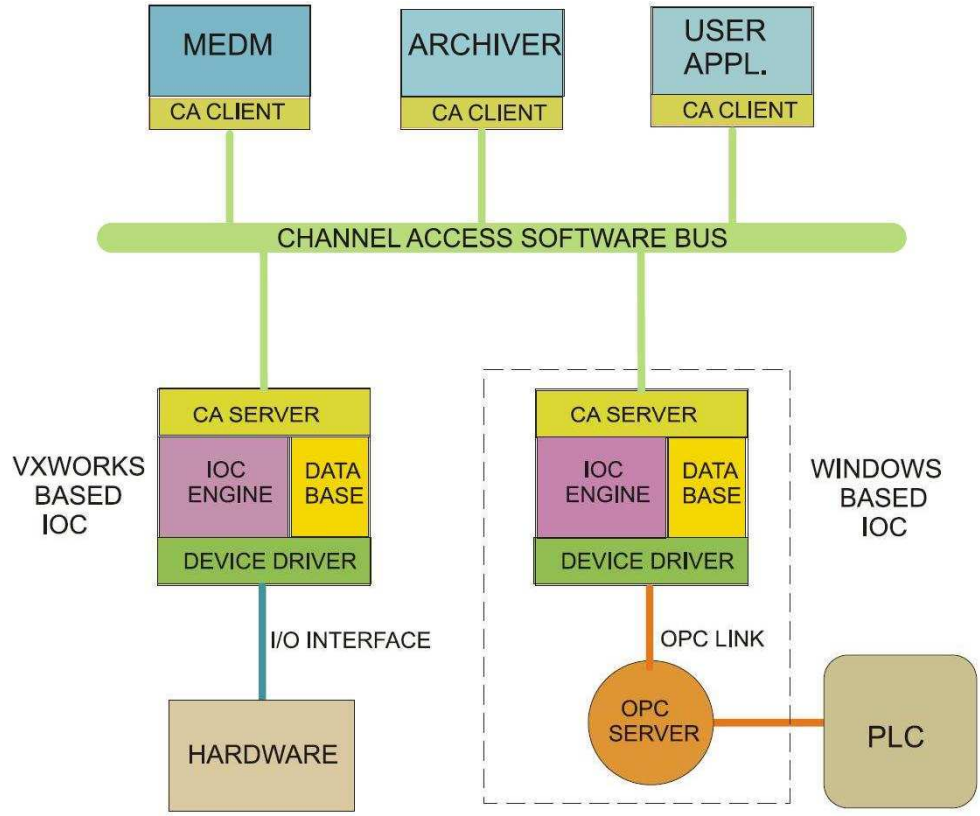

Figura 2.11 Esempio di architettura di un sistema di controllo basato sul software EPICS.

La rete distribuita di controlli, dedicata alla gestione e al monitoraggio dei vari dispositivi, è stata realizzata seguendo l'organizzazione gerarchica precedentemente descritta. L'ambiente EPICS costituisce il cuore di questo sistema di controllo, mentre il Channel Access rappresenta il middleware che permette l'integrazione tra i diversi elementi hardware e software presenti ai vari livelli della struttura (controllori IOC su cui vengono eseguiti algoritmi di controllo, software per interfacce LabVIEW e CSS, crate VME con S.O. VxWorks, ecc...).

Per poter utilizzare le informazioni fornite dal software EPICS all'interno dell'ambiente LabVIEW, si è resa necessaria l'installazione di opportune librerie (Shared Memory Library) sviluppate dalla comunità EPICS.
Per quanto concerne la comunicazione tra software EPICS e PLC, come già accennato sopra, non è stato possibile realizzare una connessione diretta di questi ultimi al Channel Access. I segnali di interlock, provenienti dai PLC, vengono quindi acquisiti, elaborati, e tradotti in variabili EPICS attraverso dei controllori microIOC prodotti dalla Cosylab (utilizzati anche per controllare gli alimentatori per i sistemi di estrazione, deflessione e focalizzazione). La comunicazione tra PLC e IOC è unidirezionale, poichè attualmente non vengono impiegati segnali provenienti dal software EPICS all'interno dei programmi di controllo gestiti da PLC.

2.6 Conclusioni

In questo capitolo è stata analizzato il front-end della facility SPES sia dal punto di vista meccanico che controllistico. Si è potuto osservare come esso sia un sistema complesso, eterogeneo e distribuito. Nel realizzare un'architettura di controllo, risulta quindi necessario avere un'organizzazione delle informazioni e dei dati ordinata, in modo tale da poter coordinare, in modo ottimale, tutti i sottosistemi, i dispositivi e le strumentazioni all'interno dell'impianto.
La principale problematica da risolvere risulta però essere l'eterogeneità delle soluzioni software ed hardware presenti. Come è stato precedentemente descritto, in una situazione in cui vengono impiegati strumenti dalle caratteristiche molto diverse, l'adozione di un software di controllo unico, che permetta l'integrazione di tutte queste scelte tecnologiche, diventa di importanza cruciale. In commercio sono disponibili diverse soluzioni che rispondono a queste esigenze. Tra queste spicca il software EPICS, un insieme di strumenti, librerie ed applicazioni dedicate allo sviluppo di sistemi di controllo distribuiti in tempo reale.
Secondo valutazioni eseguite da enti indipendenti, questo framework, nato da un progetto Open Source e supportato da una nutrita community, risulta essere molto competitivo dal punto di vista di robustezza e velocità di calcolo. Inoltre esso viene attualmente utilizzato, all'interno di importanti progetti, in numerosi laboratori di ricerca per la fisica nucleare. In base alle caratteristiche tecniche fornite, all'ottimizzazione dei costi che ne deriva dall'usare un software libero e privilegiando la collaborazione tra laboratori (aspetto molto importante in un ambito come quello della ricerca scientifica), è stato scelto EPICS come software di controllo per la facility SPES.

Capitolo 3 - EPICS

EPICS (Experimental Physics and Industrial Control System) nasce dalla collaborazione tra alcuni laboratori di ricerca europei ed americani con lo scopo di sviluppare strumenti software orientati alla realizzazione di sistemi di controllo di grandi macchine acceleratrici e apparati strumentali per la fisica.

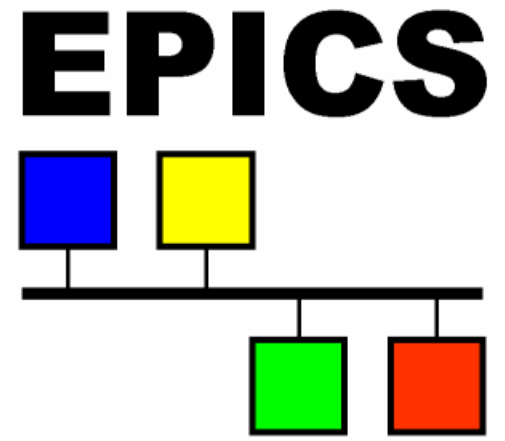

Figura 3.1 Il logo di EPICS

Esso consiste in una architettura per sistemi di controllo, costruita su un efficiente protocollo per il trasferimento dei dati (Channel Access) basato sul modello Client/Server, ed un database distribuito in tempo reale.
Per analogia con la terminologia usata nei database, una variabile di processo (PV, Process Variable) è trattata come elemento di una struttura più complessa detta Record. Il record contiene, oltre al valore della variabile stessa, un numero, anche elevato, di attributi che descrivono le proprietà di quella variabile in una data applicazione.
Esempi di attributi che compongono i record possono essere:

- Unità di misura (Engineering Units)
- Timestamp
- Range operativi di lavoro

- Soglie di allarme e tipo di allarme
- Collegamenti a driver di dispositivi hardware
- Stringhe di descrizione

Il protocollo Channel Access implementa il livello di comunicazione tra client e server in modo tale che l'accesso a una variabile di processo avviene in modo trasparente all'applicazione: il processo client non conosce a priori quale sia l'indirizzo del server che gli fornirà il valore della variabile richiesta (l'invocazione della PV avviene semplicemente per nome).

Poiché all'interno di una rete di controllo vi possono essere un numero elevato di applicazioni server che forniscono accesso a migliaia di variabili ciascuna, e diverse applicazioni client che richiedono di accedere (anche contemporaneamente) alle stesse informazioni è essenziale che il protocollo di distribuzione dei dati (middleware) sia affidabile. Il Channel Access è unanimemente riconosciuto, all'interno della comunità degli addetti al controllo di acceleratori, essere tra i più veloci e robusti middleware oggi disponibili.

Un altro concetto fondamentale nell'architettura di EPICS è quello di IOC (Input Output Controller).
Con IOC si intende genericamente un computer (spesso un embedded controller) cui è demandato il compito di acquisire le variabili di processo dai dispositivi hardware e memorizzarle nell'istanza locale del database distribuito.
L'IOC contiene dunque il software (iocCore) necessario alla gestione del database e implementa la funzione Channel Access Server necessaria a trasferire il valore delle variabili ai processi Client che le richiedono.
Nella filosofia di EPICS il database non è solamente uno storage di variabili ma è una struttura logica che consente di implementare degli algoritmi di controllo attraverso i link tra i vari record. L'IOC è dunque anche la sede di processi di controllo configurati attraverso il database stesso; come si vedrà più avanti, questo non è l'unico modo per realizzare un controllo ma è quella più aderente al concetto originale di programmazione in EPICS e, in definitiva, anche il più veloce. Istanze di *iocCore* possono anche essere eseguite sotto forma processi su di un computer *non dedicato* (ad esempio, su un computer che svolge anche altre mansioni), ed in tale caso vengono definite *Soft IOC*.
I computer su cui l'IOC gira possono essere di vario tipo:

- calcolatori basati sullo standard VME, con sistema operativo VxWorks o RTEMS
- Pc con Windows, Linux o RTEMS
- Apple con OSX
- Workstation UNIX con Solaris

EPICS include anche molti strumenti sul lato client: tra questi vi sono programmi per la creazione e l'utilizzo di interfacce grafiche (ad es. MEDM), per la gestione degli allarmi (ad es. Alarm Manager), per l'archivazione delle

PV (ad es. Archiver), e diverse applicazioni orientate all'analisi dei dati e fisica degli acceleratori. [7]

3.1 Il protocollo Channel Access

Come già accennato in precedenza, questa infrastruttura realizza un protocollo di comunicazione comune tra tutti i dispositivi (client e server) collegati alla rete di controllo, fornendo gli strumenti necessari per localizzare e reperire qualsiasi variabile di controllo presente all'interno della memoria dati distribuita.
Le caratteristiche basilari che consentono a questo protocollo di gestire con successo l'intero sistema sono alla base del suo funzionamento, e possono essere riassunte nei seguenti punti:

- i programmi client effettuano un broadcast del nome delle variabili per trovare il server in cui esse sono disponibili;
- procedure di sicurezza (Channel Access Security) possono essere applicate per limitare l'accesso alle variabili di processo;
- i client possono essere obbligati ad aspettare il termine di una richiesta in corso prima di procedere;
- i client possono impostare delle procedure di *monitor* sulle PV, in modo da ricevere una notifica da parte del server non appena il valore della variabile cambia.

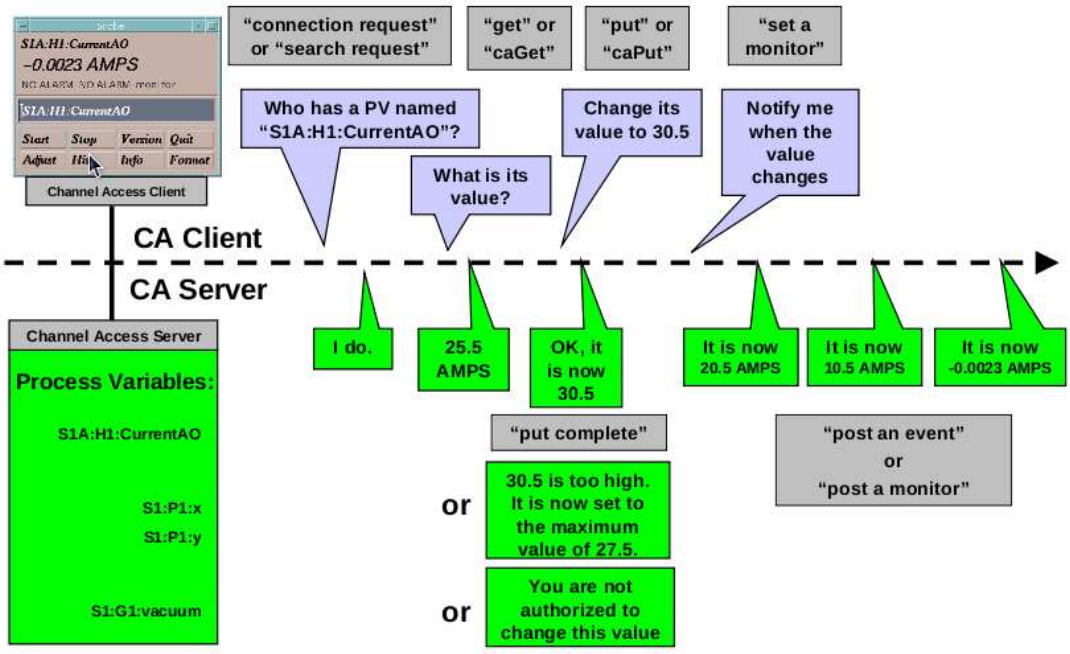

Figura 3.2 Esempio di funzionamento del protocollo Channel Access.

Nell'esempio della figura, un CA client effettua una *search request* per localizzare quale sia il CA server che dispone della process variable richiesta con un broadcast sulla rete, e rimane in attesa di una risposta da parte dell'host interessato. Il client effettua poi una richiesta di tipo *caGet* per

venire a conoscenza del valore della variabile, ed ottiene in risposta il valore attuale. Successivamente, il client effettua una operazione *caPut* per scrivere un nuovo valore su quella variabile, ed ottiene in risposta una conferma di avvenuta operazione, *put complete*. Ulteriori possibili risposte potrebbero essere state l'impostazione di un valore diverso, o una negata autorizzazione al cambio di valore. Infine il client inizia una procedura di *monitor* sulla variabile, e viene quindi informato automaticamente dal CA server ogni volta che la variabile monitorizzata cambia valore.

Il Channel Access tuttavia è modificabile, e può essere configurato a seconda delle esigenze attraverso le variabili di ambiente EPICS indicate in Tabella 3.1. Nel caso di sistemi operativi Linux, esse generalmente vengono definite all'interno del file *profile* presente nella directory */etc*; nel caso del sistema operativo VxWorks queste possono esservi inserite tramite il comando *putenv*. Se tali variabili non sono definite nell'ambiente del sistema operativo, vengono allora caricati i valori di default dal sistema EPICS.

Nome Variabile	**Range**	**Default**
EPICS CA ADDR LIST	{N.N.N.N N.N.N.N:P ...}	< none >
EPICS CA AUTO ADDR LIST	{YES, NO}	YES
EPICS CA CONN TMO	r > 0.1 sec	30
EPICS CA BEACON PERIOD	r > 0.1 sec	15
EPICS CA REPEATER PORT	i > 5000	5065
EPICS CA SERVER PORT	i > 5000	5064
EPICS CA MAX ARRAY BYTES	i ≥ 16384	16384
EPICS CA MAX SEARCH PERIOD	r > 60 sec	300
EPICS TS MIN WEST	−720 < i < 720 min	360

Tabella 3.1: Le variabili d'ambiente che configurano il protocollo Channel Access.

Il Channel Access è implementato sui protocolli TCP, IP, ed UDP, e quindi è richiesta una appropriata configurazione di rete per ogni host (indirizzo di rete, indirizzo di host, subnet mask).
Per quanto riguarda le porte che identificano il servizio, i server EPICS ricevono pacchetti di rete sulle porte 5064 (Channel Access Server Port) e 5065 (Channel Access Repeater Port). Questi due valori possono essere riconfigurati modificando il valore di due particolari variabili d'ambiente:

- EPICS_CA_SERVER_PORT → porta del Channel Access Server
- EPICS_CA_REPEATER_PORT → porta del Channel Access Repeater

Generalmente la necessità di modificare questi parametri sorge quando, in un sito, si decide di realizzare due o più sistemi di controllo indipendenti sulla stessa rete. Ad esempio, nel caso in cui si abbia un sistema di controllo di produzione ed un sistema di controllo di test, è desiderabile che le variabili di

processo dei due sistemi non collidano tra loro. Questo è possibile impostando dei diversi numeri di porta per i Channel Access Server e Channel Access Repeater.
Sotto Linux la configurazione delle variabili d'ambiente può essere eseguita inserendo, all'interno del file */etc/profile*, i seguenti comandi:

export EPICS_CA_SERVER_PORT= numero di porta desiderato
export EPICS_CA_REPEATER_PORT= numero di porta desiderato

Quando un Channel Access client si connette al canale, deve prima di tutto determinare l'indirizzo IP del server su cui le Process Variable risiedono. Per realizzare ciò, il client manda una richiesta di risoluzione del nome (search request) ad una lista di indirizzi di server; questi indirizzi di destinazione possono essere indirizzi IP unicast o indirizzi IP broadcast. Ogni search request contiene una lista di nomi di variabili di processo; se uno dei server raggiungibili conosce l'IP del Channel Access server che fornisce una o più variabili, questo manda una risposta al client contenente l'indirizzo IP e la porta del server desiderato.
Durante l'inizializzazione, il Channel Access costruisce una lista degli indirizzi dei server di destinazione. Tale processo avviene analizzando le interfacce di rete collegate all'host:

- per ogni interfaccia trovata che risulta collegata ad una sottorete IP abilitata in broadcast, l'indirizzo (broadcast) di tale rete viene aggiunto alla lista;
- per ogni interfaccia punto-punto trovata, l'indirizzo di destinazione di quel link viene aggiunto alla lista.

La lista automatica dei server può essere disabilitata se la variabile d'ambiente EPICS CA AUTO ADDR LIST è definita e configurata come segue:

EPICS_CA_AUTO_ADDR_LIST=NO

In questo modo si vincola l'host ad analizzare solamente le interfacce di rete relative agli indirizzi IP indicati nella variabile d'ambiente EPICS_CA_ADDR_LIST.
Gli indirizzi IP, in quest'ultima variabile, possono essere indicati sia in forma numerica sia sotto forma di nome dell'host, se il sistema operativo "locale" fornisce il servizio di traduzione da host name ad IP address.
Inoltre:

- se vi sono più indirizzi IP, questi devono essere separati da uno spazio;
- nel caso in cui un client debba comunicare con due server in ascolto su due porte differenti, viene usata la particolare sintassi:

indirizzo IP:nome porta

In generale, se non viene specificata alcuna porta, il sistema utilizzerà il valore indicato nella variabile EPICS_CA_SERVER_PORT. [6, 8, 9]

3.2 Input Output Controller

All'interno dell'ambiente EPICS viene definito come IOC (Input Output Controller) l'applicazione costituita dall'iocCore, un set di routine usate per definire le Process Variable ed implementare gli algoritmi di controllo Real-Time. L'IOC rappresenta quindi il modulo software fondamentale su cui è costruito il controllo.
Esso è composto da diverse routine dedicate alla definizione dei meccanismi di comunicazione (sia verso il Channel Access che verso i dispositivi hardware ad esso collegati), nonchè alla memorizzazione ed alla gestione delle Process Variable.

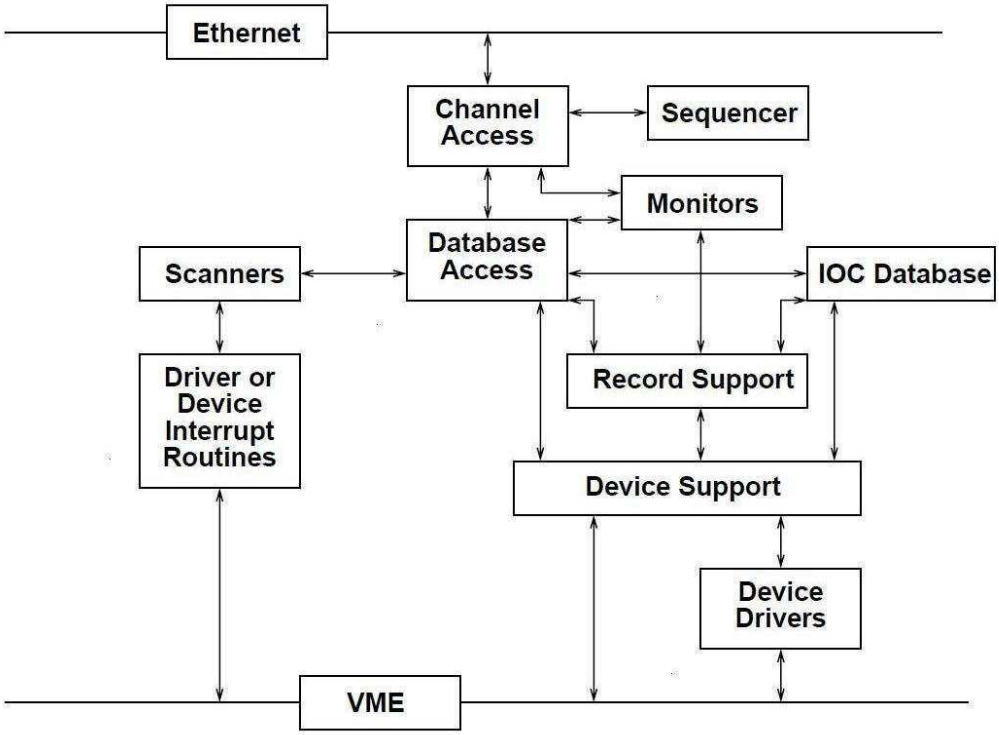

Figura 3.3 Schema degli elementi costituenti un IOC elementare.

In Figura 3.3 sono rappresentati i software che implementano le funzionalità base di un IOC. In particolare è possibile individuare:

- l'IOC Database: è la memoria dove risiedono le Process Variable definite all'interno dell'applicazione IOC. Tali variabili di processo saranno poi disponibili per tutti i client ed IOC, presenti nella rete di controllo, che ne fanno richiesta;
- il Database Access: è la routine preposta alla gestione degli accessi verso l'IOC Database. Essa però non gestisce gli accessi alla memoria dati eseguiti dal Record Support e dal Device Support;
- lo Scanner : è il meccanismo con cui vengono definite le modalità di aggiornamento dei valori delle Process Variable;
- il Record Support ed il Device Support: sono le routine dedicate alla gestione dei Record e dei Support sviluppati da terze parti. In

particolare vengono usati quando sono definite rispettivamente nuove tipologie di record e nuovi moduli per realizzare la comunicazione con i dispositivi hardware;

- il Device Driver : gestisce gli accessi verso un dispositivo esterno collegato all'IOC. Il Device Driver viene usato quando il metodo di accesso verso un particolare hardware è molto complesso rispetto a quello che un Device Support può elaborare;
- il Channel Access: in questo caso identifica il software di interfaccia tra il controllore ed il canale di trasmissione;
- il Monitor : è una routine utilizzata per supervisionare l'aggiornamento dei valori delle Process Variable e definire il meccanismo di callback per la trasmissione dei dati. La tecnica denominata "reporting by exception", o callback, si basa sulla seguente procedura: ogni volta che un client risulta interessato ad un certo dato situato in un particolare server, tale server esegue la notifica del dato solamente quando questo cambia. Così facendo, non solo si minimizza il traffico sulla rete, ma è anche possibile monitorare la salute del server;
- il Sequencer : permette di realizzare una macchina a stati finiti all'interno dell'applicazione IOC. [6, 8]

All'interno dell'intero sistema di controllo non vi possono essere variabili di processo aventi lo stesso nome; poichè il reperimento di tali parametri avviene proprio con una ricerca "per nome", questo vincolo risulta fondamentale per evitare "sovrapposizioni" di variabili. Si ha così che l'utente può definire arbitrariamente il nome di ogni Process Variable, ma questo deve essere univoco tra tutte le variabili presenti nella rete. Ogni IOC, in fase di inizializzazione, effettua un controllo sull'univocità dei nomi delle variabili memorizzate al suo interno. Il problema però si può presentare quando vi sono due Process Variable, aventi lo stesso nome, memorizzate in due IOC differenti e collegati al medesimo sistema di controllo. Una soluzione, per evitare l'insorgere di questo tipo di problemi, è l'assunzione di un'opportuna Naming Convention per la definizione dei nomi delle variabili.
L'insieme delle Process Variable, presenti nella memoria di un IOC, stabiliscono un database di stati, informazioni e variabili di controllo. L'implementazione di una Process Variable, all'interno di un database, avviene per mezzo di particolari strutture definite *record*. Nell'ambiente EPICS, con il termine *record* si identifica un oggetto avente le seguenti caratteristiche:

- un nome univoco;
- un proprio comportamento definito dalla sua tipologia (Record Type);
- alcune particolari proprietà definite per mezzo di campi (Fields);
- è opzionalmente associato ad un hardware attraverso un supporto (Device Support);
- è collegato ad altri record per mezzo di collegamenti (Links).

Si ha quindi che un database può essere visto sia come un insieme di Process Variable, sia come un elenco di record. In particolare si può affermare che ogni Process Variable viene definita tramite un record :

Process Variable $\Leftrightarrow$ Record

Si fa notare che il nome di una Process Variable coincide con il nome del record col quale tale variabile viene implementata. Il vincolo sull'univocità del nome di una variabile di processo si ripercuote quindi sull'univocità del nome del record. I record possono scambiare informazioni con altri record o con l'hardware connesso, effettuare calcoli, abilitare o disabilitare altri record, aspettare segnali dall'hardware (interrupt), verificare se i valori acquisiti sono nel range preimpostato ed eventualmente inoltrare un allarme.
In generale le operazioni che un record può effettuare dipendono dalla sua tipologia (Record Type) e da come sono configurati i suoi campi (Field).
Vi sono però alcuni concetti base che accomunano le varie tipologie di Record:

Le specifiche di scansione
Un record viene processato quando questo elabora i suoi dati ed esegue ogni azione collegata ad essi. Ad esempio, quando un Analog Output Record viene elaborato, esso recupera il valore che deve essere impostato in uscita, lo converte e successivamente lo scrive in una specifica locazione di memoria. Ogni record deve specificare il proprio metodo di scansione, il quale determina quando questi viene elaborato. Il software EPICS dispone di tre differenti modalità di scansione:

- Scansione Periodica (Periodic Scanning): viene usata per scansioni ad intervalli di tempo regolari. Per questa modalità di scansione, vi sono differenti opzioni di default disponibili: 10sec, 5sec, 2sec, 1sec, 0.5sec, 0.2sec, 0.1sec. Nel caso in cui serva un particolare periodo diverso da quelli standard, è possibile aggiungerlo tra le opzioni modificando un particolare file, il Database Definition file. Il range di valori non può però essere più piccolo di 0.015sec, ovvero 66Hz, la massima frequenza disponibile sul sistema.
- Scansione ad Evento (Event Scanning): viene usata per elaborare il Record all'arrivo di interrupt o da eventi definiti dall'utente.
- Scansione Passiva (Passive Scanning): succede quando i record collegati al Passive Record vengono scansionati o quando viene inserito un valore nel Passive Record attraverso una routine di accesso al database.

Si ha poi che l'utente, per le scansioni periodiche e "ad evento", può controllare l'ordine con cui un insieme di record viene elaborato usando un particolare meccanismo: il Phase. Tale strategia permette, configurando un opportuno campo dei record, di organizzare la successione delle scansioni dei record che verrebbero elaborati nello stesso momento. In questo modo, ad esempio, è possibile fornire ai record dipendenti da altri record dei valori opportunamente aggiornati.
E' inoltre possibile controllare, soprattutto nel caso di Event Scanning, la priorità con la quale un record viene processato.
In aggiunta ai meccanismi di Scanning e Phase, vi sono particolari link, chiamati Data Link e Forward Processing Link, che possono essere usati per causare l'esecuzione di altri record una volta che il record, contenente uno di tali link, viene elaborato. Ad esempio, quando il Forward Processing link di un record contiene l'indirizzo di un secondo record, quest'ultimo viene

processato dopo che il primo record viene elaborato. In questo modo è possibile concatenare i record affinchè eseguano delle operazioni articolate.
Esiste inoltre un particolare record, il Fanout Record, preposto alla sola propagazione del comando di processamento dei record; principalmente questo viene impiegato quando vi sono più di un record da eseguire alla fine dell'elaborazione di un particolare record. [6, 10]

Le specifiche di indirizzo

Questi parametri specificano i link tra i record e le locazioni di memoria su cui eseguire operazioni di scrittura e lettura. Per i record sono disponibili tre tipologie di Address Specification:

- Hardware Address:
  sono utilizzati per specificare le connessioni di ingresso e uscita con dispositivi hardware e forniscono le informazioni necessarie all'IOC per interfacciarsi con le strumentazioni. Attualmente vi sono moduli per realizzare la comunicazione tra IOC ed otto differenti tipi di I/O bus: VME, Allen-Bradley, Camac, Gpib, Bitbus, INST, VXI ed RF. I comandi da fornire, per indicare l'indirizzo di memoria desiderato, variano in funzione del bus utilizzato.
- Database Address:
  sono usati per specificare diverse tipologie di link, come ad esempio Input link e Forward Processing link. Il formato di ogni Database Address è il medesimo:

  RecordName.FieldName

  dove con RecordName viene indicato il nome del record a cui si fa riferimento e con FieldName il nome del campo del record. Il nome del record e del campo sono Case Sensitive e, nello sviluppo delle applicazioni, i nomi dei campi vengono sempre indicati in maiuscolo. Se non viene specificato alcun nome di un campo, di default viene utilizzato il campo VAL del record, il quale specifica il valore elaborato quando il Record viene eseguito. Per i Fanout Record non vi è invece la necessità di includere il nome del campo poichè, data la natura di tali record, non viene mai restituita alcuna informazione quando questi vengono eseguiti.
  Si ha poi che, nel caso in cui due record operanti con due tipi di dato differenti siano collegati tra loro, la conversione tra i tipi di dato deve essere specificata manualmente. Solamente quando i record interessati elaborano dati di tipo Floating Point ed Integer, la conversione viene eseguita in automatico.
  Vi è poi la necessità di specificare, per ogni record, come organizzare ed eseguire i propri input ed i propri output, al fine di elaborare i valori con le corrette specifiche. Questo meccanismo viene regolato attraverso la specifica di un'opportuna Device Support routine in un particolare campo del record: il DTYP (Device Type) field.
  Per i record, vi sono Device Support routine per comunicazioni tra EPICS e dispositivi hardware e Device Support routine per gestire gli input e gli output tra record. In particolare, in quest'ultima categoria, è possibile individuare due routine preposte alla gestione dei flussi di ingresso e uscita di un record:

Raw Soft Channel → ritrova il valore di input ed esegue una specifica conversione lineare sul valore;
Soft Channel → legge direttamente il valore nel campo VAL e non esegue alcuna conversione.
In questo modo si ha la possibilità di effettuare operazioni di scaling in automatico su particolari variabili di processo.
In generale, i link field possono anche riferirsi a variabili situate in database distinti, o in database situati in diversi IOC. I record residenti su differenti IOC vengono connessi attraverso il Channel Access, ed ogni link che si riferisce ad un record posto in un altro IOC viene chiamato Channel Access link. Questi particolari link non sono presenti in fase di realizzazione del database, ma vengono creati quando il database viene inizializzato. Nello specifico, quando in un campo di un record viene indicato un secondo Record non presente nel database locale, allo startup dell'applicazione IOC tale variabile viene cercata all'interno della rete EPICS e, se trovata, viene creato il Channel Access link.

- Costant:
  alcuni tipi di campo, come ad esempio gli Input Link field e i Desired Output Link field, possono specificare una costante al posto di un Hardware Address o un Database Address. Una costante, che non è in realtà un vero e proprio indirizzo, può essere una variabile Integer in qualsiasi formato (decimale, binario, ottale, ecc..) oppure un Floating Point. Il valore del campo viene inizializzato alla costante voluta quando il database viene inizializzato e, durante il run-time, tale valore può essere cambiato attraverso una Database Access routine. [6, 10]

Le specifiche di conversione

I parametri di conversione sono usati per convertire i dati di trasmissione in segnali significativi. In particolare i segnali discreti richiedono una conversione tra livelli e stati, mentre segnali analogici richiedono la conversione tra livelli e unità ingegneristiche. Per realizzare la conversione del segnale di un record, vi sono appositi Record Field che ne permettono la configurazione.
In questo modo è possibile fornire, all'operatore e alle applicazioni client, dei valori in unità di misura significative.

Le specifiche di allarme

Vi sono due proprietà che definiscono le caratteristiche di un allarme: lo status e la severità dell'allarme. In particolare la severità permette di dare un peso ad un particolare stato d'allarme.
Ogni record contiene il suo status di allarme attuale e la corrispondente severità per tale status. La task di scansione, preposta alla supervisione degli allarmi, è capace di generare un messaggio per ogni cambio di status.
Le tipologie di allarme disponibili ricadono nelle seguenti categorie:

- Scan Alarm:

vengono generati quando un record non viene inserito nella lista di

scansione desiderata, oppure quando la Scan Task citata precedentemente lo rileva.

- Read/Write Alarm:
  i Read Alarm vengono generati quando un valore viene acquisito dall'hardware o dal campo di un record, ma la routine di lettura fallisce. I Write Alarm sono simili ai precedenti, ma avvengono quando è la routine di scrittura a fallire.
- Limit Alarm:
  per alcune tipologie di record, come ad esempio gli Analog Input/Output Record, vi sono dei limiti di allarme configurabili dall'utente: due soglie superiori e due soglie inferiori. Si ha inoltre che ognuno dei limiti è caratterizzato da una propria severità d'allarme.
- State Alarm:
  concettualmente simile alla precedente tipologia di allarme, lo State Alarm fornisce all'utente la possibilità di associare, per i valori discreti desiderati ed elaborati da opportuni record, una condizione di allarme.

Si può osservare come solo alcune delle tipologie di allarme sono configurabili dall'utente (Limit Alarm e State Alarm), mentre le altre vengono gestite in automatico dal software di controllo. [6]

Come si evince, la caratterizzazione di un record avviene tramite la configurazione dei suoi campi. Si è potuto osservare che ogni aspetto di un record, dalla scelta della modalità di scansione all'indicazione delle locazioni di memoria dove reperire e/o scrivere valori, viene definita attraverso opportuni Record Field.
In EPICS, i campi di un record vengono classificati in due categorie:

- Common Fields: sono i campi che definiscono le proprietà comuni a tutte le tipologie di record. In questo insieme è possibile determinare due sottocategorie:
  – Design Fields: sono i campi che individuano la descrizione del record, il meccanismo di scansione, la successione dei record da processare, l'inizializzazione del record, ecc..
  – Run-Time Fields: sono i campi impiegati per la gestione degli allarmi e della loro severità, la gestione di errori, ecc..
- Specific Fields: sono i campi che dipendono dal Record Type e che definiscono le particolari proprietà del record.

Si ha quindi che, per avere una Process Variable con le caratteristiche desiderate, bisogna selezionale l'opportuno Record Type e configurarne i Common Field e Specific Field. Il pacchetto EPICS base fornisce un insieme di Record Type che permettono di elaborare i principali tipi di dato e realizzare algoritmi di controllo più o meno complessi.
I tipi di record più comuni sono:

- Analog Input (ai) ed Analog Output (ao), che effettuano input/output (anche da/verso hardware) di un valore analogico
- Binary Input (bi) e Binary Output (bo), per input/output di un valore binario (bit)
- Long Input (longin) e Long Output (longout), per input/output di valori interi fino a 32bit
- Multi-Bit Binary Input/Output (mbbi/mbbo), per input/output di segnali

binari a più bit.

- String Input/Output (stringin/stringout), per input/output di stringhe di 40 caratteri
- Calculation (calc), per effettuare operazioni matematiche di vario tipo
- Calcolation Output (calcout), effettua le stesse operazioni matematiche, ma dispone anche di link in uscita.
- Compression (compress), che utilizza array circolari per effettuare semplici operazioni statistiche su una serie di valori
- Fanout (fanout), per fornire un segnale di Forward Processing verso diversi record con processamento passivo
- Data Fanout (dfanout), per inoltrare un valore in output verso diversi altri record
- Waveform, per gestire array di qualunque tipo e dimensione
- Subarray, per ottenere sub-array di lunghezza arbitraria, a partire da record di tipo waveform
- Motor, per gestire parametri e funzionamento di motori
- Subroutine, che permette di eseguire delle routine (scritte in C, C++, …) all'inizializzazione o ad ogni processamento del record.

Per una descrizione più dettagliata dei diversi Record Type disponibili si rimanda al Record Reference Manual reperibile nel sito del software EPICS[10].
Va inoltre indicato che tale insieme è ampliabile integrando, nella particolare applicazione IOC desiderata, appositi Record Support sviluppati dalla comunità EPICS.
Come è stato precedentemente detto, un insieme di Record, memorizzati in un IOC, individua un database di informazioni e variabili di controllo. La realizzazione di un Database avviene attraverso la definizione di due file: il Database Definition file ed il Record Instance file. In particolare:

- nel Database Definition file viene definito ogni aspetto del database (menu e opzioni dei campi dei record, tipologie di record disponibili, device, driver, ecc..). Ad esempio, è in questo file che si ha la possibilità di inserire un periodo di scansione diverso da quelli standard, secondo quanto è stato detto descrivendo le specifiche di scansione dei record. Nel file, avente estensione .dbd, non vengono però indicati i record, definiti dall'utente, necessari per realizzare le variabili di processo del sistema di controllo;
- nel Record Instance file vengono indicati solamente i record, opportunamente configurati, che vengono inizializzati per realizzare gli algoritmi di controllo implementati dall'utente. Il file creato è un file di testo avente estensione .db.

Si ha quindi che un Database Definition file definisce il set di regole necessario per poter interpretare ed elaborare correttamente tutti i record, definiti tramite i Record Instance file, che risiedono nella memoria di un IOC. Si deduce che, attraverso la modifica del Database Definition file, è possibile modellare le caratteristiche e le opzioni del software secondo le necessità richieste dall'applicazione.

Inoltre, ogni combinazione di definizioni possono apparire in un singolo file oppure in un set di file relazionati tra loro. Per la modifica dei parametri costituenti il Database Definition file si fa riferimento al manuale EPICS: Input/Output Controller Application Developer's Guide [11].

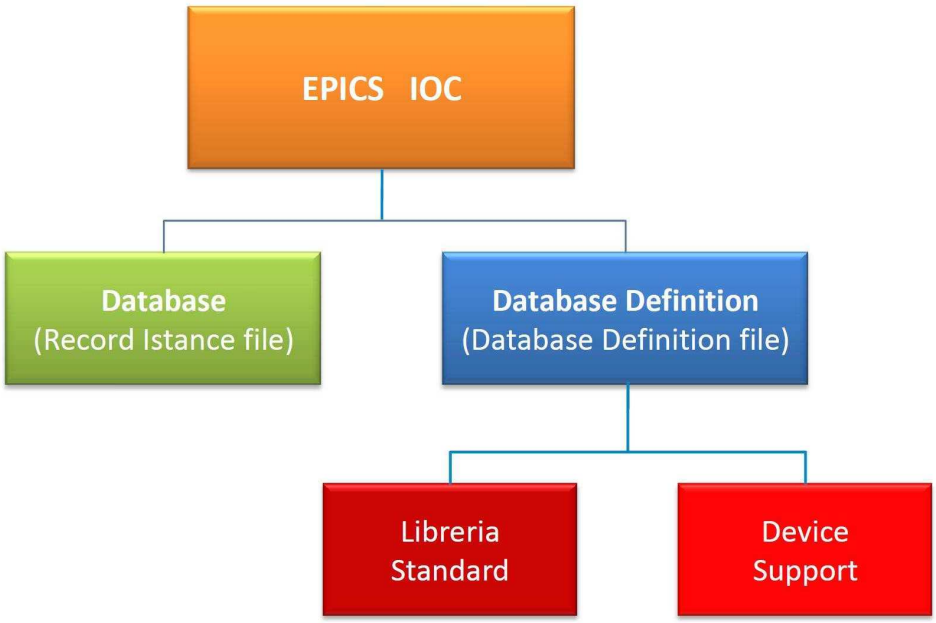

Figura 3.4 Schema dei componenti su cui è costituita l'applicazione

In Figura 3.4 è possibile osservare gli elementi necessari per implementare i parametri di controllo all'interno di un'applicazione IOC. All'interno di questa struttura si distinguono:

- il Database utilizzato per definire le Process Variable del sistema di controllo;
- il Database Definition che fornisce l'insieme delle regole per poter elaborare le Process Variable definite nel Database. Esso è a sua volta costituito:
  - dalle Librerie Standard fornite dall'ambiente EPICS base;
  - dagli EPICS module sviluppati da terze parti, come ad esempio il Device Support necessario per implementare l'interfaccia con un particolare hardware collegato all'IOC.

L'interfacciamento con l'hardware viene effettuato da un IOC, il quale accede periodicamente (secondo criteri configurabili) ai dati dello strumento e li inserisce nel database sotto forma di PV. Quando un client richiede attraverso il channel access il valore della variabile dello strumento, in realtà accede all'ultimo valore acquisito, cioè quello presente nel record.
Quando un record viene processato, l'IOC engine (ovvero il processo che gestisce lo scanning dei records) va a verificare il contenuto di un particolare campo del record (DTYP – device type) per verificare se tale record sia di tipo soft channel oppure se sia collegato ad un dispositivo hardware. Nel caso il campo corrisponda ad un dispositivo hardware, il controllo viene passato alla funzione che gestisce la comunicazione con lo strumento.

Il set di funzioni che implementano l'interfaccia tra il record ed il dispositivo hardware si chiama Device Support. Nella pratica, il Device Support contiene i dettagli relativi alla comunicazione con la strumentazione: ad esempio, uno strumento che ha un'interfaccia di tipo seriale avrà un protocollo di comunicazione basato su stringhe ASCII. Questo protocollo è specifico per un particolare strumento ed è generalmente indipendente dal driver della porta di comunicazione. Inoltre, esso definisce anche il comportamento sincrono o asincrono del record, a va a gestire gli eventuali meccanismi di interrupt.

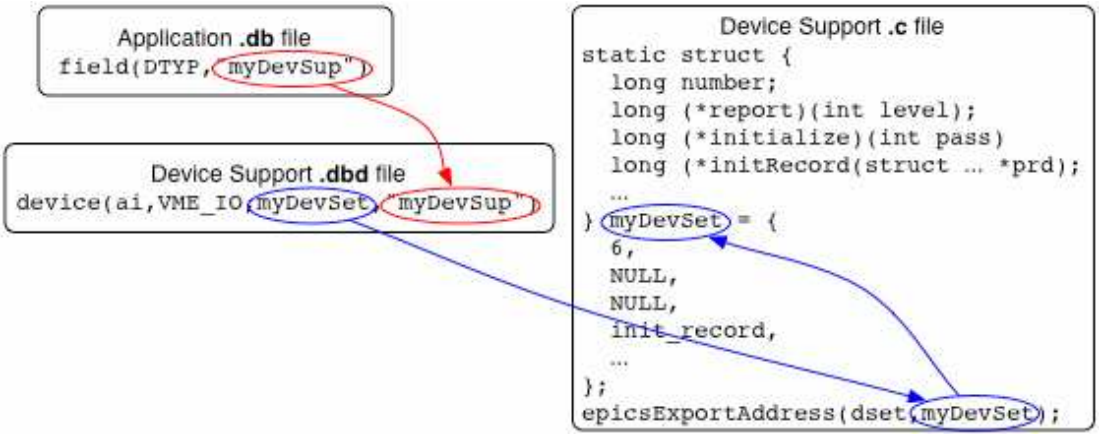

Figura 3.5 – Meccanismo che permette al record di trovare il proprio device support

Il Device Support invocherà a sua volta le funzioni di I/O di basso livello che sono richieste per la gestione del canale fisico a cui è collegato lo strumento (Driver Support). Queste funzioni sono quasi sempre dipendenti dal sistema operativo sopra il quale funziona l'IOC.
Quando si vuole sviluppare il supporto per un nuovo strumento, è buona pratica tenere separato il livello del Device Support da quello del Driver Support al fine di ottenere una migliore riusabilità del codice, specialmente nel caso di sostituzioni nel tipo di hardware. Per esempio, uno strumento che comunica con un protocollo a stringhe ASCII potrebbe essere fornito con un'interfaccia RS232 o GPIB, ma il set di comandi rimane invariato in entrambi i casi.
Tuttavia, nel caso in cui le funzioni per gestire l'hardware siano abbastanza semplici, è anche possibile scrivere le funzioni del Device Support e del Driver Support all'interno di un unico file.
Riassumendo, l'architettura di un IOC EPICS può essere suddivisa in vari livelli, ciascuno dei quali comunica solamente con il livello precedente e con il successivo. [6, 8]

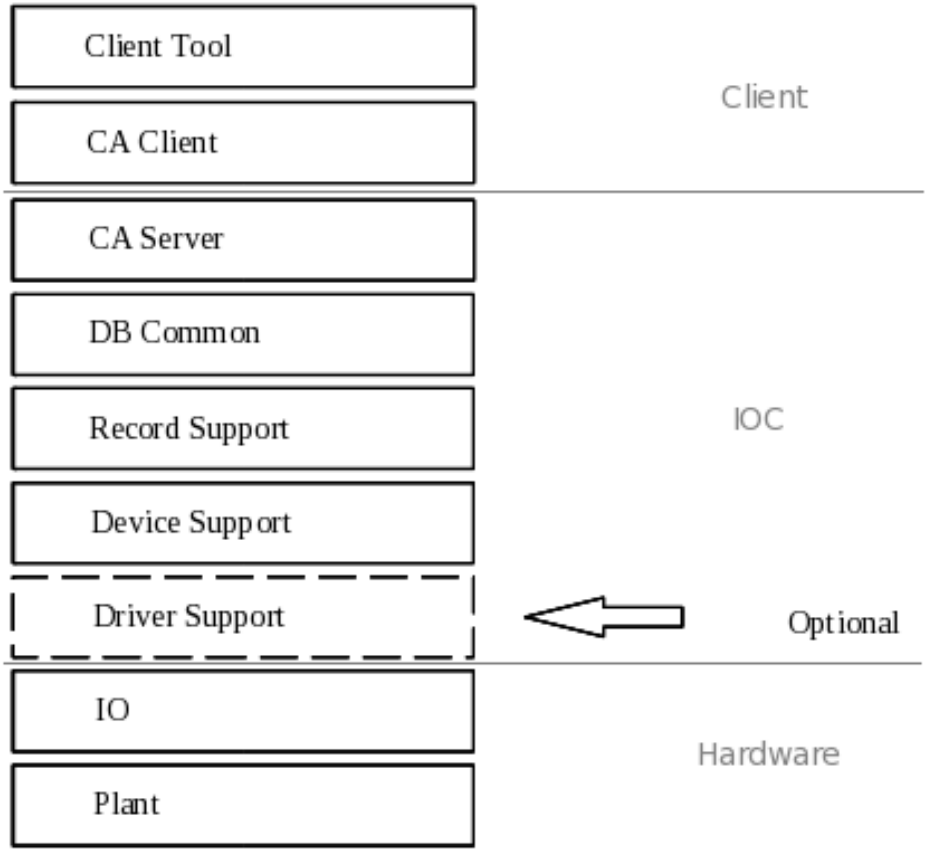

Figura 3.6 Architettura del software EPICS

3.3 Strumenti di sviluppo e gestione

EPICS dispone di un ampio set di strumenti, sviluppati dalla comunità, che permettono di soddisfare pressoché qualsiasi esigenza inerente ai sistemi di controllo distribuiti.
Per la configurazione di un proprio sistema di controllo, la parte più rilevante è senza dubbio quella della configurazione del database, che definisce quindi i dati e le interazioni fra di essi. Esso è composto, come già accennato, dai record contenenti le variabili di processo e dai link fra i record. Tutti i campi che vanno a formare un record e le sue funzioni possono essere definiti con il solo utilizzo di stringhe ASCII; per la configurazione di un database è quindi sufficiente utilizzare un semplice editor di testo.
Esistono tuttavia degli strumenti grafici che permettono di configurare un database visualizzando tutti i record in esso presenti, i link fra di essi e, per ciascuno, l'elenco dei campi modificabili.
Fra i vari strumenti atti a questo scopo è noto il VDCT (Visual Database Configuration Tool), attualmente in uso anche presso i Laboratori Nazionali di Legnaro.
Come si può vedere nella seguente immagine, VDCT consente di visualizzare graficamente i record, i relativi campi, ed i vari link fra i record; questi ultimi si possono distinguere fra link per il passaggio dei dati (ad esempio quelli fra i campi OUT e VAL), e quelli per invocare il

processamento dei record (quelli uscenti dal campo FLNK e dal record di tipo fanout).

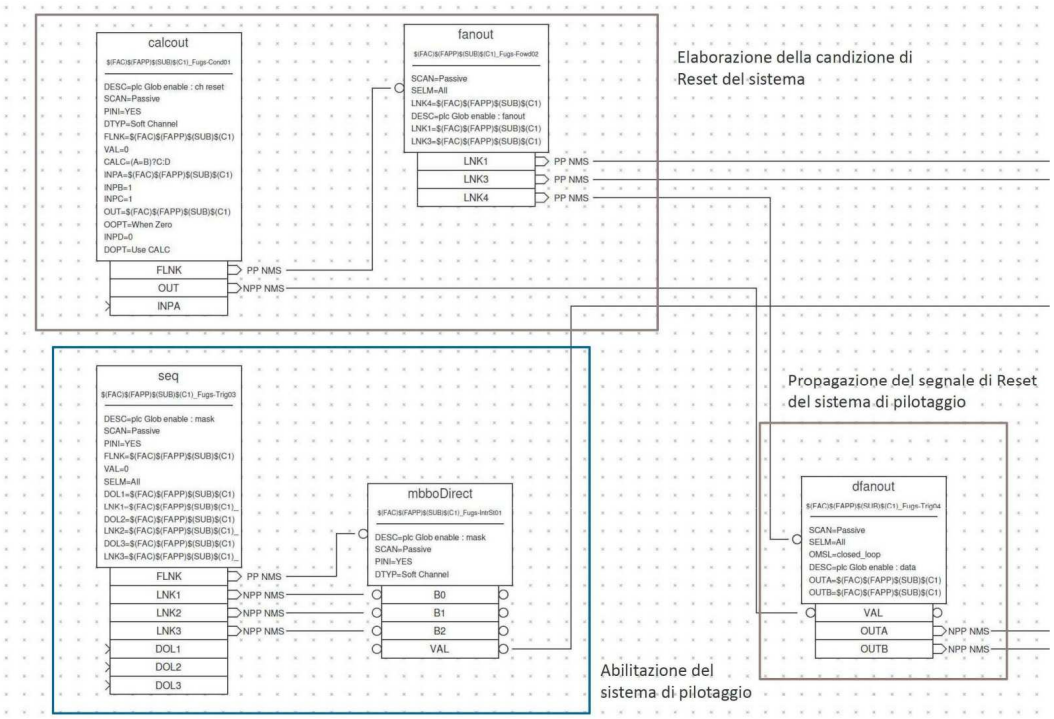

Figura 3.7 Visualizzazione dei record di un database con VDCT

Per il presente lavoro di tesi tuttavia, vista la possibilità di suddividere il database per il sistema di diagnostica del fascio in moduli di dimensioni abbastanza ridotte, per la sua configurazione si è utilizzato solo un editor di testo.

Sul lato client, esiste un notevole numero di applicazioni intese a velocizzare la realizzazione di un sistema di controllo completo nelle sue funzionalità; un ruolo particolarmente importante è quello della realizzazione di interfacce per l'operatore (OPI).
Un tool storico per la creazione di interfacce per utente è MEDM (Motif Editor and Display Manager), un applicativo client standalone che consente di collegare in modo immediato una PV ad un oggetto grafico.

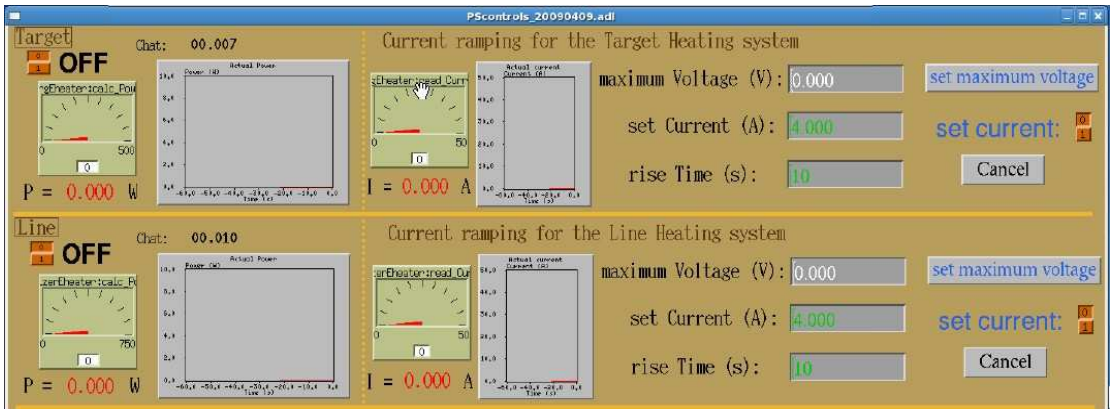

Figura 3.8 Esempio di interfaccia MEDM (usata per il progetto SPES)

Lo strumento principalmente in uso presso i Laboratori Nazionali di Legnaro per la creazione e l'utilizzo di interfacce per l'operatore è il Control System Studio (CSS).
CSS è sviluppato in Java, e consente in modo estremamente rapido di creare ed utilizzare interfacce; esso è dotato di una vasta gamma di oggetti grafici configurabili adattabili a moltissime esigenze. Nonostante sia ancora in fase di sviluppo, esso rappresenta già una soluzione efficiente e completa per la gestione di impianti anche complessi.

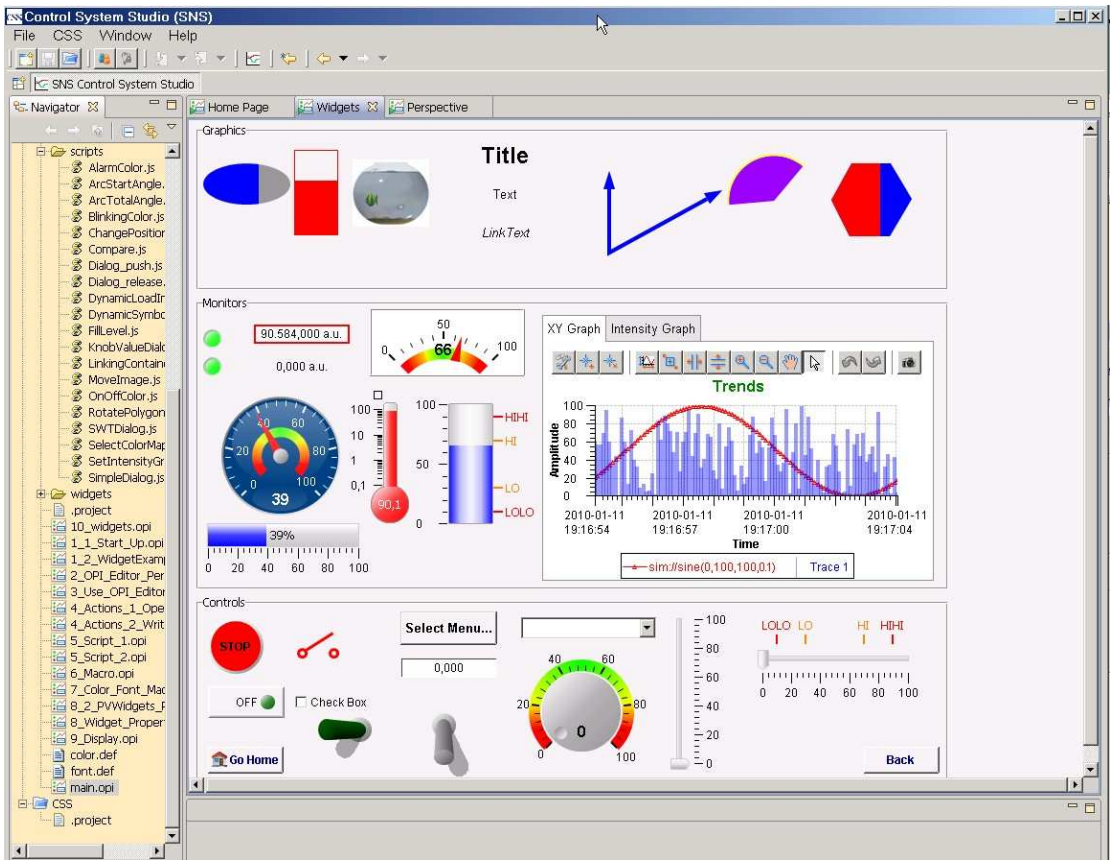

Figura 3.9 Esempio di interfaccia CSS

Ulteriori strumenti utili alla gestione di un sistema di controllo sono quelli dedicati all'archiviazione dei dati di certe variabili di processo per una

successiva analisi, come ad esempio il Channel Archiver. Con questo strumento è possibile risalire ad uno storico completo dei dati monitorati, che possono venire salvati in locale o anche su di un database (ad esempio MySQL) remoto.
Un altro strumento fondamentale è quello per la gestione degli allarmi, l'Alarm Handler. Esso infatti permette di visualizzare gli allarmi secondo la loro sequenza temporale o a seconda della gravità, consentendo di analizzare rapidamente la correlazione tra i vari eventi. [7, 12]

Capitolo 4 – Le Misure

La strumentazione per la diagnostica è una componente fondamentale di un acceleratore, in quanto consente di capire la struttura longitudinale e trasversale del fascio accelerato e quindi di ottimizzare i parametri di trasporto. Essa interagisce direttamente con il fascio e può catturarne tutte le particelle accelerate (diagnostica distruttiva) oppure solamente una piccola frazione.
Esiste una vasta varietà di parametri che possono venire misurati, e per un buon allineamento del fascio tutte le proprietà rilevanti dovrebbero essere controllabili.
Il tipo di strumentazione differisce a seconda dell'acceleratore; ad esempio, in un acceleratore lineare a bassa intensità di corrente si usano normalmente profilatori a griglia che vengono attraversati dal fascio assorbendo solamente una piccola quantità di energia. In acceleratori ad alta corrente questo non sarebbe possibile poiché la potenza assorbita è sufficiente a distruggere il dispositivo: in tal caso si usano dei rilevatori di posizione basati su pick-up capacitivi che non intercettano il fascio ma riescono comunque a fornire informazioni sulla struttura spaziale dello stesso.

Nei paragrafi seguenti vengono descritti lo scopo ed i metodi di misura della corrente di fascio, del profilo del fascio, ed anche della sua emittanza, con i metodi attualmente implementati nel sistema di diagnostica del Front End del progetto SPES.

4.1 La corrente del fascio

Come descritto nei capitoli precedenti, un fascio di particelle è composto da atomi che vengono ionizzati ed accelerati in una certa direzione all'interno di una conduttura in alto vuoto. Questo flusso ordinato di atomi aventi una carica elettrica negativa o positiva (essendo dotati di un elettrone in più oppure, come nel nostro caso, avendone una carenza) va a formare una corrente elettrica.

La misura dell'intensità di corrente (che dipende dal numero di ioni accelerati e dal loro stato di carica) è importante per capire se l'impostazione (setup) della macchina e' stata fatta in modo corretto; inoltre il valore della corrente è un parametro fondamentale ai fini delle valutazioni statistiche sui dati del processo fisico che si vuole studiare.
Nel caso del Front-End del progetto SPES, gli ioni accelerati sono di tipo 1+, cioè hanno carica positiva (un elettrone in meno). La misura dell'intensità di corrente è quindi direttamente proporzionale al numero di atomi al secondo che vengono ionizzati ed accelerati.
Poiché la carica elementare di un elettrone è di $1.602 \cdot 10^{-19}$ Coulomb e un Ampere corrisponde ad un Coulomb/sec, una corrente di 1nA corrisponde ad un flusso di $6.25 \cdot 10^{9}$ elettroni al secondo.
Nel nostro caso la misurazione della corrente del fascio viene effettuata con uno strumento chiamato Faraday Cup, il cui funzionamento viene descritto in seguito in questo capitolo.

4.1.1 Misurazione con Coppa di Faraday

La Faraday Cup (o coppa di Faraday) non è altro che una coppa di metallo avente lo scopo di catturare gli ioni (o gli elettroni) liberi nel vuoto. Nel nostro caso, quando gli ioni vengono in contatto con il metallo della coppa, prelevano l'elettrone di cui necessitano per bilanciare la propria carica. Nel caso di un fascio ionico continuo e stabile, il flusso di elettroni prelevati va a formare una corrente che può essere misurata.

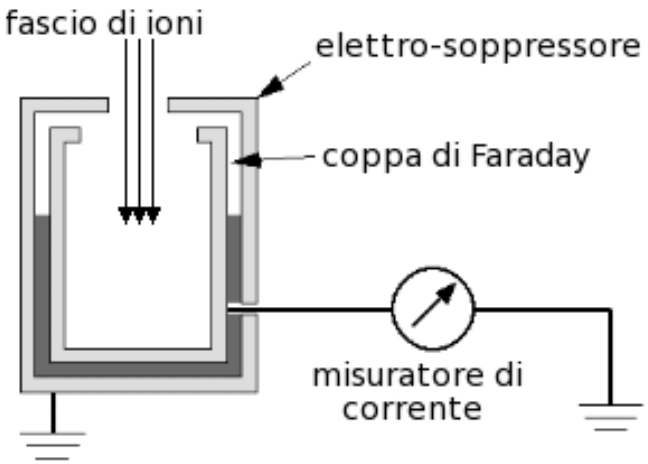

Figura 4.1 Schema di una Faraday Cup

La corrente rilevata sulla coppa di Faraday può variare di diversi ordini di grandezza che vanno dal microampere (10^{-6}) fino al limite inferiore del picoampere (10^{-12}); si tratta quindi segnali molto deboli che devono essere elaborati prima della conversione A/D.
La corrente raccolta dalla Faraday cup viene dunque pre-amplificata e convertita in tensione in un range compatibile con la dinamica di ingresso dell'ADC (tipicamente 0 .. 10V).

Tale operazione viene effettuata con un circuito progettato all'interno dei Laboratori Nazionali di Legnaro basato su amplificatori operazionali a bassissima corrente di bias (es. AD542). Un multiplexer analogico DG528CJ viene utilizzato per cambiare il range di conversione, attraverso la commutazione dei resistori di feedback. Il canale del multiplexer viene selezionato attraverso due bit pilotati da una scheda di output digitale (Xycom XVME-240) consentendo di scegliere tra quattro fattori di conversione I → V nel range da 1nA=10V a 1nA=1mV.
La tensione ottenuta viene misurata da un ADC su scheda VME (Xycom XVME-566), che ne legge il valore in una scala 0-10 Volt e una risoluzione di 12 bit.

Esiste un altro problema da considerare: gli effetti di emissione secondaria di elettroni. Essi sono fenomeni che avvengono quando particelle aventi una certa energia colpiscono una superficie o attraversano un materiale, e causano l'emissione di particelle secondarie. Nel nostro caso, è possibile che, quando gli ioni vanno a colpire la Faraday Cup, dal metallo della coppa vengano liberati degli elettroni. Ai fini della misura, ci si ritroverebbe quindi a leggere non solo la corrente dovuta agli elettroni “prelevati” dagli ioni del fascio, ma anche quella dovuta all’emissione secondaria di elettroni. Per evitare questo effetto indesiderato, si è utilizzata una coppa di Faraday dotata di elettro-soppressore. Ciò non è altro che un rivestimento metallico esterno alla faraday cup (tuttavia completamente isolato da essa), a cui viene applicato un potenziale di -300 Volt. In questo modo la maggior parte degli elettroni eventualmente liberati rimangono all'interno della coppa o, se fuoriuscenti, vengono respinti al suo interno dal potenziale del soppressore. [13]

Figura 4.2 Faraday Cup con soppressore inserita nella traiettoria di fascio.

Una ulteriore necessità è quella di poter inserire o estrarre la coppa dalla linea di fascio. Per effettuare questa movimentazione si è optato per un pistone meccanico ad aria compressa comandato da una valvola a 24 Volt AC.
Per impartire da software la posizione della coppa, si è quindi utilizzato una scheda per output digitale sempre in formato VME, la Xycom XVME-220. Utilizzando pertanto un solo bit in uscita, e' possibile pilotare un relè elettromeccanico che alimenta la valvola, stabilendo quindi la sua posizione.

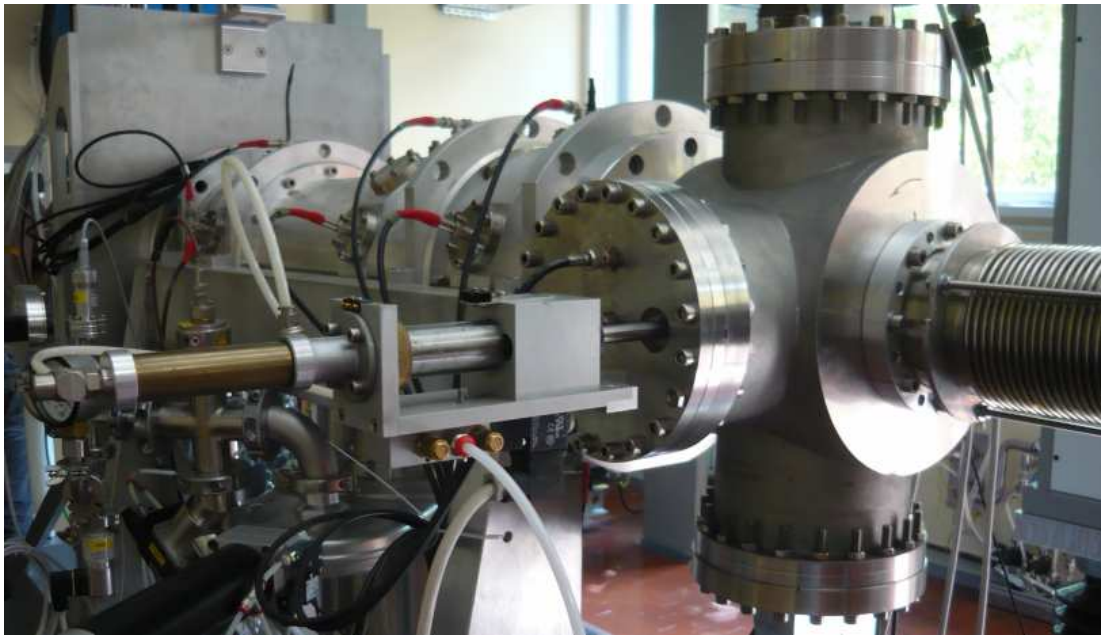
Figura 4.3 Sistema di movimentazione della Faraday Cup

4.2 Il profilo del fascio

Un altra caratteristica importante di un fascio è la sua intensità spaziale, detta anche profilo.
Gli ioni di un fascio infatti non viaggiano tutti secondo traiettorie esattamente parallele, ma le traiettorie sono generalmente divergenti e necessitano di periodica rifocalizzazione. Lo strumento di rilevamento del profilo serve proprio a questo: con un funzionamento simile a quello della coppa di Faraday, riesce a restituire dei valori di corrente proporzionali al numero di ioni che attraversano una certa area del piano perpendicolare alla traiettoria del fascio.
Tale strumento, chiamato beam profiler (profilatore di fascio), è composto, nel caso della strumentazione in uso ai Laboratori Nazionali di Legnaro, da una coppia di griglie metalliche ed è in grado di restituire dei valori di corrente proporzionali alle sezioni orizzontali o verticali del piano. E' quindi possibile ottenere due serie di correnti, una per la scala orizzontale ed una per quella verticale.
Tali valori sono particolarmente utili in quanto permettono di ricostruire e visualizzare la forma del fascio, la sua larghezza, ed anche la sua intensità

secondo le varie coordinate spaziali. In fase di settaggio dei parametri di trasporto, la visualizzazione dei dati del profilo permette quindi di capire se il fascio sia effettivamente centrato o se debbano essere ritoccati i campi dei magneti deflettori o delle lenti elettrostatiche.

4.2.1 Misurazione con Profilatore di Fascio

La griglia di misurazione del profilo è composta da 80 fili, 40 orizzontali e 40 verticali, aventi uno spessore di 50 μm ed un passo di 750 μm, privi di alcun contatto fra di loro.
In modo analogo alla faraday cup, quando gli ioni vengono in contatto con il metallo di uno dei fili della griglia (o anche solo se vi passano estremamente vicino) si riprendono l'elettrone di cui necessitano per bilanciare la propria carica elettrica. Misurando quindi le deboli correnti su questi fili si può avere una quantità indicativa della carica che sta transitando lungo tutta la lunghezza del filo.
Mentre nella Faraday cup ciò che si acquisisce è il valore di tutta la corrente del fascio, i valori letti dalla griglia del profilatore vanno analizzati con più attenzione. Essi infatti forniscono dati "relativi" sulla quantità di ioni che transitano in una certa zona, indicando ad esempio che la corrente transitante su di un filo centrale è di un certo numero di volte più grande di quella laterale, ma senza fornire dati assoluti.
Tuttavia questa misura è importante perchè, oltre a stabilire se un fascio è centrato e messo a fuoco, fornisce anche informazioni importanti sulla distribuzione della carica nello spazio, ossia permette di visualizzare il cosiddetto "profilo" del fascio.

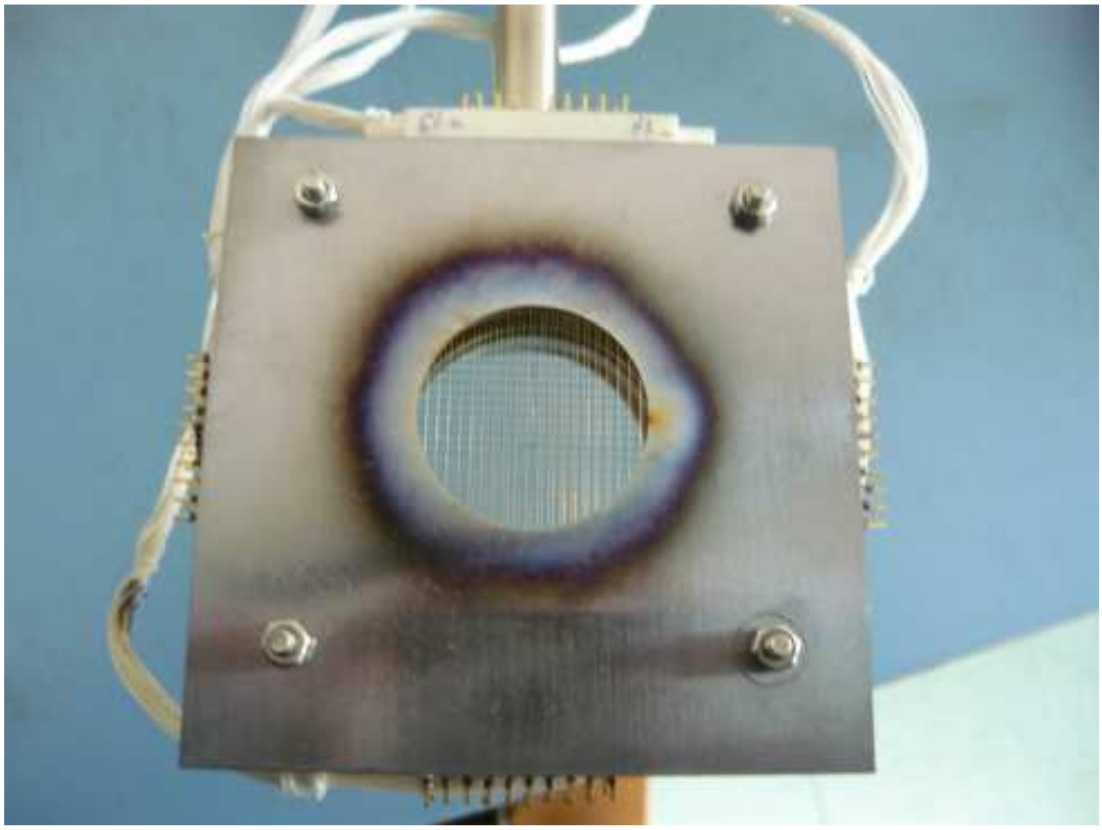
Figura 4.4 La griglia del profilatore di fascio.

Per effettuare la lettura dei dati, la corrente di ogni filo della griglia viene convertita in tensione e successivamente misurata con una tecnica analoga a quella della faraday cup. Ogni filo della griglia è collegato ad un amplificatore operazionale in configurazione invertente, anch'esso con due possibili rapporti di guadagno. Tutti i 40 amplificatori si trovano su di un unico circuito stampato, che viene montato all'interno di una scatola metallica in prossimità della linea di fascio; tale circuito include inoltre un multiplexer seriale analogico in grado di restituire sul canale di uscita il valore di tutti i 40 segnali in ingresso (convertiti ed amplificati) seguendone una determinata sequenza. Il passaggio da un canale al successivo viene impartito alla scheda con un impulso sul canale Clock_In.
Per avere in uscita tutti i 40 valori della griglia occorre quindi che ci sia un treno di almeno 39 impulsi; il contatore interno del multiplexer è progettato per azzerarsi al ricevimento del 42° impulso, oppur e dopo un certo tempo di inattività regolato da un circuito monostabile.
La scheda ADC contiene un counter-timer configurato per generare un treno di impulsi che viene utilizzato per incrementare il valore di controllo del multiplexer seriale. Pertanto, con questo meccanismo, per leggere una intera griglia di 40 valori è sufficiente utilizzare un solo canale. I 40 valori letti in sequenza vengono quindi memorizzati e gestiti come un unico array.
Analogamente alla coppa di faraday, anche il profilatore necessita di essere spostato dentro e fuori dalla traiettoria del fascio. Diversamente dal caso precedente però, per la sua movimentazione si è optato per un motore passo-passo. Questo, mettendo in rotazione una vite elicoidale, inserisce e disinserisce la griglia nella linea di fascio.

Al controller del motore viene impartito un movimento relativo pari ad un numero relativamente grande di passi (step), che può essere eseguito in un senso o nell'altro, e ciò mette in movimento il motore fino al raggiungimento di uno dei due finecorsa.
Anche per il controllo dei motori passo-passo si è optato per una scheda in formato VME, da inserire nello stesso box (crate) assieme all'ADC e alle schede I/O digitali.

4.3 L'emittanza

L'emittanza è una proprietà molto importante in quanto descrive la qualità di un fascio di particelle.
Il moto di un fascio può essere descritto da una equazione differenziale lineare del secondo ordine. Ciò ammettendo l'assenza di componenti non-lineari, quali forze dovute a cariche elettriche o accoppiamenti tra i due piani trasversali. La qualità del fascio è data dal volume dello spazio di fase, che è una costante del moto. Per un piano, l'emittanza è definita da:

$$\epsilon_x = \frac{1}{\pi} \int_A dx dx'$$

dove A=πϵ è l'area dello spazio di fase occupata dal fascio. La determinazione dell'emittanza è equivalente alla determinazione della distribuzione di coordinate spaziali *x* (ad esempio il profilo), la distribuzione angolare *x'* e la correlazione fra *x* ed *x'*.
L'interpretazione dell'area assume una distribuzione omogenea limitata dall'ellisse. Un caso più realistico è la distribuzione di densità gaussiana *ρ(x,x')* definita alla posizione del vettore *(x,x').* Il valore dell'emittanza sarà quindi dato da:

$$\epsilon_x = \frac{1}{\pi} \int \rho(x, x') dx dx'$$

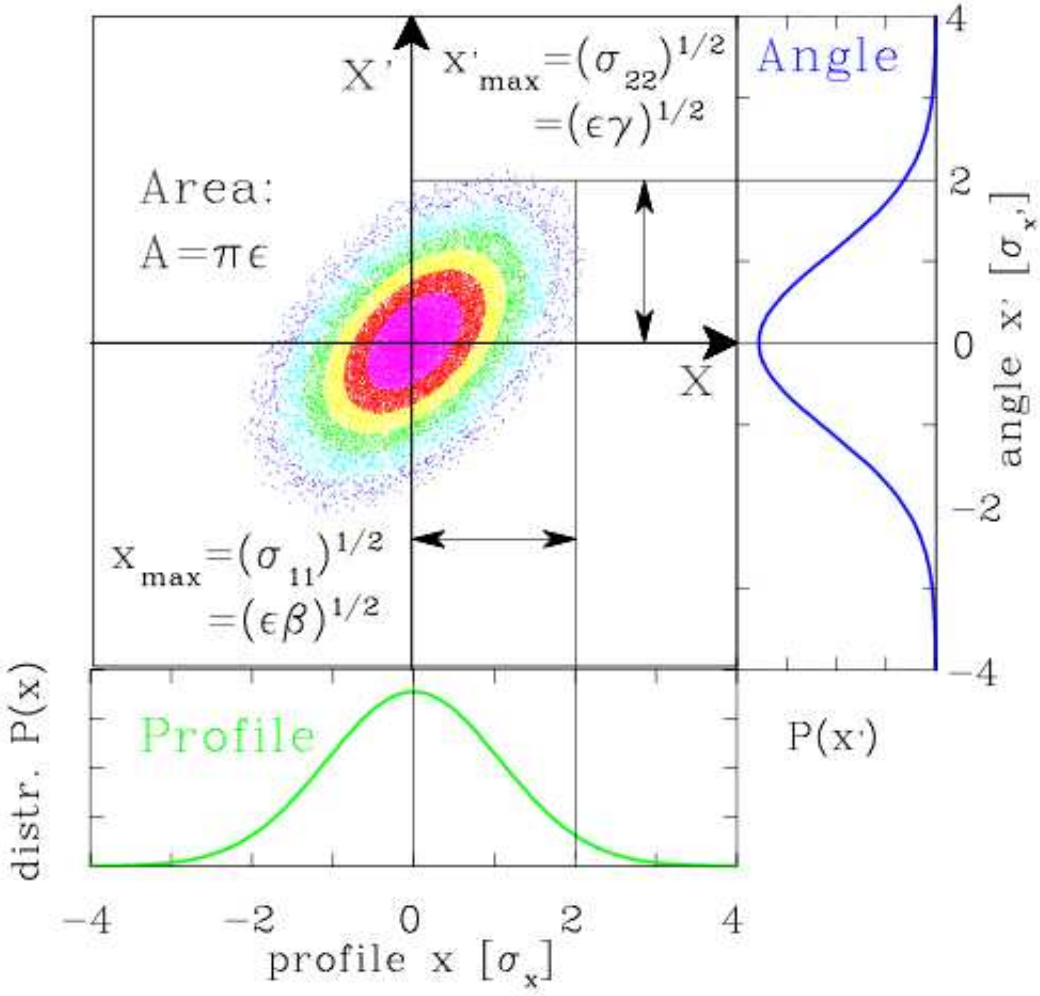

Figura 4.5 – L'ellisse di emittanza e le proiezioni delle coordinate spaziali ed angolari per una densità di distribuzione gaussiana.

La distribuzione gaussiana può essere espressa dal vettore *(x, x')* e dalla matrice simmetrica di fascio *σ*

$$\sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix}$$

ad una posizione *s* nella linea di trasporto del fascio. La matrice di fascio *σ* è una rappresentazione dell'ellisse del fascio a questa posizione s lungo il suo percorso. La densità bidimensionale è quindi:

$$\rho(x,x') = \frac{1}{2\pi\epsilon_x}\exp\left[\frac{-1}{2}\vec{x}^T\sigma^{-1}\vec{x}\right] = \frac{1}{2\pi\epsilon_x}\exp\left[\frac{-1}{2\det\sigma}\left(\sigma_{22}x^2 - 2\sigma_{12}xx' + \sigma_{11}x'^2\right)\right]$$

Il parametro $\sqrt{\sigma_{11}}$ è la deviazione standard per il profilo di distribuzione ottenuto integrando la densità *ρ* su *x'*. $\sqrt{\sigma_{22}}$ è il corrispondente valore per la distribuzione angolare ottenuta integrando *ρ(x,x')* su *x*. σ_{12} invece descrive la correlazione fra *x* ed *x'* (ad esempio, l'angolo dell'ellisse).
Usando questa notazione, l'emittanza può quindi essere definita come

$$\epsilon_x = \sqrt{\det \sigma} = \sqrt{\sigma_{11}\sigma_{22} - \sigma_{12}^2}$$

e corrisponde ad una occupazione del 39% dell'intera area dello spazio di fase.
In altre parole, la qualità del fascio gaussiano ad una certa posizione *s* è interamente descritto dalla matrice di fascio σ(s).
Frequentemente sono utilizzati i parametri di Twiss, che sono gli elementi della matrice normalizzati per l'emittanza:

$$\alpha = -\sigma_{12}/\epsilon \qquad \beta = \sigma_{11}/\epsilon \qquad \gamma = \sigma_{22}/\epsilon$$

La matrice di fascio diventa quindi:

$$\sigma = \epsilon \cdot \begin{pmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{pmatrix}$$

e l'equazione dell'ellisse di fascio può essere scritta come

$$\gamma x^2 + 2\alpha x x' + \beta x'^2 = \epsilon$$

con la normalizzazione

$$\beta\gamma - \alpha^2 = 1$$

La larghezza del profilo e della distribuzione angolare sono date da

$$x_{max} = \sqrt{\sigma_{11}} = \sqrt{\epsilon\beta}$$

e

$$x'_{max} = \sqrt{\sigma_{22}} = \sqrt{\epsilon\gamma}$$

Il loro significato geometrico è una o due deviazioni standard a seconda della definizione di emittanza.
Ma il fascio non ha sempre una forma gaussiana. Per descrivere la qualità del fascio il valore *rms* può essere calcolato come

$$\epsilon_{rms} = 4\sqrt{\langle x^2\rangle\langle x'^2\rangle - \langle xx'\rangle}$$

Il valore numerico può essere notevolmente diverso; ad esempio, $\epsilon_{rms}=4\epsilon$ per una distribuzione gaussiana corrisponde ad un contorno contenente l'87% del fascio.
Quando un fascio viene accelerato, la divergenza diminuisce (attenuazione adiabatica) a causa della variazione nel rapporto fra la velocità longitudinale v_s e la velocità trasversa v_x: $x'=v_x/v_s$.
Per confrontare l'emittanza per momenti longitudinali diversi $p_s=m_0 \cdot \gamma_{rel} \cdot v_s/c$, l'emittanza normalizzata ϵ_{norm}

$$\epsilon_{norm} = \frac{v_s}{c}\gamma_{rel} \cdot \epsilon$$

è riferita ad un valore $\gamma_{rel} \cdot v_s/c = 1$, ad esempio ad una velocità di fascio

$$v_s = \sqrt{\tfrac{1}{2}} \cdot c = 0.71 \cdot c$$

(dove c è la velocità della luce e $\gamma_{rel} = 1/\sqrt{1-(v_s/c)^2}$ è il fattore relativistico).
L'emittanza normalizzata è costante sotto condizioni ideali di accelerazione.
Una misura di emittanza consiste nella determinazione del valore numerico di ϵ, come anche dell'orientamento e della forma della distribuzione in spazio di fase.
L'argomentazione qui sopra riportata è basata sulla separazione degli spazi di fase in tutte e tre le direzioni. Questo non è completamente realistico a causa dei magneti flettenti, che introducono dispersione nella direzione orizzontale. [14, 15]

4.3.1 Acquisizione con il Misuratore di Emittanza

Il metodo per determinarla è basato sulla misura del profilo di fascio in determinate condizioni; tuttavia non è importante quale sia il metodo utilizzato per misurarla, a patto che questo abbia una risoluzione adeguata.
Le possibilità di misura anche in questo caso sono diverse, e dipendono dal tipo di acceleratore utilizzato (Linac, sincrotrone, …).
Il tipo di misuratore scelto e realizzato all'interno del progetto SPES è un apparecchio di tipo fessura-griglia (slit-grid), in cui la coordinata spaziale viene fissata da una apertura, mentre viene misurato l'angolo delle particelle.
Questo metodo è abbastanza popolare per i Linac con protoni o ioni pesanti in cui le profondità di penetrazione sono minori di 1 cm.
L'apertura *x* viene fissata da un collimatore, perpendicolare alla traiettoria del fascio, avente una fessura di 0.1-0.5mm. L'angolo *x'* invece viene determinato con una griglia metallica posta ad una distanza che va da 10cm ad un metro, a seconda della velocità degli ioni. Leggendo quindi la corrente che va a depositarsi su ciascuno dei fili, si riesce ad interpretare l'angolo della traiettoria con cui le particelle stanno attraversando la fessura, partendo dal presupposto che in uno spazio privo di campi le traiettorie delle particelle sono linee rette.
Nel nostro caso il collimatore è dotato di una fessura con apertura di 0.1mm ed è posto a 30 cm di distanza dalla griglia.

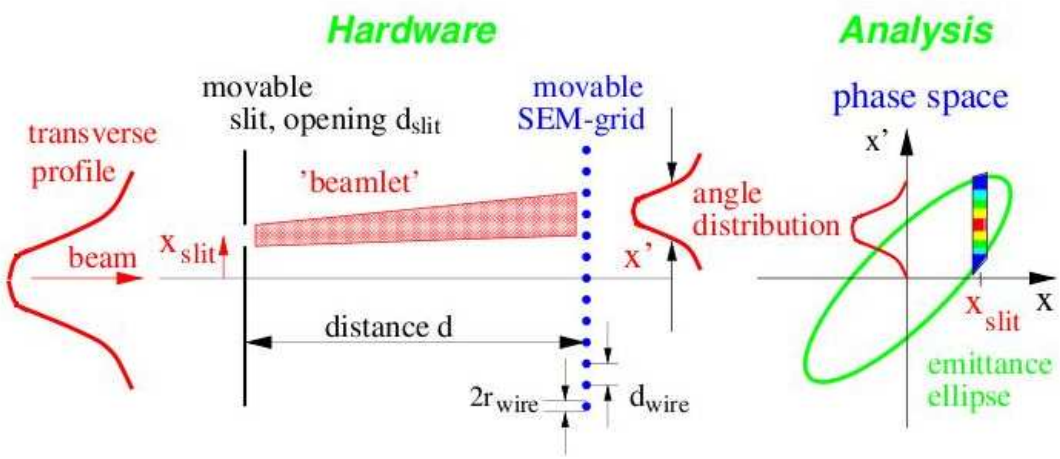

Figura 4.6 – Schema di un apparecchio di misura dell'emittanza di tipo fessura-griglia

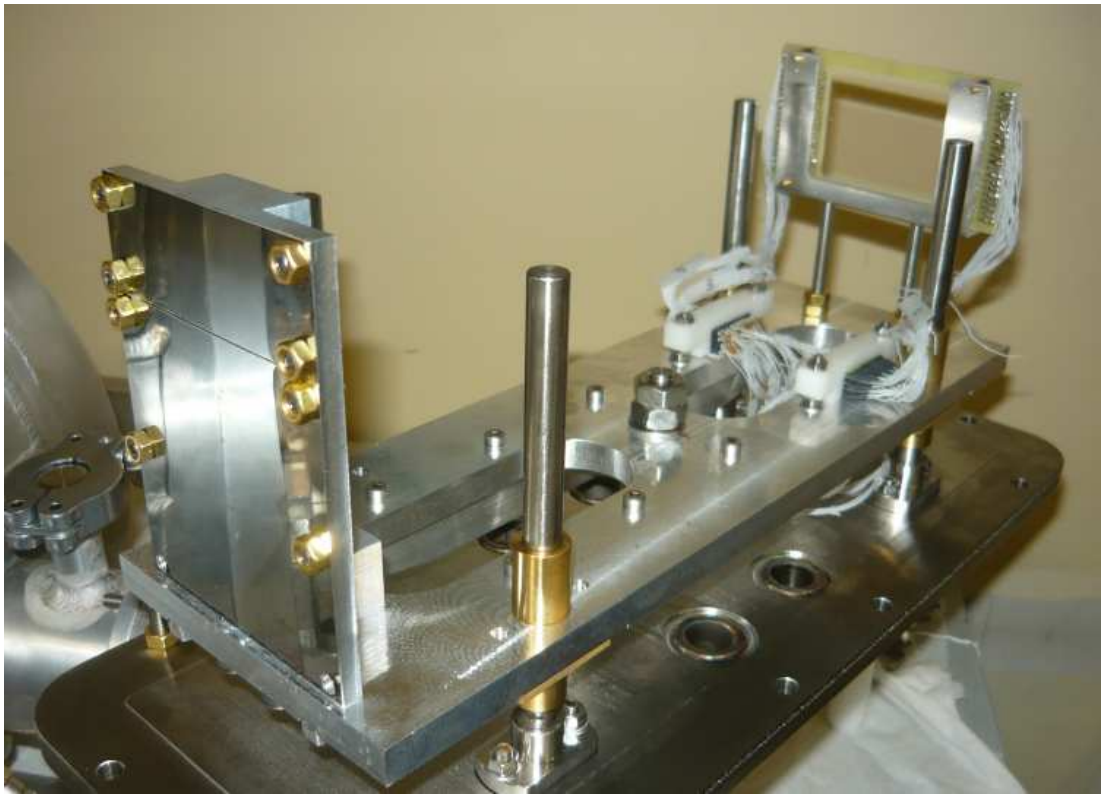

Figura 4.7 Sistema Fessura-Griglia del progetto SPES. (collimatore con fessura a sinistra, griglia di fili sulla scheda a destra)

Il contributo al grafico dell'emittanza è dato dalla distribuzione angolare ad ogni posizione della fessura. Il collimatore viene quindi fatto scorrere lungo il fascio per ottenere tutte le posizioni interessate. Al termine della scansione completa può essere visualizzato il grafico dell'emittanza e dai dati si può calcolare il valore dell'emittanza rms; inoltre si interpola una forma ellittica comprendente una certa percentuale del fascio e da questa interpolazione si ricavano i parametri di Twiss.
Un esempio è dato dal seguente grafico, ottenuto con un fascio di ioni a bassa energia ottenuto presso il CERN di Ginevra.

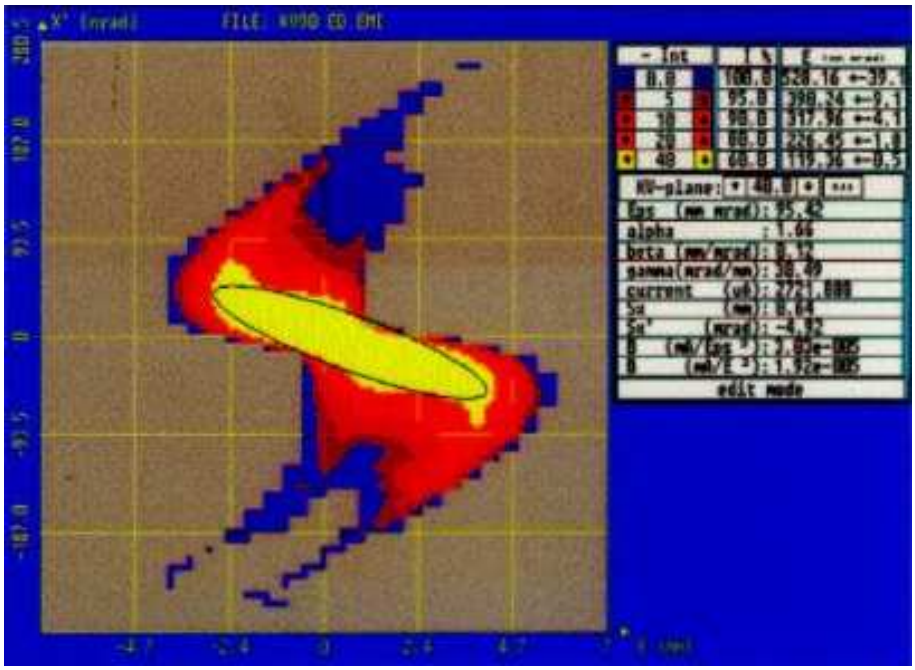

Figura 4.8 Esempio di misura dell'emittanza ottenuta con un fascio di ioni a bassa energia

Questo metodo tuttavia può determinare distribuzioni di spazio e fase ben diverse ed articolate, non solo gaussiane (un chiaro esempio è dato dall'immagine precedente). Vicino alla sorgente di ionizzazione succede di frequente di avere distribuzioni abbastanza particolari, a causa delle ampie forze in spazio di carica o della larghezza del profilo, dove le aberrazioni dei magneti iniziano ad influire sul fascio.
La risoluzione per le coordinate spaziali Δx è limitata dalla larghezza della fessura $\Delta x = d_{slit}$. La risoluzione angolare $\Delta x'$, misurata alla distanza d, è data dal raggio del filo r_{wire} e dalla larghezza della fessura, risultando $\Delta x' = (d_{slit} + 2r_{wire})/d$. La dimensione degli elementi discreti nello spazio di fase è data dal prodotto $\Delta x \cdot \Delta x'$. Questo porta ad un errore di discretizzazione, in particolare nel caso di fasci di basse dimensioni (focalizzati) o con angoli di distribuzione molto piccoli (fasci paralleli). La risoluzione è migliorata scansionando la griglia con passi minori della distanza dei fili d_{wire}, aumentando la densità degli elementi discreti nell'analisi dello spazio di fase. Questo porta ad avere elementi in sovrapposizione, perchè la loro dimensione $\Delta x \cdot \Delta x'$ rimane costante. Lo stesso vale per movimenti del collimatore con passo minore della larghezza della fessura.
La progettazione meccanica e la realizzazione dello strumento sono state curate all'interno dei Laboratori Nazionali di Legnaro, seguendo il modello di un misuratore realizzato in precedenza ed effettuandone alcuni miglioramenti.
Il collimatore e la griglia sono montati su di una unica piattaforma mobile, che scorre su delle guide in direzione ortogonale alla traiettoria del fascio. La movimentazione dell'intera piattaforma viene comandata via software ed attuata con un motore a step posizionato nella parte esterna dell'apparecchio; tale motore serve sia per l'inserimento ed il disinserimento della strumentazione dalla traiettoria del fascio, che anche per la

movimentazione sottile ai fini della misura. In modo analogo alla meccanica per la movimentazione del profilatore di griglia, l'asse del motore va a ruotare una vite elicoidale, che a sua volta agisce sul blocco della piattaforma; a differenza della precedente però, il passo della vite è molto più fino per consentire uno spostamento ortogonale al fascio più preciso.
Per il controllo della posizione si utilizza un encoder analogico lineare che fornisce in uscita una tensione 0-10 Volt.

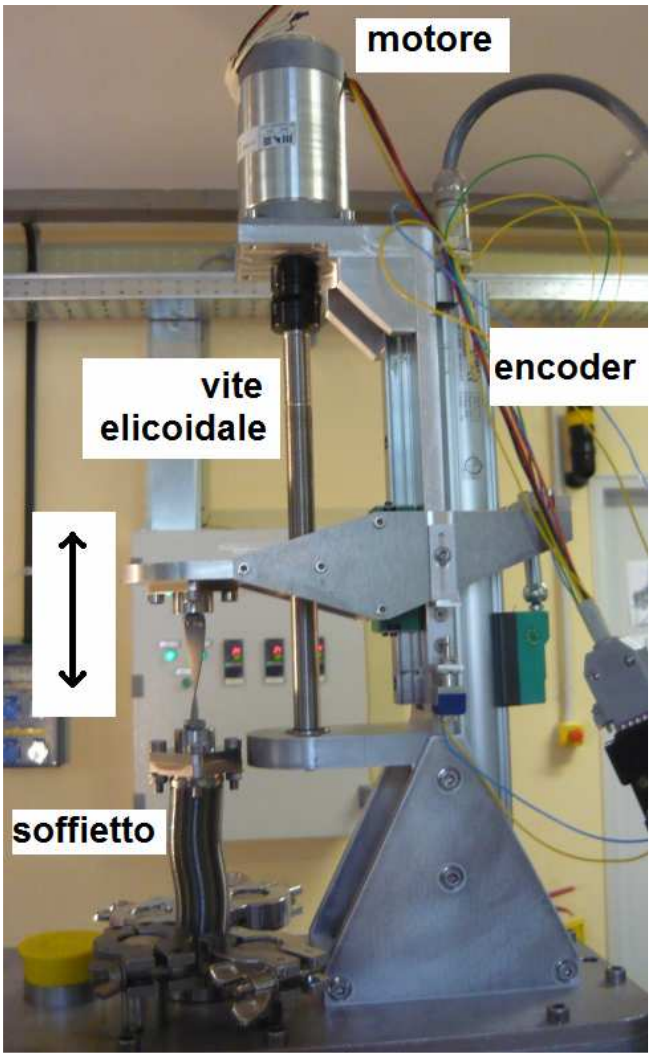

Figura 4.9 Sistema di movimentazione del blocco griglia-collimatore. (encoder lineare a tecnologia magnetostrittiva con cursore verde sulla destra, soffietto contenente il perno di movimentazione in basso)

L'acquisizione dei dati viene effettuata in modo simile a quella del profilatore di griglia, ma in modo più complesso. Essa infatti consiste nell'acquisizione combinata ed istantanea degli 80 valori di corrente sulle griglie e dell'encoder di posizione. La corrente sulle griglie viene convertita in tensione ed acquisita dall'ADC con lo stesso metodo e lo stesso tipo di elettronica del profilatore, multiplexando i valori degli 80 fili su due soli canali. Un terzo canale è invece dedicato alla lettura diretta del valore di tensione generato dall'encoder.

Sia per la movimentazione del motore che per l'acquisizione sono state riutilizzate le stesse schede in formato VME già in uso per la coppa di Faraday ed il profilatore.
Il programma di acquisizione, a seconda della risoluzione richiesta, regola la movimentazione del motore ed il passo con cui acquisire i campioni. Tutti i dati acquisiti, insieme ad altri parametri che verranno descritti in seguito, vengono salvati in un file di tipo testuale, chiamato *rawdata*.
Un secondo programma, successivamente all'acquisizione, va ad accedere al suddetto file rawdata, e procede con l'elaborazione dei dati, fino a giungere al calcolo dell'emittanza, ai parametri di Twiss, ed alla generazione dei dati per il grafico spazio-fase. [14, 16]

4.4 Conclusioni

La strumentazione diagnostica è una componente fondamentale di un acceleratore di particelle, i quanto permette lo studio di determinati componenti del sistema (ad esempio, lo studio dell'efficienza di ionizzazione di quella del sistema di trasporto del fascio).
Il sistema di diagnostica del complesso Front End è composto da una Faraday Cup, necessaria per misurare ed integrare la corrente del fascio, un Beam Profiler, indispensabile per visualizzare la forma del fascio e garantirne una corretta focalizzazione, ed un misuratore di emittanza.
Mentre i dati dei primi due strumenti non necessitano di particolari elaborazioni e possono essere visualizzati direttamente su interfaccia, la misura di emittanza richiede calcoli più complessi, e bisogna quindi separare l'acquisizione dalla post-elaborazione dei dati.

Capitolo 5 – Hardware e Software utilizzato

Per la realizzazione del sistema di acquisizione e controllo della strumentazione diagnostica si è scelto di utilizzare degli standard ben consolidati, cioè un controller su bus VME e con sistema operativo VxWorks, in modo da riuscire a garantire delle tempistiche di acquisizione ed elaborazione ben determinate.

Nei paragrafi seguenti viene quindi descritto il funzionamento del bus VME, le schede hardware utilizzate per il sistema di diagnostica, e le caratteristiche principali del sistema operativo VxWorks.

5.1 Il VMEbus

Il VMEbus è un bus standard per computer orientati ad applicazioni di controllo. Il termine VME è acronimo di Versa Module Eurocard ed è stato coniato per la prima volta nel 1981 da un consorzio delle compagnie Motorola, Mostek e Signetics. Il termine *bus* indica genericamente una tecnologia di interfacciamento di dispositivi elettronici, da cui il nome VMEbus.

Un computer basato su standard VME [vedi foto] si presenta come un cestello (crate) dove si possono installare da 1 a 21 moduli che si connettono attraverso un circuito stampato (backplane) dove sono montati dei connettori a 96 poli.

Figura 5.1 – Crate VME a 21 slot

Ogni modulo è costituito da una scheda elettronica che può avere la funzione di master o di slave.
Una scheda è di tipo master se è in grado di assumere il controllo del bus VME e quindi di gestire il trasferimento di un dato da (o verso) un' altra scheda del sistema.

La scheda che va installata nella posizione più a sinistra del backplane (slot 1) ha una funzione particolare ed è detta "System Controller".
Questo modulo ha un duplice compito:

- Bus Arbiter: riceve le richieste di accesso al controllo del bus (Bus Request) da parte di potenziali master e concede tale accesso (Bus Grant) non appena è completato il ciclo di bus corrente.
- Gestisce le richieste di Interruzione del flusso di elaborazione corrente (Interrupt Request) da parte di moduli che richiedono il servizio di eventi hardware.

Il VME è un sistema che supporta il multiprocessing: ciò significa che diverse schede con capacità di "intelligenza locale" possono essere installate sullo stesso backplane e trasferire dati sul bus senza interferire con l'attività di elaborazione del System Controller (se non per chiedere la mastership temporanea del bus). Il protocollo di arbitraggio del bus è completamente hardware e pertanto è molto veloce: il criterio di priorità, nel caso in cui due moduli chiedano contemporaneamente di assumere il controllo del bus, è determinato dalla posizione fisica delle schede sul backplane. Infatti, il segnale di Bus Grant generato dal System Controller si propaga lungo il backplane seguendo l'ordine di installazione delle schede (daisy chain):
in pratica, se una scheda riceve il segnale Bus Grant senza avere il segnale Bus Request attivo trasmette il Grant alla scheda successiva, altrimenti

assume la mastership del bus attivando il segnale BGACK (Bus Grant Acknowledge) e quindi inizia il traferimento dei dati.

Il VME, nello standard originale, è un bus di tipo asincrono: ciò significa che il ciclo di trasferimento di un dato si adatta automaticamente al tempo di risposta del modulo slave secondo un protocollo di handshake hardware (Address Strobe, Data Transfer Acknowledge).
Le specifiche del VME hanno subito una significativa evoluzione nel corso degli anni pur mantenendo una piena compatibilità con lo standard originale; la larghezza del bus dati, ad esempio, è passata dalla prima versione a 16 bit (sul solo connettore P1) all'attuale che consente trasferimenti a 64 bit (possibili multiplexando dati e indirizzi nella modalita Block Transfer Mode).
Le specifiche 2eSST (Two edge Source Synchronous Transfer), introdotte recentemente, prevedono la possibilità di trasferire i dati in modo sincrono usando entrambi i fronti del segnale di Address Strobe. Di conseguenza, la banda passante nominale è aumentata dai 40 Mbytes/sec originali agli attuali 320 Mbytes/sec. [17, 18, 19]

Caratteristiche del VME
Un elenco delle principali caratteristiche del VMEbus è rappresentato nella seguente tabella.

Oggetto	Specifiche	Note
Architettura	Master/Slave	
Meccanismo di trasferimento	Asincrono, con multiplexing opzionale	Non c'è una sincronizzazione centrale del clock
Range di indirizzamento	16-bit (Short I/O) 24-bit (standard) 32-bit (extended) 64-bit (long)	Range di indirizzo selezionabile dinamicamente
Larghezza dei dati	8, 16, 24, 32 o 64 bit	Larghezza selezionabile dinamicamente
Trasferimenti di dati non allineati	Supportato	Compatibile con la maggior parte dei microprocessori
Rilevamento dell'errore	Supportato	Utilizzando Berr (opzionale)
Velocità trasferimento dati	0 - 320 Mbyte/sec	
Interrupts	7 livelli	Sistema di priorità degli interrupt con Status/Id
Capacità di multiprocessore	1 - 21 processori	Arbitraggio flessibile del bus

Oggetto	Specifiche	Note
Capacità di diagnostica del sistema	Supportato	Utilizzando Sysfail (opzionale)
Standard meccanici	Singola altezza, Doppia altezza	160x100 mm eurocard 160x233 mm eurocard connettori DIM 603-2
Standard internazionali	Si	IEC 821, IEEE 1101, IEEE 1014

Tabella 5.1 Principali caratteristiche del VME

Tra le caratteristiche principali merita di essere evidenziata la possibilità di gestire fino a 7 livelli di interrupt e di configurare dinamicamente l'ampiezza del bus dati.
Il controllo del trasferimento di dati avviene su 4 sotto-bus, chiamati Data Transfer Bus, Data Transfer Arbitration Bus, Priority Interrupt Bus, e Utility Bus.

Il VME64: un'architettura ibrida per accrescere la larghezza di banda.
I bus per microcomputer solitamente ricadono all'interno di una delle due seguenti categorie: "multiplexed" e "non-multiplexed". I bus multiplexed condividono lo stesso set di pin per gli indirizzi e per le linee dati: durante la prima parte del ciclo viene trasferito l'indirizzo, mentre i dati vengono trasferiti durante la seconda metà del ciclo. Viceversa, le architetture non-multiplexed godono di un set di pin dedicato ai bit di indirizzo, ed uno ai bit di dati.

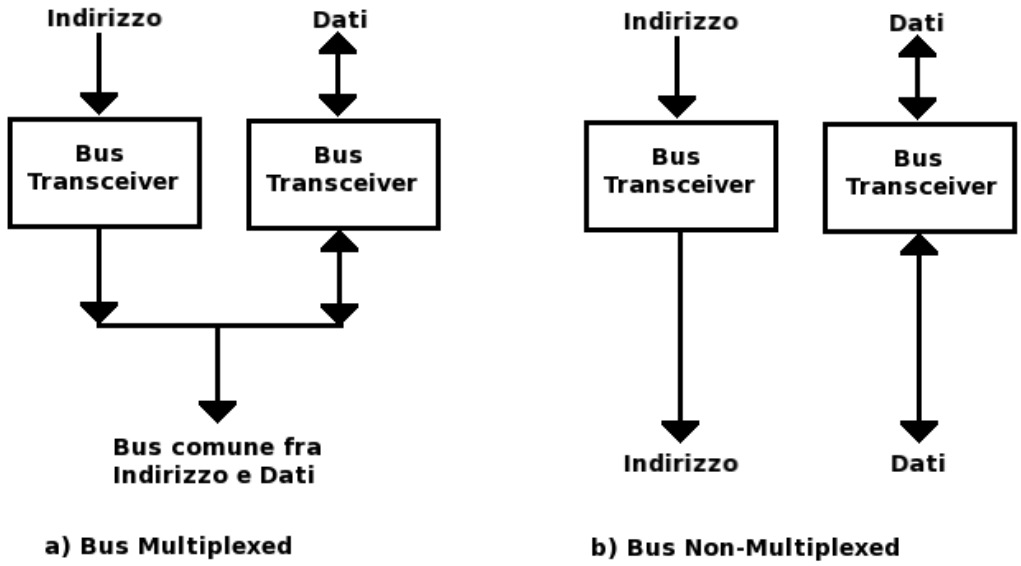

Figura 5.2 Multiplexed e non multiplexed buses.

Le specifiche VME64 prevedono l'utilizzo di entrambe le modalità. Per indirizzi e trasferimenti di dati fino a 32 bit, esso utilizza la modalità standard non-multiplexed, mentre il multiplexing è attivato per trasferimenti in block mode (MBLT) ottenendo di fatto una larghezza del bus dati di 64 bit.

Questo può ridurre (per trasferimenti in streaming di blocchi di grandi dimensioni) il rapporto tra cicli di indirizzo e cicli di dati dal 50% a meno dello 0.5%, diminuendo quindi drasticamente l'overhead.

Il Backplane
I moduli VME sono collegati tra loro attraverso un backplane. A seconda che il sistema sia a 3U o 6U il backplane ha una o due file di connettori a 96 poli (J1 e J2). Se il sistema e' conforme alle specifiche VME64, i connettori J1 e J2 hanno due file di pin laterali addizionali che estendono il numero di pin a 160.
I connettori sono conformi agli standard industriali DIN 41612.
Il backplane del VME può essere lungo fino a 500mm. La massima velocità del bus è limitata di fatto dal carico capacitivo e induttivo delle piste del circuito stampato del backplane e dalle caratteristiche elettriche dei circuiti driver.
Appropriati resistori di terminazione devono essere applicati alle linee del bus per ridurre i disturbi (skew) sulle commutazioni.

Il bus VME è diffuso nelle applicazioni di controllo perchè riesce a coniugare delle buone prestazioni in termini di banda passante con un' architettura modulare e una notevole robustezza meccanica. [18, 19]

5.2 Schede utilizzate

Figura 5.3 Crate VME e schede utilizzate

5.2.1 MVME3100

La MVME3100 è una scheda VME a singolo slot basata sul processore integrato MPC8540 PowerQUICC III. Essa fornisce interfacce ATA (sATA), USB 2.0, 2eSST VMEbus, due PMC sites 64-bit/100Mhz, fino a 256Mb di Flash, due 10/100/1000 Ethernet, una 10/100 Ethernet, e 5 porte seriali. Questa scheda supporta I/O sia frontale che posteriore, e un singolo modulo SODIMM per la memoria DDR. L'accesso all'I/O posteriore è disponibile con un RTM (Rear Transition Module).
Il protocollo 2eSST offre una larghezza di banda fino a 320MB/s, e mantiene la retrocompatibilità con VME64 e VME32. Ma la MVME3100 non consiste solo in un veloce trasferimento di dati sul bus VME; essa fornisce anche performance bilanciate sul processore, un sottosistema di memoria, bus locali, e sottosistemi di I/O. Il processore integrato Freescale MPC8540, con frequenze fino a 833MHz, si adatta bene ad applicazioni intensive di I/O ed elaborazione dati; esso combina un core a basso consumo e cache L2 sul chip con controlloti integrati per memoria, PCI-X, DMA, Ethernet, ed I/O per apparecchi locali.

Un elenco riassuntivo delle caratteristiche è presente nella tabella in appendice A. [20]

5.2.2 Xycom XVME-566

La scheda XVME-566 prodotta dalla Xycom è un modulo VME per input analogico ad alte prestazioni, in grado di effettuare conversioni analogico-digitali con risoluzione a 12 bit. Questo modulo fornisce 32 canali single-ended oppure 16 canali differenziali in input.
Tutti i canali in ingresso possono essere configurati come unipolari o bipolari con i seguenti range: 0-10V, +/-5V, +/-10V. La scheda consente di impostare via jumper un guadagno di base, valido per tutti i canali, con possibili valori 1, 4, o 10; esiste poi un guadagno programmabile dinamicamente per ogni canale, che va a moltiplicare il gain di base di 1, 2, 5, o 10 volte.
La raccolta di fino a 32K (32767) campioni a 12-bit è possibile attraverso 64Kbyte di RAM dati dual-ported. L'architettura del modulo consente tempi di conversione dati a 12 bit di 10µsec, e un flusso di campioni a 100KHz; con conversioni a 8bit invece, il tempo di acquisizione di 7µsec consente un flusso fino a 144KHz).
Una sequence RAM di 256 byte consente poi all'utente di specificare l'ordine in cui deve essere effettuata l'acquisizione sui vari canali di ingresso; una volta inizializzata, la XVME-566 può quindi operare ed acquisire dati senza l'intervento dell'operatore.
Il System Timing Controller (STC) Am9513A presente sul modulo è poi in grado di generare degli interrupts sul bus VME, e di fornire vari segnali necessari per le acquisizioni, come ad esempio il Sample Clock, il Trigger Clock, e l'Event Counter.

Le schede XVME-566 utilizzate in questo progetto tuttavia sono leggermente modificate, in modo da disconnettere l'external trigger input, e far fuoriuscire il clock di sample dell'ADC attraverso questo pin. [21]

5.2.3 Xycom XVME-240

La scheda XVME-240 prodotta dalla Xycom è un modulo VME per Input/Output Digitale. Essa fornisce 8 porte Input/Output (bidirezionali) da 8 bit ciascuna, la cui direzione può essere programmata anche dinamicamente. I 64 canali disponibili possono quindi essere impostati come input o come output a gruppi di 8, agendo su un bit di Port Direction. [22]

5.2.4 Xycom XVME-220

La scheda XVME-220 prodotta dalla Xycom è un modulo VME per Output Digitale. Essa fornisce 32 canali DC Output, isolati otticamente; i 32 canali sono programmabili indipendentemente, e sono in grado di gestire carichi resistivi o induttivi fino a 30V e 100mA. In aggiunta, ogni output utilizza uno schema a collettore aperto in modo da permettere una gestione del carico sufficiente, e una soppressione dei transitori in modo da minimizzare la possibilità di danni dovuti al sovravoltaggio. [23]

5.2.5 Stepper-Motor Controller

Per l'utilizzo di motori passo-passo, è stata utilizzata una scheda VME sviluppata all'interno dei Laboratori Nazionali di Legnaro. Questa permette di controllare fino ad 8 assi di rotazione per ogni scheda; l'interfaccia di controllo è basata su registro, con la scheda risiedente nello spazio di indirizzamento A16 del VME (shortIO).
Ogni modulo VME di questo tipo fornisce un set di funzionalità di controllo per ogni asse:

- avvia/ferma il movimento
- segnali dagli switch di finecorsa
- rotazione in senso orario o antiorario
- velocità (step/s)
- numero desiderato di step.

5.3 Il Sistema Operativo VxWorks

I moderni sistemi in tempo reale sono basati sui concetti di multitasking e di comunicazioni fra task (o intertask). Un ambiente multitasking permette ad una applicazione real-time di essere costruita come una serie di task indipendenti, ciascuno dei quali con il proprio processo di esecuzione e il

proprio set di risorse di sistema. Le facility per la comunicazione fra task consentono ai vari processi di sincronizzarsi e di comunicare, ai fini di coordinare e svolgere le loro attività.
Un altro fattore chiave nei sistemi real-time è il meccanismo di gestione degli interrupt, in quanto essi sono il meccanismo per eccellenza per informare il sistema su eventi esterni.
E' quindi di fondamentale importanza garantire che esista un tempo massimo entro il quale il sistema risponda a determinati segnali ed esegua le elaborazioni richieste, rispettando le priorità di ciascuna richiesta.
Nel caso del sistema operativo VxWorks, le strutture per gestire la comunicazione fra task sono diverse, e comprendono semafori, code di messaggi, e altro. Le richieste di interrupt sono invece gestite dalle ISR (Interrupt Service Routine), le quali vengono eseguite in un contesto speciale separato, al di fuori di ogni altro task, al fine di garantire risposte il più veloce possibile agli interrupt.

5.3.1 Introduzione al SO

Il sistema operativo VxWorks è un sistema operativo real-time sviluppato e venduto dalla Wind River Systems a partire dalla metà degli anni '80. Esso è nato come integrazione al kernel VRTX, al quale forniva strumenti propri di un sistema operativo ed un ambiente di sviluppo. Successivamente venne sviluppato anche un proprio kernel, svincolando quindi il sistema VxWorks dal VRTX, e un sistema operativo più piccolo, veloce e leggero del precedente.

I sistemi UNIX e Windows sono eccellenti sistemi operativi per lo sviluppo di programmi e per moltissime applicazioni interattive; tuttavia però, non sono decisamente appropriati per le applicazioni di tipo real-time. D'altro canto i tradizionali sistemi operativi real-time forniscono ambienti molto scarsi per lo sviluppo di applicazioni e per i componenti non-real-time, come ad esempio le interfacce grafiche (GUIs, Graphical User Interfaces).
Invece di provare a creare un unico sistema operativo in grado di fare tutto, la filosofia WindRiver è quella di utilizzare due sistemi operativi complementari e cooperanti (VxWorks e UNIX, o VxWorks e Windows) e far fare a ciascuno ciò in cui riesce meglio. VxWorks quindi gestisce le questioni in cui il real time è un fattore critico, mentre la macchina host su cui si trova l'altro sistema operativo viene dedicata allo sviluppo dei programmi e per le applicazioni definite non-time-critical.
VxWorks comunque può essere ridimensionato in modo da includere esattamente le combinazioni di caratteristiche e servizi necessari alle applicazioni richieste. Durante lo sviluppo si possono includere funzioni addizionali per velocizzare il lavoro, e poi escluderle nella versione finale dell'applicazione.
E' possibile lavorare con uno sviluppo incrociato (cross), utilizzando la macchina host per modificare, compilare, e salvare il codice real-time, e poi eseguirlo e farne il debug sulla macchina VxWorks. L'applicativo finale infatti

non necessita più della macchina host, e può essere eseguito indipendentemente.
Tra i prodotti notevoli che utilizzano il sistema operativo VxWorks, vi sono diverse strumentazioni per le reti, alcuni robot (principalmente industriali), alcuni aerei (tra cui prodotti Boeing) ed elicotteri militari Apache, il sistema iDrive della BMW, e molto altro, anche in ambiti di grandi acceleratori e telescopi.
Tra gli ambiti più rinomati di utilizzo, vi è sicuramente anche quello dell'industria aerospaziale; tale sistema operativo è stato utilizzato all'interno di diversi spacecraft, tra cui i Rover Sojourner Mars Pathfinder, Spirit ed Opportunity, il Phoenix Mars Lander, e la sonda spaziale Deep Impact. [24]

5.3.2 Caratteristiche

Il nucleo del sistema operativo (kernel), chiamato Wind, consente il multitasking con uno scheduling di tipo “preemptive priority”, sincronizzazione fra task e facility per la comunicazione, supporto alla gestione degli interrupt, timers e gestione della memoria.
Il sistema inoltre è compatibile con le specifiche POSIX e con gli standard ANSI C, ed implementa un sistema di I/O veloce ed efficace. Include inoltre drivers per reti, RAM, tastiere, display, periferiche SCSI, periferiche IDE e porte parallele; l'installazione di ulteriori driver avviene dinamicamente, e quindi non richiede il riavvio del sistema operativo. Esso supporta anche diversi tipi di File System, sia locali che di rete (NFS).
La customizzazione del sistema operativo viene effettuata all'interno di un ambiente di sviluppo chiamato Workbench (o Tornado), creato appositamente per gli sviluppi di tipo incrociato (cross-development).
Tornado è l'ambiente di sviluppo per VxWorks 5.x, e comprende strumenti per lo sviluppo di applicazioni (cross-compilatori e programmi associati), un ambiente di sviluppo per facilitare la costruzione di progetti e l'esecuzione / debug / monitoraggio di applicazioni, ed il simulatore VxSim.
Workbench invece è l'ambiente di sviluppo, costruito su Eclipse, per VxWorks 6.x e comprende molto altro. Fra le cose meritevoli di essere menzionate rientrano il debugger, una Kernel Shell per VxWorks, un VxWorks kernel configurator, un tool di analisi run-time, ed il simulatore VxSim.
VxSim, il simulatore per VxWorks, è un programma che simula una macchina target VxWorks per utilizzo a scopo di test. Le sue capacità sono identiche a quelle di un vero sistema VxWorks in esecuzione in una macchina target remota. E' quindi possibile eseguire una applicazione e ricostruire l'immagine VxWorks esattamente nello stesso modo di un qualsiasi altro ambiente equivalente con sviluppo incrociato (cross-development).
La differenza tra VxSim e un target VxWorks remoto è che su VxSim l'immagine esegue se stessa sulla macchina host come se fosse un processo dell'host. Non vi è alcuna simulazione delle istruzioni, poiché il codice eseguito rispetta l'architettura CPU della macchina host. E, poiché l'interazione con l'hardware del target non è possibile, lo sviluppo di device drivers non è adatto alla simulazione. Comunque sia, lo scheduler VxWorks

viene implementato all'interno del processo VxSim, mantenendo la reale interazione fra task e rispettando le priorità e il preemption. Ciò significa che ogni applicazione scritta in stile portabile e con una interazione con l'hardware minimale, dovrebbe risultare portabile tra VxSim e VxWorks. [25]

Gestione dei Task

Il multitasking è il meccanismo fondamentale per far sì che le applicazioni controllino e reagiscano ad eventi multipli e discreti provenienti dall'esterno. Il kernel real-time di VxWorks (Wind), fornisce l'ambiente di base per questo meccanismo.

Il multitasking infatti crea l'apparenza di avere molti processi in fase di esecuzione contemporaneamente, quando invece il kernel alterna la loro esecuzione in base ad un algoritmo di scheduling; ciascuno di questi processi può essere chiamato task. Ogni task è associato ad un proprio “contesto”, che comprende lo stato della CPU e le risorse di sistema che vengono interessate dal task quando questo viene messo in esecuzione. Ad ogni cambio di task in esecuzione, il suo contesto viene salvato nel Task Control Block (TCB), per venire in seguito ripristinato. Una particolarità del sistema VxWorks è che lo spazio di indirizzamento in memoria non fa parte del contesto; tutto il codice infatti viene eseguito in un singolo spazio di indirizzamento, limitando quindi la versatilità della memoria virtuale.

Lo scheduling, come già accennato poco fa, è l'algoritmo che assegna l'utilizzo della CPU ai processi; per default viene utilizzata una procedura di programmazione preventiva basata sulle priorità (priority-based preemptive scheduling), ma è anche possibile utilizzarla assieme ad un più classico round-robin.

Con lo scheduler di default ogni task ha una priorità, e il kernel si assicura che la CPU è assegnata al processo pronto ad essere eseguito con la priorità più alta. Questo metodo viene definito pre-emptive (preventivo) poiché se diventa pronto ad essere eseguito un task con priorità più alta rispetto a quella del task in esecuzione, il kernel salva il contesto del task attuale e ripristina il contesto del processo con priorità maggiore, riprendendo quindi la sua esecuzione.

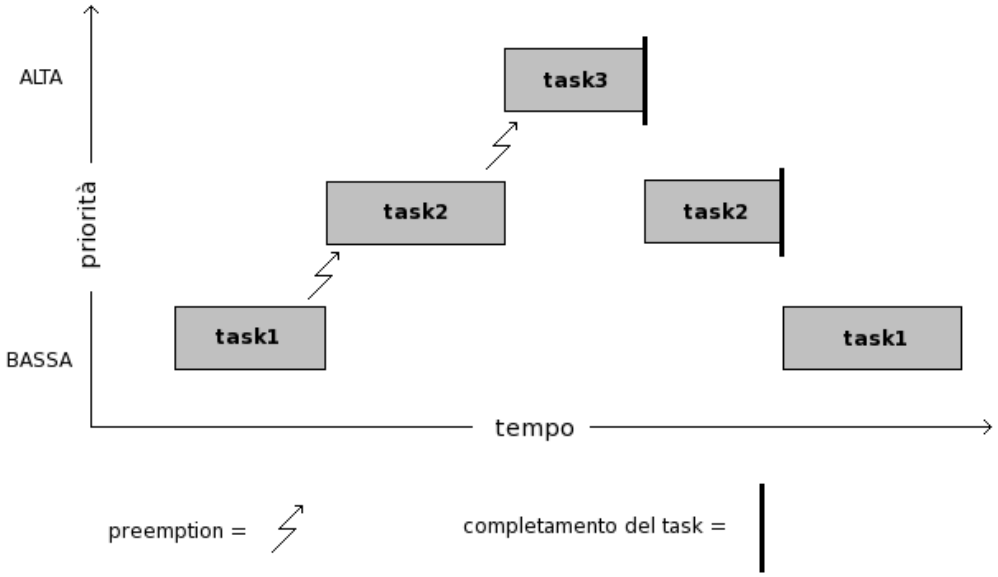

Figura 5.4 Scheduling di tipo Priority Preemption

Il kernel wind dispone di 256 livelli di priorità, numerati da 0 a 255, dove 0 è il livello a priorità più alta e 255 quello più basso; ad ogni task viene assegnata una priorità nel momento in cui vengono creati, ma questa può venire cambiata dinamicamente.
Come già accennato, il priority preemption può essere utilizzato assieme ad uno scheduling di tipo round-robin; in questo modo, viene gestita anche la condivisione tra tutti i task della stessa priorità.
Senza round-robin, un singolo task può “usurpare” l'utilizzo del processore e tenere bloccati tutti gli altir processi aventi la stessa priorità.
Con il round-robin invece, i task aventi lo stesso livello di priorità vengono intervallati a seconda di un approccio conosciuto come “time slicing”. Ogni task infatti viene eseguito per un certo lasco di tempo (detto time slice), dopo di ché lascia spazio ad un altro task della stessa priorità, se presente; e così via per tutti gli altri, a rotazione. Ogni task, una volta giunto al termine del proprio time slice, non viene rieseguito finchè tutti i processi aventi la stessa priorità non sono stati anch'essi eseguiti.

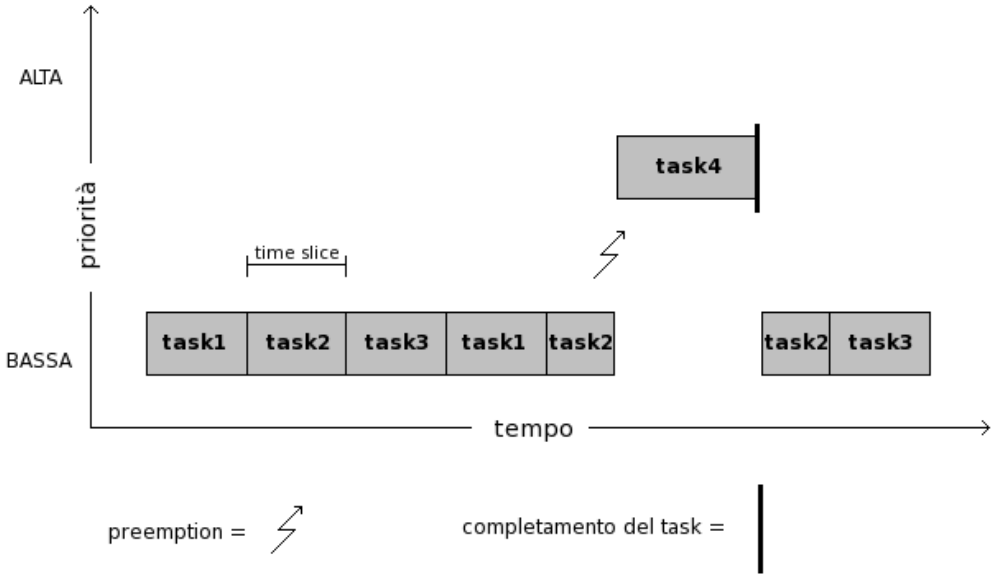

Figura 5.5 Scheduling di tipo Round Robin

Quando un task viene creato, ad esso vengono assegnati una stringa ASCII di qualunque lunghezza ed un Task ID, valore numerico di 4 byte. Questo valore viene utilizzato per identificare il processo in tutte le routine di gestione dei task.
Esistono tuttavia anche diversi task di sistema, come ad esempio: il task root (primo task ad essere eseguito dal kernel, che va ad inizializzare alcuni altri task di sistema), il logging task (usato dai moduli di VxWorks per loggare messaggi di sistema senza bisogno di effettuare operazioni di I/O), il task exception (per la gestione delle eccezioni), il task network, il task target agent, più alcuni task opzionali (per shell, telnet, o altro). [25, 26]

Comunicazione fra Task
Il sistema VxWorks è dotato di un ampio set di meccanismi per la comunicazione fra task, fra cui:

- Memoria condivisa, per la semplice condivisione di dati
- Semafori, per meccanismi di mutua esclusione e sincronizzazione
- Code di messaggi e “pipes” per lo scambio di messaggi all'interno di una CPU.
- Sockets e procedure remote di chiamata, per la comunicazione fra task attraverso una rete
- Segnali, per la gestione delle eccezioni.

Il modo più ovvio per scambiare dei dati fra task è chiaramente l'accesso a delle strutture dati condivise; ed essendo che tutti i programmi utilizzano uno spazio di memoria unico e lineare, la condivisione di strutture dati è estremamente pratica ed immediata da effettuare.
Ma tuttavia,sebbene lo spazio di indirizzamento comune semplifichi lo scambio dei dati, esso richiede che gli accessi alla memoria vengano sincronizzati e si evitino contese sull'accesso alle risorse. Tra le varie possibilità per ottenere accesso esclusivo alle risorse, vi sono quelle di

disabilitare gli interrupt, disabilitare il preemption, o bloccare le risorse con dei semafori.
Disabilitare gli interrupt garantisce che nessun altro task avrà accesso alla CPU, garantendo così un utilizzo esclusivo a tutte le risorse fino a che si effettua la riabilitazione degli interrupt; questo tipo di meccanismo è tuttavia sconsigliato per lunghi tratti di codice, in quanto impedisce al sistema di rispondere ad eventi esterni.
Un modo meno restrittivo è la disabilitazione del preemption. Con questa tecnica, è concesso alle Interrupt Service Routine (ISR) di intervenire, mentre i task a priorità più alta rimangono ancora bloccati e in attesa.
I semafori invece sono il metodo più efficace per garantire il semplice accesso esclusivo ad una risorsa condivisa (mutua esclusione), o per coordinare l'esecuzione dei task tramite eventi esterni (sincronizzazione). I semafori utilizzati sono principalmente di 3 tipi: binari (generici, multiuso), mutual exclusion (ottimizzati per la mutua esclusione), o counting (in grado anche di contare).
Le code di messaggi (message queues) invece sono delle code FIFO di lunghezza variabile, utilizzate per lo scambio di messaggi (anch'essi di lunghezza variabile) fra task. Per dare una idea generica del funzionamento, basti pensare che diversi task possono inviare o ricevere messaggi da una stessa coda, che risulta quindi pubblica e condivisa; l'utilizzo di due code per lo scambio tra due soli task invece è un meccanismo efficace di comunicazione full-duplex fra di essi, utilizzandone una per ciascuna direzione.
Le “pipes” invece forniscono una interfaccia alternativa che passa attraverso il sistema di I/O; esse sono delle periferiche virtuali, che possono quindi essere utilizzate dai task attraverso delle routine standard di I/O.
La comunicazione attraverso la rete, come accennato poco sopra, viene effettuata con due metodi: i Socket, che vengono utilizzati come estremi di comunicazione, mandando dati da uno all'altro attraverso i protocolli tcp e udp, e le Remote Procedure Calls (RPC), che consentono ad un processo di chiamare una procedura sulla stessa macchina o su di una macchina remota.
Infine vi sono i segnali (signals), che vengono lanciati da un qualsiasi task o da una ISR, e che vengono ricevuti e gestiti attraverso una apposita Signal handler Routine dal task ricevente. Sono spesso lo strumento più appropriato per la gestione di errori ed eccezioni. [25]

Il sistema di I/O

Per quanto riguarda il meccanismo di input e output, questo può essere effettuato attraverso i File Descriptors (FD), oppure attraverso un buffer (stdio). Con “file” si può intendere sia un archivio logico su una periferica strutturata ad accesso casuale, sia una periferica non strutturata (o raw), come ad esempio un canale per comunicazioni seriale.
Le periferiche su cui effettuare operazioni di input/output sono di diverso genere, e possono essere periferiche seriali (es: terminali), pipes, periferiche di pseudo memoria, file system di rete (NFS, Network File System), periferiche di rete non-NFS (rsh o ftp), periferiche organizzate come una

sequenza di di blocchi di dati individualmente accessibili (File System, RAM disk, SCSI), o di tipo socket.
In via generale comunque vi sono diverse differenze tra un sistema VxWorks e il sistema host anche per quanto riguarda la gestione dell'I/O. Per quanto riguarda la configurazione delle periferiche, il sistema prodotto dalla Wind River, a differenza degli altri, consente di installare e rimuovere driver dinamicamente. Poi, mentre in una generico sistema Unix o Windows i descrittori dei file (FD, File Descriptor) sono unici per ogni processo, in VxWorks essi sono variabili globali accessibili da ogni task. Le routine dei driver sotto Unix sono eseguite in modalità system, e quindi non preemptive; nell'altro sistema invece essi sono preemptive, e vengono eseguiti all'interno del contesto del task che li ha invocati. [25]

Capitolo 6 – Acquisizione ed elaborazione dati

Per l'utilizzo del sistema di diagnostica si richiede la creazione di una applicazione EPICS di tipo Input Output Controller per VxWorks in grado di gestire tutte le schede hardware, di acquisirei dati e di controllare la diagnostica attraverso i record del database; si richiede inoltre la creazione dei programmi per la misura di emittanza, e la creazione delle interfacce per tutto il sistema.

In questo capitolo vengono quindi descritte le procedure per la creazione dell'applicazione, la configurazione dei record del database e delle subroutine collegate, lo script di avvio per la corretta inizializzazione del sistema, i programmi di acquisizione e di calcolo per la misura di emittanza, e i parametri con cui configurare le interfacce.

6.1 Creazione dell'applicazione

Il passo preliminare alla creazione del nostro IOC, è l'installazione sul pc host (pc con processore Intel E7400, 2GB ram, e sistema operativo Linux Fedora10) del software EPICS base configurato per la cross-compilazione per il sistema target (VME con processore mpc8540 e sistema operativo VxWorks 6.7).
L'applicazione iocCore è stata creata con un semplice script in perl, incluso nella versione base di EPICS, chiamato makeBaseApp.pl (make base application).
Esso viene eseguito all'interno della cartella app nella directory del progetto dalla riga di comando con la seguente sintassi:

- `makeBaseApp.pl -t ioc app`

(dove app indica il nome dell'applicazione)

- `makeBaseApp.pl -i -t ioc app`

e indicando successivamente i seguenti parametri (quando richiesti)
Target: `vxWorks-mpc8540`

IocBoot: `app`

Una volta creata l'applicazione con questo script, si rende necessario modificare il file configure/RELEASE al suo interno, indicando il percorso su disco delle cartelle (applicazioni) contenenti pacchetti/librerie necessari alla nostra applicazione.
Nel nostro caso, si è reso necessario aggiungere 3 percorsi: uno chiamato XYCOMIOC, necessario per utilizzare il device support per le schede Xycom, uno chiamato MOTOR, necessario per interfacciarsi con la scheda motori, ed uno GENSUB, per permettere l'utilizzo di record di tipo genSub (general subroutine).
Bisogna poi aggiungere nel file appApp/src/Makefile tutti i file .dbd richiesti (device support e record support), le librerie, e i sorgenti delle routine scritte in c o c++. Alcuni esempi sono:
`app_DBD += motorRecord.dbd` (record support per i motori passo passo)
`app_DBD += devInfnMtr.dbd` (device support per la scheda motori)
`app_LIBS += xy566` (librerie per la scheda ADC Xycom 566)
`app_SRCS += beamProfileCorrection.c` (sorgenti della routine per la correzione del profilo del fascio)
Una ulteriore modifica da apportare allo stesso file è la modifica della riga
`PROD_IOC = app`
in
`PROD_IOC_vxWorks = app.`

Fatto ciò, l'applicazione è pronta per essere configurata. A questo punto, si richiede la configurazione del database e dello startup script.
Tutti i file .db e .substitutions che compongono il database vanno inseriti all'interno della cartella appApp/Db e all'interno del relativo Makefile.
Lo startup script invece si trova in iocBoot/iocapp/st.cmd, e deve essere configurato in modo da inizializzare correttamente l'iocCore, le varie schede hardware, ed il database.
I dettagli sulla configurazione di questi due componenti sono trattati nei paragrafi seguenti.
L'applicazione è poi pronta per essere compilata con il comando *make*, ed essere eseguita sul sistema VME.
I parametri con cui configurare l'avvio di VxWorks sono riportati in appendice C. [6, 27]

6.2 Configurazione del Database

Il database per la strumentazione diagnostica è composto da diverse decine di record, ciascuno con una propria funzione, e di tipo diverso.
Il modo più logico per spiegare l'intero database è quello di analizzare i record a seconda della loro funzione finale. Qui di seguito viene descritto il modo in cui i record vengono generati e come interagiscono fra di loro; tale meccanismo può essere compreso anche analizzando il codice dei vari file, riportato in appendice B.

Innanzitutto vi è la necessità di inizializzare l'ADC e di impostare le sue tempistiche di acquisizione. Questo viene fatto con i record TaDiag_Adco01_IntrRd e TaDiag_Adco01_IntrSt, creati richiamando il file xy566base.db da common.substitutions. In particolare, il campo SCAN del record TaDiag_Adco01_IntrSt va a definire la frequenza con cui l'ADC deve iniziare una nuova sequenza di scansione tra i vari canali. Il fatto che sia impostata a ".1 second" indica quindi che per ogni variabile di processo avremo 10 acquisizioni al secondo.

Tutti i record legati alla faraday cup sono invece richiamati dal template faradaycup.substitutions.
Il file xy566gain.db crea il record TaDiagFacp01_GainSt, che permette di impostare un gain dinamico sull'ADC, con valore a scelta fra 1,2,5,10. Tuttavia, il valore di questa variabile si inizializza ad 1 e non varia, in quanto si è scelto di utilizzare un sistema hardware diverso e più completo per la gestione dei guadagni.
Il file xy566val.db inizializza il record TaDiagFacp01_CurrRd, che contiene il valore istantaneo di corrente letta dal segnale della faraday cup. Il campo RVAL (raw value) di questo record contiene il numero dello step dell'adc corrispondente al valore letto; impostando i campi ASLO ed EGU a 0.00244 e nA, si fa in modo che il valore VAL del record corrisponda al valore di corrente reale (nel caso il guadagno sia il più alto possibile).
Il file 240port.db va a creare il record TaDiagFacp01_GainBit_port0, che stabilisce se la porta 0 della scheda Xycom 240 (bit 0-7) sia in input o in output.
Il file xybo.db genera i record TaDiagFacp01_GainBit_out0 e out1, che corrispondono ai 2 bit che vanno a pilotare il guadagno, in uscita in formato 0/5V dalla Xycom 240, e il TaDiagFacp01_out0 che invece va a stabilire se la faraday cup deve essere inserita o disinserita dalla traiettoria del fascio impostando un bit in uscita sulla Xycom 220.
Il file FCaverage.db invece contiene diversi record. Il record TaDiagFacp01_CurrRda è un record di tipo calc che va a moltiplicare l'informazione di corrente letta dall'ADC per un opportuno valore 1/10/100/1000 a seconda dell'impostazione di guadagno attuale; il suo campo VAL corrisponderà quindi alla corrente reale che colpisce la coppa di faraday. Il valore di guadagno viene letto dalla variabile TaDiagFacp01_CurrRdGain.
Esistono poi 3 record di tipo compress, TaDiagFacp01_CurrRdb, TaDiagFacp01_CurrRdc e TaDiagFacp01_CurrRdd che, utilizzando un meccanismo ad array circolari, vanno a calcolare le medie dei valori di corrente dell'ultimo minuto. Essi sono quindi un vettore con le 6 medie degli ultimi 6·10secondi, uno con le 2 medie degli ultimi 2·30secondi, ed il valore con la media degli ultimi 60secondi. Il modo in cui questi valori vengono passati da un record all'altro si può osservare nello schema seguente.

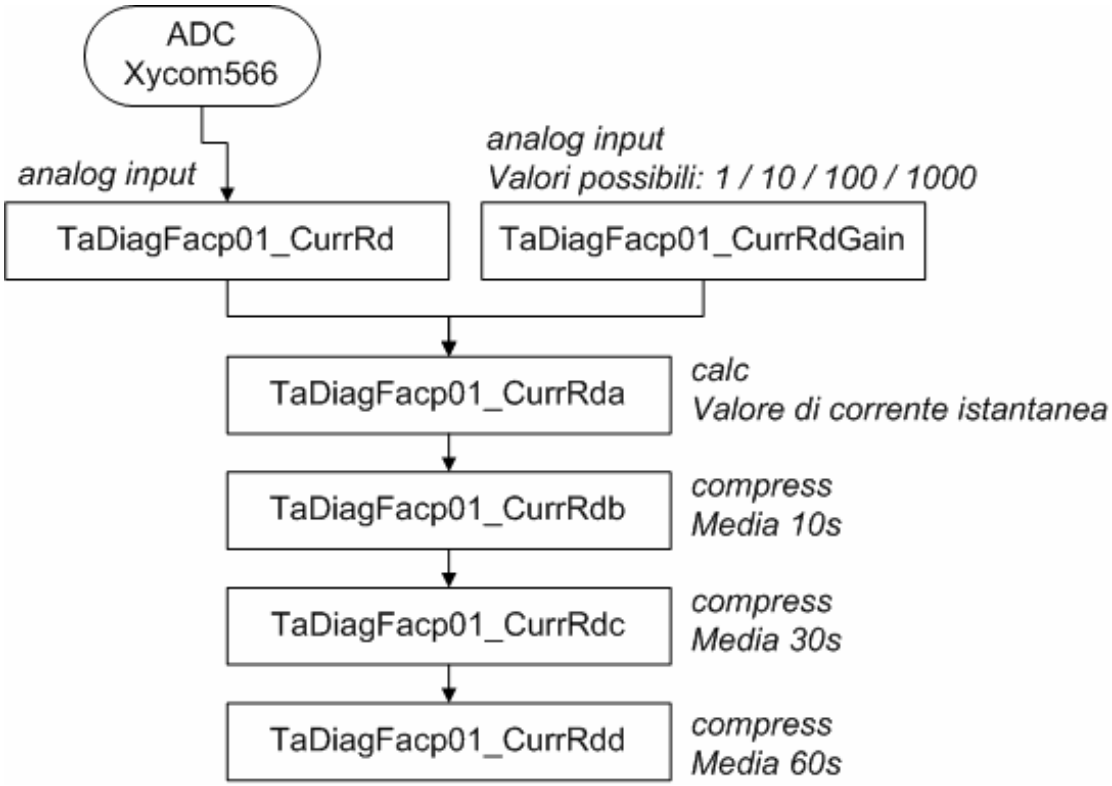

Figura 6.1 Meccanismo di calcolo del valore di corrente istantaneo e medio

Il valore di guadagno viene stabilito dalla variabile numerica TaDiagFacp01_HwGainSt00, avente come possibili valori 1,2,3 o 4. Il cambio di valore di questo record causa il processamento del record di tipo general subroutine TaDiagFacp01_HwGainSt01; ad ogni processamento del record viene eseguita una subroutine scritta in C che, a seconda del valore della variabile TaDiagFacp01_HwGainSt00, va a scrivere opportuni valori di guadagno (1/10/100/100) su TaDiagFacp01_CurrRdGain, si accerta che la porta 0 della Xycom240 sia in output, scrive i due bit che vanno fisicamente a cambiare il guadagno dell'amplificazione (bit 0 e 1 della Xycom 240), e informa l'operatore dell'avvenuto cambio di guadagno attraverso un record di tipo stringa (TaDiagFacp01_GainStatus).

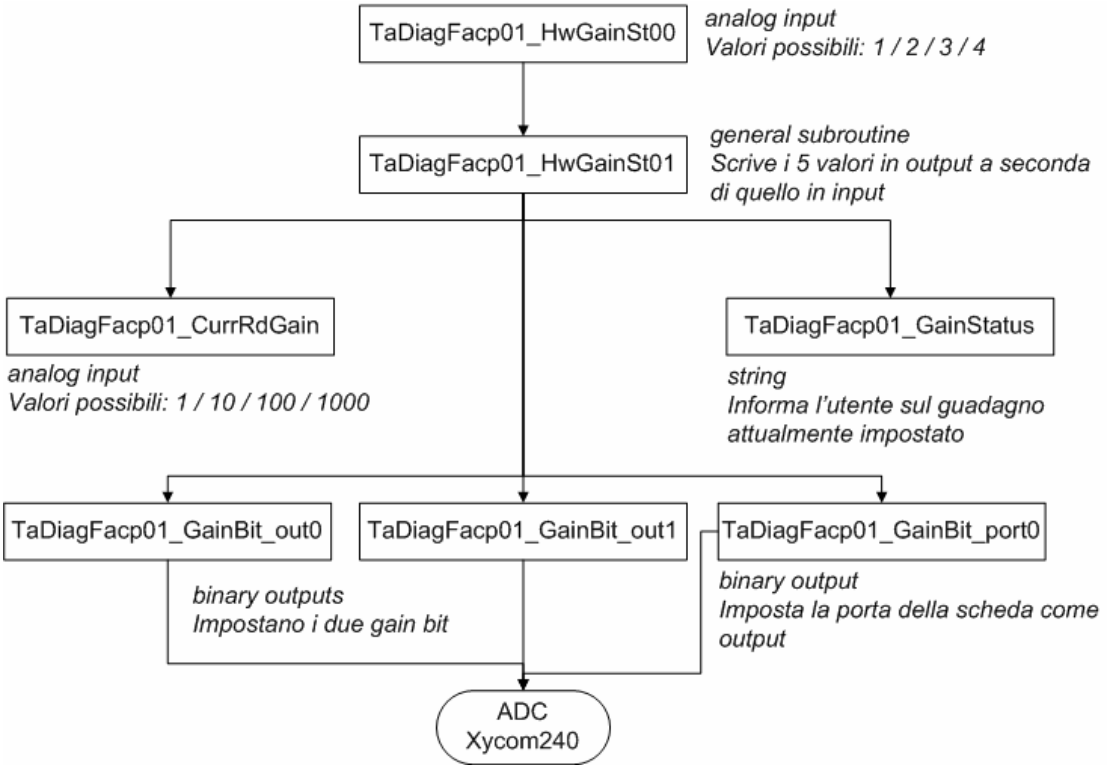

Figura 6.2 Meccanismo per il cambio di guadagno della coppa di Faraday

Tale subroutine scritta in C, viene salvata nella cartella appApp/src con il nome faradayCupGainSet.c, e viene anche inserita all'interno del relativo *makefile* tra gli *app_SRCS*. Essa viene invocata dal record genSub con una chiamata per nome; ciò è reso possibile dalla funzione *epicsRegisterFunction* presente all'interno del file.
Osservando il seguente codice, si può vedere come la subroutine vada a leggere e scrivere i valori sui link in ingresso e in uscita, indicati nei campi *A* (ingresso) e *VALA*, *VALB*, *VALC*, ... (uscita) del record.

```
long faradayCupGainSet( genSubRecord *pgsub )
{
  double *gain;
  gain = (double *)pgsub->a;
  double gainval= *gain;

[...]

  port0=1;
  if ((gainval>3.5)&&(gainval<4.5))
  {
    out0=1;
    out1=1;
    gaincalc=1000;
    strcpy(status, "Gain D. Max=10.000nA. Min=20nA.");
  }
  else if
```

```
[...]

  memcpy( pgsub->vala, &port0, sizeof(short));
  memcpy( pgsub->valb, &out0, sizeof(short));
  memcpy( pgsub->valc, &out1, sizeof(short));
  memcpy( pgsub->vald, &gaincalc, sizeof(double));
  memcpy( pgsub->vale, &status, 40*sizeof(char));

  return(0);
}

epicsRegisterFunction (faradayCupGainSet);
```

Esiste poi un integratore per la corrente di faraday cup. Un record di tipo calc (TaDiagFacp01_InteRd), processato ogni decimo di secondo, va a sommare al valore integrale il valore di corrente attuale opportunamente convertito in coulomb/secondo e moltiplicato per una variabile booleana (TaDiagFacp01:InteSt), corrispondente all'interruttore on/off dell'integratore. Nel caso la variabile booleana sia impostata a 1, essa integra il valore di corrente; nel caso questa sia impostata a 0, essa mantiene il proprio valore.

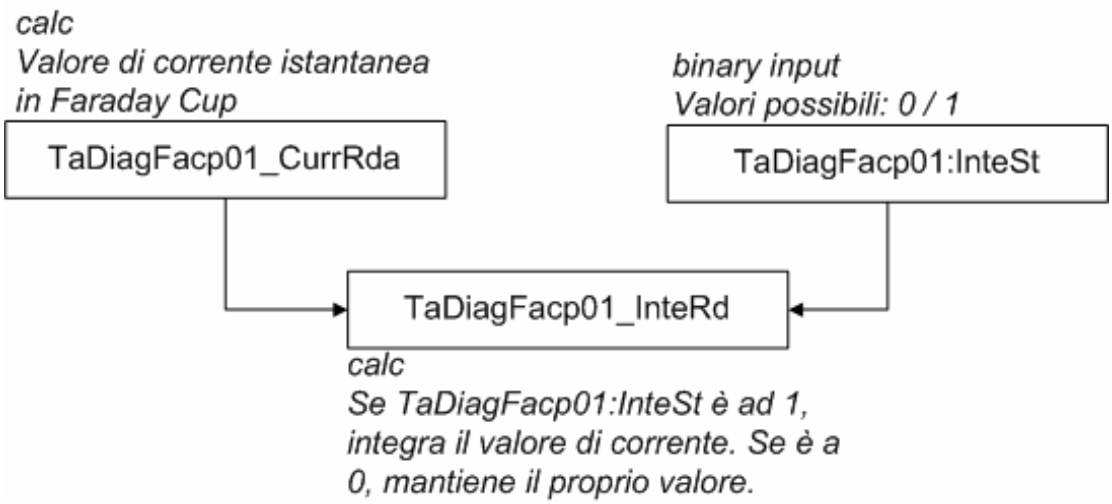

Figura 6.3 Meccanismo di funzionamento dell'integratore

I record per il profilatore di fascio sono invece generati dai file beamprofiler.substitutions e provasubgen.substitutions.
Innanzitutto vi sono i due record per il guadagno dinamico dell'ADC (TaDiagBmpr01_Hgri_GainSt e TaDiagBmpr01_Vgri_GainSt) che, analogamente alla coppa di faraday, hanno sempre valore 1.
Il file xy566wave.db crea due vettori da 40 valori, contenenti i valori raw letti nelle due griglie, chiamati TaDiagBmpr01_Hgri_CurrRaw e TaDiagBmpr01_Vgri_CurrRd.
Il file waveform.db crea poi un terzo array, sempre di 40 valori, chiamato TaDiagBmpr01_Hgri_CurrRd.

Il motivo di questa scelta è che, mentre la griglia verticale non presenta difetti e i valori letti possono essere visualizzati direttamente su interfaccia, quelli provenienti dalla griglia orizzontale presentano delle imperfezioni dovute a dei problemi meccanici di realizzazione. Nel caso due fili adiacenti si tocchino, infatti, è possibile che in uno di questi non si riesca a leggere alcun valore, mentre l'altro vada in saturazione. Per risolvere questo problema, si è optato per la correzione via software dei valori, effettuando la media tra la corrente letta dei due fili adiacenti funzionanti.
La correzione viene fatta dal record TaDiagBmpr01_Hgri_CurrElab, generato da dbgensub.db, che ad ogni processamento esegue la subroutine *beamProfileCorrection* scritta in C; questa va a leggere i dati della griglia orizzontale dal record TaDiagBmpr01_Hgri_CurrRaw, effettua le opportune correzioni, e va a scrivere i dati per l'interfaccia su TaDiagBmpr01_Hgri_CurrRd.

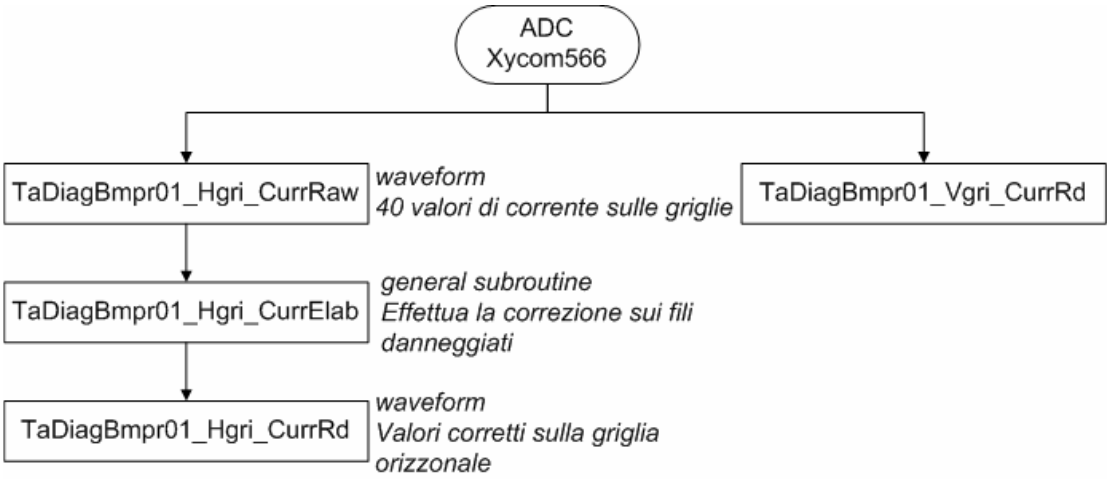

Figura 6.4 Meccanismo di correzione dei valori del profilatore di fascio

Per la movimentazione del profilo invece si usa il record TaDiagBmpr01_Motr01 per il pilotaggio del motore a step. I parametri per il controllo del motore vengono letti/scritti sui vari campi dello stesso record; ad esempio, il campo VELO indica la velocità in termini di step al secondo, TWF impartisce un movimento relativo in avanti del numero di step indicati dal campo TWV, e così via.

I record per il funzionamento del misuratore di emittanza sono invece generati dal file emittancemeter.substitutions, e sono analoghi a quelli già visti in precedenza.
Per la misura utilizziamo in totale 6 canali dell'ADC: due per la griglia orizzontale, due per la griglia verticale, e due per i due encoder di posizione.
Si hanno quindi 6 record per il gain sul canale dell'ADC (TaDiagEmit01_Hgri01_GainSt, TaDiagEmit01_Hgri02_GainSt, TaDiagEmit01_Vgri01_GainSt, …) che, analogamente alla strumentazione già descritta, avranno sempre valore 1.
Le due griglie del misuratore sono da 80 fili ciascuna, e vengono quindi acquisite utilizzando quattro record di tipo waveform, cioè vettori da 40 segnali ciascuno. Analogamente al profilatore di fascio questi record

vengono generati dal file xy566wave.db e, nel caso della misura orizzontale, sono chiamati TaDiagEmit01_Hgri01_CurrRd e TaDiagEmit01_Hgri02_CurrRd.
In modo analogo alla coppa di faraday, la tensione letta dagli encoder viene memorizzata in un record di tipo input analogico formato da un singolo valore, chiamato TaDiagEmit01_HencRd nel caso orizzontale.
La movimentazione degli assi, come nel beam profiler, è assegnata ai record di tipo stepping motor generati da devInfnMtr.db, in cui il nome delle variabili è TaDiagEmit01_Motr01 e TaDiagEmit01_Motr01.
Ma oltre ai record per il funzionamento del misuratore e per l'acquisizione dei dati, si è reso necessario anche l'utilizzo di diversi parametri per l'interfaccia utente; questi servono per fornire al programma che comanda l'acquisizione i parametri con cui deve effettuare la misura, per passare al programma che effettua la post-elaborazione alcuni parametri di calcolo, e per la visualizzazione dei dati elaborati (sia in formato numerico che grafico).
A tal fine, si è deciso di raggruppare tutti i parametri richiesti dai due programmi nel file emittance_interface_in.db, e tutti i dati elaborati da mettere in output in emittance_interface_out.db.

6.3 Script di avvio

Lo startup script che permette l'esecuzione dell'IOC EPICS sul sistema VME (riportato in appendice D) viene creato dal makeBaseApp ma deve essere successivamente modificato per consentire il funzionamento di tutte le periferiche per la strumentazione diagnostica e per il loro software.
Esso consiste in una serie di comandi che vengono automaticamente eseguiti sulla riga di comando di VxWorks subito dopo il suo avvio.
Sono descritti qui di seguito alcuni di questi comandi, che vanno a svolgere le mansioni salienti.
Innanzitutto c'è la riga di codice

```
ifconfig "mottsec0 10.5.0.23 netmask 255.255.255.0"
```

che imposta un opportuno indirizzo alla scheda di rete sul pannello frontale della MVME3100. L'indirizzo IP era già impostato dai parametri di boot, ma questo comando serve per accertarsi che anche la maschera di rete sia quella corretta.
Un altro comando rilevante è

```
putenv "EPICS_TS_NTP_INET=10.5.0.254"
```

Esso crea una variabile d'ambiente EPICS_TS_NTP_INET contenente l'indirizzo IP del server NTP di rete. Esso consente all'IOC EPICS di

sincronizzare il proprio timestamp (e quindi anche quello delle proprie variabili) con quello del server. NTP (Network Time Protocol) è un protocollo usato per sincronizzare l'orologio di diversi computer all'interno di una rete, ed è un componente fondamentale per l'analisi dei dati in un sistema sperimentale come SPES. [28]
Altri comandi di fondamentale importanza al nostro scopo sono quelli che inizializzano le schede VME.
Nell'esempio delle schede XVME-220 e XVME-240, basta richiamare l'apposita funzione passando come parametri un numero identificativo (ID) per la scheda (3 e 4), e l'indirizzo dello ShortIO in esadecimale in cui esse vengono mappate. Tale indirizzo è scritto fisicamente sul circuito della scheda, oppure può essere impostato manualmente spostando degli interruttori.

```
xycom240setup(3,0xe000)
xycom220setup(4,0x3000)
```

La scheda per il controllo dei motori passo-passo invece, oltre ai due parametri precedenti, richiede anche la frequenza di aggiornamento dello stato degli assi motore controllati. Nell'esempio, 10Hz.

```
infnModConfig(1,0x5000,10)
```

Un po' più complicata risulta invece l'inizializzazione della Xycom XVME-566. Essa infatti richiede, oltre all'ID ed all'indirizzo shortI/O, l'indirizzo base (a 24 bit) del buffer per l'acquisizione dei dati (anch'esso impostato fisicamente sulla scheda con degli switch), il livello di priorità degli interrupt generati dalla scheda (da 1 a 7), l'indirizzo del vettore con le richieste di interrupt, ed un valore binario che indica se l'ADC deve considerare i propri ingressi come 32 canali unipolario come 16 canali differenziali.

```
xycom566setup(1, 0xf800, 0xa00000, 3, 0xf2, 1)
```

Essa richiede anche l'inizializzazione e la configurazione del proprio clock interno, che viene fatto con il comando

```
stc566seqmulti(1, 2, 0x100, 0x4800)
```

in cui viene indicato l'ID della scheda da impostare (1), il divisore (2) sul clock a 4MHz di riferimento, il periodo del clock di campionamento (0x100 = 256) ed il periodo della sequenza di campionamento (0x4800 = 18432).

Avremo quindi un periodo di campionamento di 0.25µs · 2 · 256 = 128µs ed un periodo di sequenza di 0.25µs · 2 · 18432 = 9,216ms.
Le sequenze di acquisizione, intese come numero di campioni da acquisire da ciascun canale e il relativo ordine di acquisizione, vengono anch'esse programmate nello script di startup. Una volta che queste vengono inserite nell'ADC, esso provvederà autonomamente ad acquisire i valori e ad informare il processore, tramite interrupt, che i valori sono resi disponibili.

```
seq566set(1, 0, 40, 1, 1, 40)
seq566set(1, 1, 40, 2, 1, 40)
seq566set(1, 2, 1, 3, 1, 1)
seq566set(1, 3, 40, 4, 1, 40)
seq566set(1, 4, 40, 5, 1, 40)
seq566set(1, 5, 1, 6, 1, 1)
seq566set(1, 6, 40, 7, 1, 40)
seq566set(1, 7, 40, 8, 1, 40)
seq566set(1, 8, 1, 9, 1, 1)
```

In questo caso, i valori indicano l'ID della scheda (1), il canale da cui acquisire (0-31 se unipolare, 0-15 se bipolare), il numero di campioni, l'ordine di campionamento (1-31), un valore di priorità (per eliminare ambiguità nel caso di canali con lo stesso numero di ordine), ed un valore di stop facoltativo. Nel caso il sesto valore N, cioè quello di stop, sia presente e sia maggiore di zero, l'acquisizione farà una pausa dopo il campione N-esimo.
Per terminare la configurazione degli ADC, si esegue il comando seguente. [27]

```
xycom566finish()
```

Il database viene caricato con i comandi dbLoadTemplate e dbLoadRecords; il primo si utilizza per caricare file di tipo template o substitutions, il secondo invece si utilizza per caricare i record indicati nel file .db che viene passato come parametro.

```
dbLoadTemplate("db/faradaycup.substitutions")
dbLoadRecords("db/emittance_interface_in.db")
```

Nell'esempio qui sopra viene caricato il file faradaycup.substitutions, che autonomamente andrà a richiamare tutti i file .db necessari e provvederà alla creazione di tutti i record in essi indicati. Il secondo invece si utilizza per creare tutti i record descritti all'interno del file emittance_interface_in.db.

Il principale vantaggio nell'utilizzare i file .substitutions è che si rende la configurazione del database modulare ed il codice diventa agevolmente riutilizzabile. Ad esempio, se avessi la necessità di aggiungere una nuova faraday cup alla linea di trasporto, mi basterebbe aggiungere poche righe all'interno del file substitutions; nel caso avessi utilizzato solo file .db invece avrei bisogno di creare diversi nuovi file ed aggiungerli al sistema.

L'ultimo comando presente nello startup script è *iocInit*, che provvede ad avviare l'esecuzione dell'IOCcore; in questo modo si inizializza il database e si dà inizio al processamento dei record.

6.4 Programmi per la misura di emittanza

Il software per la misura di emittanza è strutturato in due parti: una dedicata all'acquisizione dei dati di misura, ed una dedicata al calcolo dell'emittanza e dei parametri collegati.
Entrambi i programmi sono stati scritti in C e girano come applicativi client su un pc linux interno alla rete dei controlli.
I programmi accedono alle variabili EPICS utilizzando il channel access, secondo la seguente procedura:

- Inizializzazione del protocollo Channel Access (solo ad inizio del programma)
  `ca_context_create(ca_disable_preemptive_callback);`
- Ricerca della variabile di processo
  `ca_create_channel(indata01,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);`
- Effettiva esecuzione del comando precedente
  `ca_pend_io(timeout);`
- Lettura e/o scrittura dei valori delle variabili
  `ca_get(DBR_SHORT,pCh1,&plane);`
  `ca_put(DBR_STRING,pCh1,&status);`
  `ca_array_get(DBR_SHORT,40,pCh3,&datagrid1);`
- Esecuzione dei precedenti comandi (bufferizzati)
  `ca_pend_io(timeout);`
- Libera il canale
  `ca_clear_channel(pCh1);`
- Chiude il Channel Access (solo alla fine del programma)
  `ca_context_destroy();`

L'unico requisito per l'utilizzo di queste funzioni da parte del programma scritto in C è l'inclusione della seguente riga di codice: [8, 9]

```
#include <cadef.h>
```

Il codice integrale dei due programmi è riportato in appendice E.

6.4.1 Acquisizione

Il programma di acquisizione, dopo aver inizializzato il Channel Access, legge dalle variabili EPICS (impostate tramite interfaccia) i parametri fondamentali per effettuare l'acquisizione e per il calcolo; essi sono il piano (X o Y), il passo di acquisizione (ΔX o ΔY), le posizioni iniziali e finali, e il numero di massa dell'isotopo.
A titolo di esempio, l'acquisizione della variabile del piano viene acquisita attraverso il CA con il seguente codice:

```
chid pCh1;
char indata01[] = "TaDiagEmit01_PlanSt";
stat=ca_create_channel(indata01, NULL, NULL,
CA_PRIORITY_DEFAULT, &pCh1);
stat=ca_pend_io(timeout);
stat=ca_get(DBR_SHORT,pCh1,&plane);
stat=ca_pend_io(timeout);
stat=ca_clear_channel(pCh1);
```

Sui parametri acquisiti viene poi effettuato un controllo, per accertarsi che essi siano coerenti. Nel caso questi non abbiano valori appropriati, il programma scrive su una variabile di tipo stringa (visualizzata da interfaccia) l'errore rilevato, chiude il channel access, e termina il programma.
Di seguito, l'esempio del controllo sulla variabile che indica il piano da cui acquisire.

```
chid pCh1;
char indata01[] = "TaDiagEmit01_PlanSt";
stat=ca_create_channel(indata01, NULL, NULL,
CA_PRIORITY_DEFAULT, &pCh1);
stat=ca_pend_io(timeout);
stat=ca_get(DBR_SHORT,pCh1,&plane);
stat=ca_pend_io(timeout);
stat=ca_clear_channel(pCh1);

if ((plane!=0)&&(plane!=1))
    {
      strcpy(status,"Plane Error.");
      stat=ca_create_channel(status1Set, NULL, NULL,
CA_PRIORITY_DEFAULT, &pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
      return(0);
    }
```

Un ulteriore controllo viene effettuato sulla posizione dei motori, per accertarsi che entrambi siano fermi ed in posizione completamente esterna alla traiettoria del fascio. Questo viene effettuato accedendo ai campi LLS (finecorsa esterno) e MOVN (moving) dei due motori.

Fatto ciò, si procede con la creazione del file rawdata, contenente i dati acquisiti. Si è stabilito che il percorso sul file system dove questo file viene creato può essere reso univoco utilizzando parametri come la data e l'ora di inizio acquisizione. Considerando che una acquisizione completa non dura meno di un minuto, ciò è stato ritenuto sufficiente.
Per accertarsi che la data e l'ora corrispondano con quelli esatti dell'intero sistema essi vengono acquisiti da una variabile; in questo modo, essi corrispondono con quelli del server NTP.
I dati di anno, mese, giorno, ora e minuto vengono quindi estratti sotto forma di stringa e convertiti in formato numerico:

```
nBytes = dbr_size_n ( DBR_TIME_DOUBLE, 1 );
pTD = ( struct dbr_time_double * ) malloc ( nBytes );
stat = ca_array_get ( DBR_TIME_DOUBLE, 1, pCh1, pTD );
stat = ca_pend_io(timeout);
epicsTimeToStrftime ( tempstring, sizeof ( tempstring ),
"%Y", & pTD->stamp );
year=atoi(tempstring);
```

In modo analogo essi vengono concatenati in una stringa per andare a formare il percorso dove salvare il file. Il percorso viene creato a partire dalla directory dove si trova il programma, e creando (qualora non vi fossero già) le sottodirectory
Data/<YYYYMMDD>/<HHmm>
In questo modo, tutte le misure sono raggruppate per giorno, e in ordine di esecuzione.

```
char folder1[] = "Data";
strcpy(path,folder1);
epicsTimeToStrftime ( tempstring, sizeof ( tempstring ),
"/%Y%m%d", & pTD->stamp );
strcat(path,tempstring);
epicsTimeToStrftime ( tempstring, sizeof ( tempstring ),
"/%H%M", & pTD->stamp );
strcat(path,tempstring);
```

Il file di dati creato con l'acquisizione è chiamato rawdata e si trova all'interno della cartella. Esso viene creato con i seguenti comandi:

```
strcpy(comando,path);
strcat(comando,"/");
strcat(comando,file1);
FILE *outputfile = NULL;
outputfile = fopen(comando,"w");
```

A questo punto inizia la scrittura del file, che viene effettuata con il comando fprintf. Nell'esempio qui sotto le prime due righe, una per l'intestazione, e la seconda con data e ora.

```
fprintf(outputfile,"EMITTANCE MEASUREMENT\n");
epicsTimeToStrftime ( tempstring, sizeof ( tempstring ),
"%Y %m %d - %H %M", & pTD->stamp );
fprintf(outputfile,"%s\n",tempstring);
```

A seguire, vengono inserite le informazioni facoltative per la misura (titolo, tipo di sorgente, isotopo), le informazioni sulla misura (delta, piano, ecc...) e una serie di informazioni sullo stato della macchina (ad esempio lo stato del vuoto, il valore di alta tensione, la posizione dell'estrattore, le tensioni sulla

linea di trasporto, ecc...) che vengono autonomamente lette dal programma sulle variabili di processo del sistema.
A questo punto viene impostato il motore ed inizia l'acquisizione dei dati.
Bisogna però prima effettuare le opportune conversioni dei parametri di misura da millimetri a step; il passo di acquisizione viene quindi convertito in step motore, la posizione iniziale viene convertita sia in step motore che in step dell'encoder, mentre quella finale solo in step dell'encoder.
La posizione iniziale viene raggiunta con un movimento relativo del motore pari al numero di step previsti per raggiungerla; ogni decimo di secondo il programma controlla se il motore sia ancora in movimento (cioè se il campo MOVN del record sia impostato ad 1).
Al raggiungimento della posizione iniziale, inizia l'acquisizione.
Per ogni campione acquisito, vado a scrivere sul file la posizione della slitta (indicata con il numero di step dell'encoder) e, sulla riga successiva, gli 80 valori letti dall'ADC sui fili della griglia.
Il passaggio da una posizione alla successiva viene effettuato impartendo al motore il numero di passi corrispondenti al delta impostato ed attendendo che il motore sia nuovamente fermo.
L'acquisizione termina con 3 condizioni:

- Il valore dell'encoder è superiore del numero di step corrispondenti alla fine dell'acquisizione (cioè, la posizione finale impostata è stata raggiunta);
- Il numero di acquisizioni effettuate ha raggiunto il valore massimo (attualmente 500);
- La slitta ha raggiunto il finecorsa interno.

La porzione del file con i dati dell'acquisizione è delimitata da due tag: "RAW DATA:" e "END DATA".
L'ultima riga del file è invece dedicata al numero di campionamenti effettuati, ed è utile ad agevolare il programma di calcolo.
Terminato il file raw, la slitta viene riportata in posizione esterna alla traiettoria del fascio, impartendo al motore un movimento relativo verso l'esterno di un numero di passi sufficientemente grande.
Le informazioni numeriche indicanti data e ora vengono infine scritte su delle variabili di processo utilizzate come input per il programma di calcolo dell'emittanza. In questo modo, non appena l'acquisizione è terminata, sarà possibile iniziare il calcolo con un semplice click del mouse, senza dover inserire alcun parametro.
L'utente viene informato che l'acquisizione è andata a buon fine dalla presenza della stringa "OK" nella variabile di stato visualizzata sull'interfaccia; una volta scritta questa stringa sulla apposita variabile di processo, il programma chiude il channel access e termina. [9, 14, 16, 29, 30, 31]

6.4.2 Calcolo

Il programma di calcolo, analogamente a quello di acquisizione, inizializza il channel access e lo utilizza per accedere alle variabili necessarie al proprio funzionamento. In questo caso, le variabili in oggetto sono la data e l'ora di inizio acquisizione (necessari per ricostruire il percorso dove si trova il file

con i dati raw), ed il valore di soglia (threshold) da considerare per distinguere il rumore dal fascio ionico.
Anche in questo caso viene effettuato un controllo tra i parametri per accertarsi che abbiano un valore appropriato e, in caso negativo, viene segnalato un errore all'utente attraverso una variabile di tipo stringa visualizzata su interfaccia.

```
if ((year<1900)||(year>2500) || (month>12)||(month<1) ||
(day>31)||(day<1) || (hour>23)||(hour<0) ||
(minute>59)||(minute<0) ||
(threshold>4095)||(threshold<0))
    {
      strcpy(status,"Parameter Error.");
      stat=ca_create_channel(status2Set, NULL, NULL,
CA_PRIORITY_DEFAULT, &pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
      return(0);
    }
```

I valori di data e ora vengono a questo punto utilizzati per ricostruire il path dove dovrebbe trovarsi il file richiesto. La ricostruzione viene effettuata componendo una stringa carattere per carattere, in modo da ricostruire il formato <YYYYMMDD>/<HHmm>

L'accesso al file viene infine eseguito in sola lettura. Qualora il file non sia disponibile al percorso indicato, viene segnalato un errore all'utente attraverso la solita stringa di status ed il programma termina.

```
FILE *textfile = NULL;
textfile = fopen(comando,"r");
```

A questo punto inizia un primo controllo del file in cui il file viene interamente scansionato; vengono effettuati controlli per accertarsi che vi siano tutti i tag richiesti, e che la struttura del file corrisponda a quella richiesta. Ad esempio, nel caso dei campi facoltativi (come ad esempio il titolo), ci si accerta solamente che la prima parola della riga sia quella che ci si aspetta (ad esempio “TITLE:”); in altri casi in cui il valore richiesto è un parametro, si va a controllare che il valore presente sia numerico utilizzando la funzione isdigit(i) sul primo carattere.

Nel caso il file in ingresso sia diverso da quello previsto oppure termini prima del previsto, l'errore viene segnalato all'utente assieme al numero della riga del file contenente l'errore e l'esecuzione termina.
Giunti all'ultima riga, si effettua la lettura del numero di campionamenti effettuati (cioè il numero di volte in cui ho effettuato la lettura contemporanea dei valori del trasduttore di posizione e del valore di corrente sulla griglia); qualora questo non sia indicato o non corrisponda al numero di campioni presenti nell'area dati del file, ancora una volta viene segnalato un errore.
A questo punto, il file viene chiuso e riaperto, e si procede alla lettura di tutti i valori utili per il calcolo dell'emittanza e dei parametri collegati.
Tutti i valori di posizione dell'encoder vengono inseriti in un vettore di interi di dimensione pari al numero di campionamenti; i valori delle griglie invece vengono inseriti in un vettore bidimensionale chiamato rawdata di larghezza 80 (il numero di fili) e lunghezza pari sempre al numero di campionamenti.

Tutte queste operazioni di lettura vengono effettuate principalmente con l'utilizzo di funzioni standard presenti nel linguaggio C, e con l'aiuto di alcune funzioni che, per comodità, sono state inserite in un file readfilefunctions.c esterno. Tra queste vi sono:
la funzione itoa, che converte un valore numerico intero in una stringa:

```
void itoa(int n, int len, char *buf)
{
  int i;
  char sign = ' ';
  if (0 > n) {
    n = -n;
    sign = '-';
  }
  for (i = len; i > 0; i--) {
    if (n > 0 || len == i) {
      buf[i-1] = '0' + (n % 10);
      n = n / 10;
    } else {
      buf[i-1] = sign;
      sign = ' ';
    }
  }
  if (strlen(buf)>len)
    buf[len]='\0';
}
```

La funzione launchlineerror invece indica all'utente quale sia il numero di riga del file rawdata in cui si è verificato l'errore:

```
void launchlineerror(int i)
{
      strcpy(status,"Error reading file, line:");
      itoa(i,5,tempstring);
      strcat(status,tempstring);
      stat=ca_create_channel(status2Set, NULL, NULL,
CA_PRIORITY_DEFAULT, &pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
}
```

La funzione nextline invece scorre il file in lettura fino a portarsi all'inizio di una nuova riga:

```
void nextline(FILE *rdf)
{
  int c;
  do
  {c=fgetc(rdf);}
  while (c != '\n');
}
```

La funzione readline infine scorre il file in lettura e salva il contenuto della riga corrente in una stringa, fino a portarsi all'inizio di una nuova riga:

```
void readline(FILE *rdf, char *buf)
{
  int c;
  int i=0;
  c=fgetc(rdf);
  while (c != '\n')
    {buf[i]=c;
      i++;
      c=fgetc(rdf);
    }
  buf[i]='\0';
}
```

A questo punto il programma lascia spazio ad eventuali correzioni da effettuare sui dati, qualora vi siano degli errori sui fili della griglia dovuti a

problemi di tipo meccanico (fili rotti o in contatto fra loro) o di elettronica, ed infine porta a zero tutti i valori minori di quelli di soglia.

```
for (i=0;i<iterations;i++)
  for (j=0;j<wiresnumber;j++)
    if (rawdata[i][j]<threshold)
      rawdata[i][j]=0;
```

Una volta raccolti tutti i dati il programma inizia la fase di calcolo.
Vi sono innanzitutto dei calcoli preliminari, che servono per stabilire l'angolo in milliradianti di ogni filo rispetto alla fessura, e la posizione in mm della griglia ad ogni campionamento (a partire dal valore del trasduttore indicato in step dell'ADC).
Viene poi effettuata una scansione di tutta la matrice con i dati di corrente sulle griglie, si effettua la ricerca del valore massimo, e vengono fatte alcune integrazioni sui valori letti.

```
maxdata=0;
for (i=0; i<iterations; i++)
{
  for (j=0; j<wiresnumber; j++)
  {
   if (rawdata[i][j]>maxdata) /*searches the max value*/
   maxdata=rawdata[i][j];

   px[i]=px[i]+rawdata[i][j];
   pxp[j]=pxp[j]+rawdata[i][j];
   sumw=sumw+rawdata[i][j];
   xave=xave+rawdata[i][j]*positionsmm[i];
   xpave=xpave+rawdata[i][j]*wiresmrad[j];
  sx2ave= sx2ave+
rawdata[i][j]*positionsmm[i]*positionsmm[i];
   sxp2ave= sxp2ave+
rawdata[i][j]*wiresmrad[j]*wiresmrad[j];
   sxxpave= sxxpave+
rawdata[i][j]*positionsmm[i]*wiresmrad[j];
   }
 }
xave = xave / sumw;
xpave=xpave / sumw;
sx2ave=sx2ave / sumw;
sxp2ave=sxp2ave / sumw;
sxxpave=sxxpave / sumw;
```

Come si può intuire dal codice qui sopra, il vettore px corrisponde all'integrazione dei valori di corrente per ogni posizione di acquisizione e pxp

è l'integrazione delle correnti su ogni filo, mentre sumw è l'integrazione di corrente totale.
Xave, xpave, sx2ave, sxp2ave ed sxxpave corrispondono quindi rispettivamente alla media della posizione delle particelle del fascio (quindi, al suo centro), alla media del momento angolare, alla media del quadrato della posizione, alla media del quadrato del momento, ed alla media del prodotto posizione·momento.
Tutti questi valori vengono utilizzati per il calcolo della matrice di fascio, composta dai quadrati delle deviazioni standard del profilo di distribuzione (x2ave) e del profilo angolare (xp2ave) , e dalla correlazione fra posizione e momento (xxpave).

```
x2ave=sx2ave - (xave * xave);
xp2ave=sxp2ave - (xpave * xpave);
xxpave=sxxpave - (xave * xpave);
```

A questo punto, come già descritto nel capitolo 4, l'emittanza può essere definita come la radice del determinante della matrice simmetrica di fascio σ, composta dai 3 valori appena descritti.
Essa viene quindi calcolata nel seguente modo:

```
emitrms = sqrt(x2ave * xp2ave - xxpave * xxpave);
```

A seguire, si procede alla determinazione delle larghezze del profilo e della distribuzione angolare, dei parametri di Twiss, del valore massimo di emittanza delle particelle, e di altri parametri meno rilevanti.
Un valore particolarmente sensibile è invece quello del calcolo dell'emittanza normalizzata, calcolata moltiplicando il precedente valore di emittanza per un fattore Beta relativistico. Questo fattore è calcolato con la formula:
Brel= radq (1- 1/(1 + v/(M · 931494))^2),
corrispondente a $\gamma_{rel} \cdot v_s/c$, dove v è l'energia delle particelle in Kev, M è il numero di massa, e 931494 è la massa del nucleone in KeV/c^2.

Terminati i calcoli, il programma procede all'output dei dati numerici in due diversi modi. Il primo consiste nella creazione di un file di testo contenente tutte le informazioni sullo stato della macchina al momento della misura, i parametri della misura, ed un elenco dettagliato di tutti i valori calcolati. Il secondo consiste nell'andare a scrivere solamente i parametri principali sulle variabili EPICS che vengono visualizzate sull'interfaccia di output.
In questo modo, l'utente può visualizzare i parametri principali immediatamente dopo il calcolo già da interfaccia grafica e, nel caso abbia bisogno di controllare parametri più specifici, può andare a reperirli manualmente sul file di testo.
Questo file viene creato all'interno della stessa cartella contenente il file rawdata, e si chiama emittancedata.

Terminato l'output dei valori numerici, si procede con l'elaborazione dei dati grafici. Tutti i valori acquisiti superiori a quello di soglia vengono elaborati e suddivisi in 4 gruppi, a seconda che il loro valore sia minore di un quarto del massimo, compreso tra un quarto ed un mezzo, compreso tra un mezzo e tre quarti, o sia maggiore di tre quarti del massimo valore rilevato. Le coordinate x (cioè la posizione della fessura al momento dell'acquisizione) ed y (o x', cioè la posizione angolare del filo rispetto alla fessura) vengono salvate in due array distinti. Ovviamente, ad ogni gruppo di valori corrispondono due array, uno per la x ed uno per la y.
Questi array andranno poi scritti su delle variabili EPICS, e i punti (x,y) andranno visualizzati su un grafico, con un colore diverso per ogni gruppo. In questo modo, la visualizzazione dello spazio di fase risulta molto intuitiva e di rapida comprensione.
Ma oltre alla visualizzazione grafica dei punti, si procede anche con quella delle elllissi dell'emittanza.
Le ellissi vengono disegnate prendendo 30 punti da ciascuna di esse (calcolati con un passo angolare costante) e visualizzando sul grafico i punti uniti da una linea. Per una corretta visualizzazione i vettori x e y hanno 31 valori, in cui il primo coincide con l'ultimo, in modo tale da completare la figura. Anche i vettori delle ellissi andranno scritti su una apposita variabile di processo, per essere poi recuperati dall'apposita interfaccia.
Per concludere, il programma chiude il channel access e termina. [9, 14, 29, 30, 31]

6.5 Interfacce utente

Le interfacce utente per l'utilizzo del front end sono interamente realizzate con CSS (Control System Studio).
L'interfaccia per l'utilizzo della coppa di faraday è la seguente.

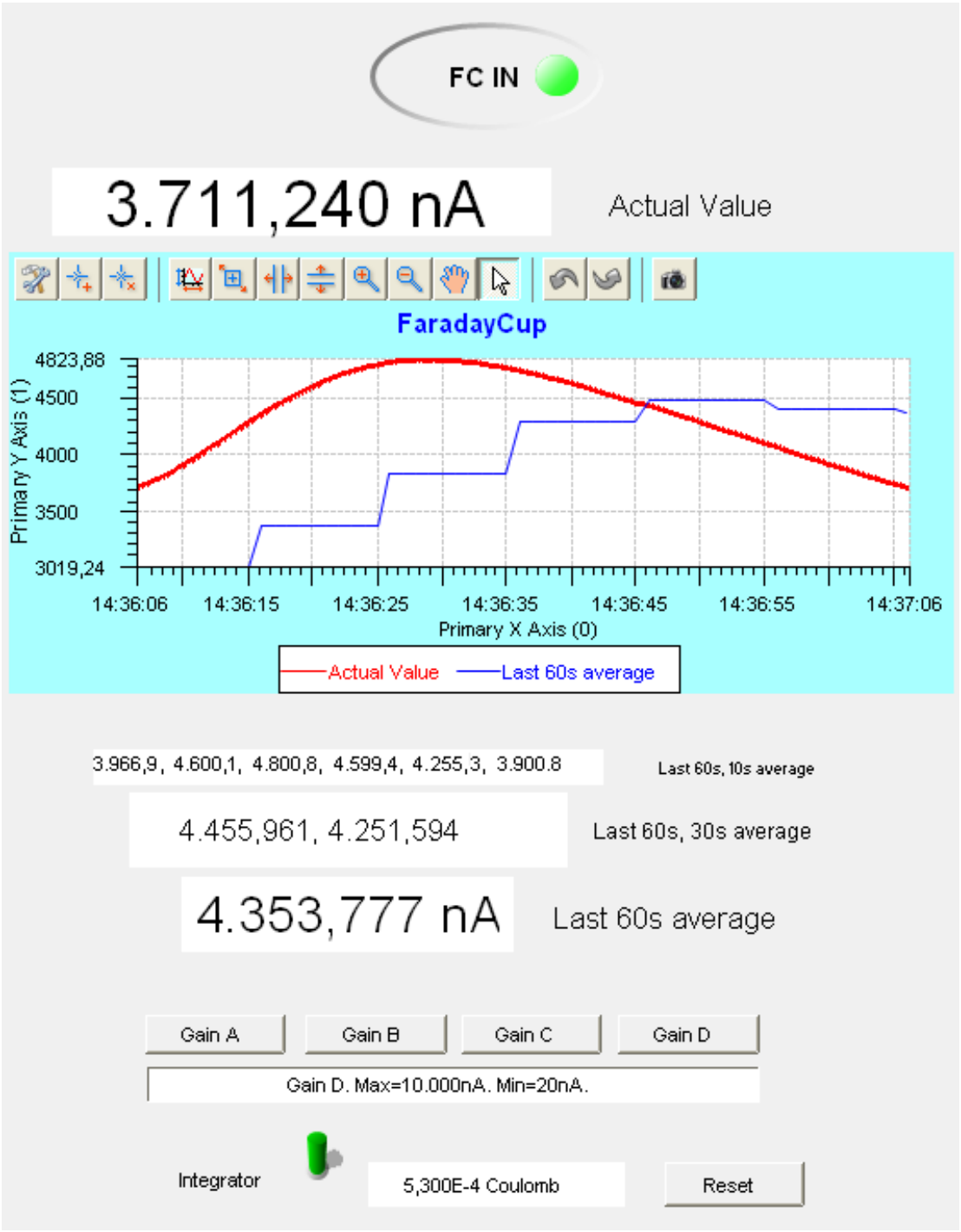

Figura 6.5 Interfaccia per l'utilizzo della coppa di Faraday

Essa è composta da un pulsante binario che, andando a scrivere i valori 0 ed 1 sulla variabile TaDiagFacp01_out0, stabilisce se la coppa debba essere inserita o meno nella traiettoria del fascio.

Figura 6.6 Pulsante per la movimentazione della Faraday Cup

A seguire c'è un display numerico indicante il valore di lettura attuale della corrente, ed un grafico rappresentante l'andamento di corrente nell'ultimo minuto (in rosso) ed il valore della media degli ultimi 60 secondi (aggiornata ogni 10s, in blu).
Per quanto riguarda la visualizzazione della corrente, è stato necessario impostare su CSS determinate proprietà alla traccia da visualizzare. Il Buffer Size è stato impostato a 600 (in modo da avere i campioni dell'ultimo minuto), sono stati spuntati i valori Chronological e Concatenate Data, il plot mode è “Plot last n pts.”, il tipo di traccia è Solid Line, ed il nome della variabile è stato inserito nel campo Y PV. Campi analoghi anche per la visualizzazione della media, con la sola eccezione del Buffer Size (60 invece di 600).
Più in basso, tre display numerici indicano le medie dei 10, dei 30, e dei 60 secondi.
Nella parte inferiore vi sono i pulsanti per il cambio di guadagno e l'integratore.
I quattro pulsanti “Gain A”, “Gain B”, “Gain C”, e “Gain D” vanno rispettivamente a scrivere i valori 1, 2, 3, e 4 sulla variabile TaDiagFacp01_HwGainSt00, avviando così la subroutine del record TaDiagFacp01_HwGainSt01, che cambia il guadagno effettivo. Nel display viene visualizzata la stringa TaDiagFacp01_GainStatus, che indica all'operatore il guadagno attualmente impostato, ed i valori massimi e minimi consigliati per quel tipo di guadagno.
L'interruttore verde dell'integratore invece corrisponde alla variabile binaria TaDiagFacp01:InteSt; se esso è in posizione ON, il record TaDiagFacp01_InteRd (visualizzato nel display a fianco) integrerà il valore di corrente, altrimento rimane fermo. Il pulsante di reset infine, quando azionato, va a scrivere il valore 0 sulla variabile numerica TaDiagFacp01_InteRd, azzerando di fatto l'integratore.

L'interfaccia del profilatore di fascio invece è composta innanzitutto da due pulsanti di azione che vanno ad scrivere il valore 1 sui campi TWR e TWF del record TaDiagBmpr01_Mort01, impartendone la movimentazione in direzioni opposte per l'inserimento ed il disinserimento.

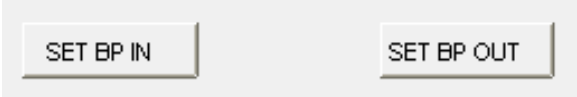

Figura 6.7 Pulsanti per la movimentazione del profilatore

L'utente viene informato della posizione del profilatore da un indicatore di tipo testuale, che rende visibili delle scritte colorate a seconda che i campi LLS (finecorsa esterno), MOVN (moving) e HLS (finecorsa interno) siano impostati ad uno.

Figura 6.8 Indicatore di posizione e di movimento del beam profiler

Il cambio di colore delle scritte viene effettuato eseguendo uno script javascript. La parte fondamentale di questo codice, riportato nell'esempio del colore rosso, è la seguente.

```
var RED = CustomMediaFactory.COLOR_RED;
var DARK_GRAY = CustomMediaFactory.COLOR_DARK_GRAY;
var value = ValueUtil.getDouble (pvArray[0].getValue());
widgetController.getWidgetModel().setForegroundColor
(value > 0? RED : DARK_GRAY);
```

Il bit del record TaDiagBmpr01_GainBit_out6 per il cambio di guadagno dei preamplificatori del beam profiler viene impostato con un pulsante binario, avente il seguente aspetto.

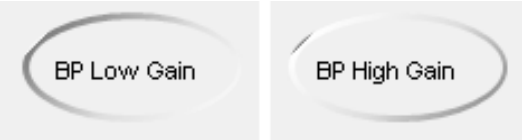

Figura 6.9 Pulsante per il cambio di guadagno del profilatore

I valori di corrente sulle griglie vengono visualizzati su due grafici a barre, aventi sull'asse X il numero del filo, e sull'asse Y il numero di passo dell'ADC corrispondente al valore letto.

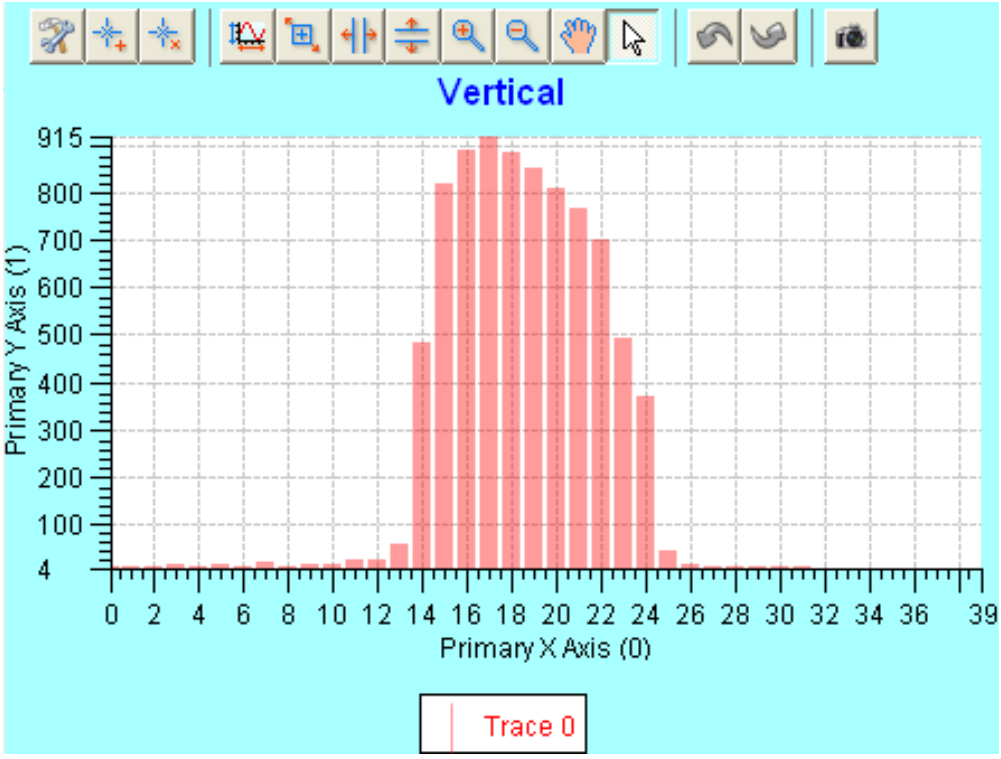

Figura 6.10 Esempio di profilo del fascio visualizzato da interfaccia

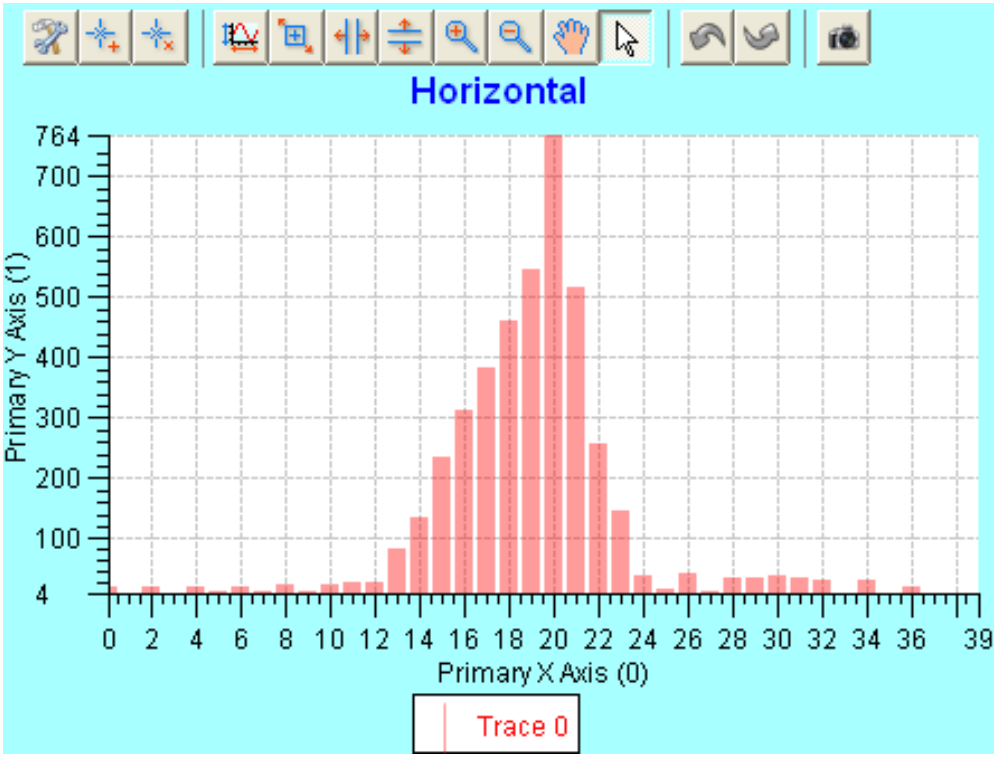

Figura 6.11 Esempio di profilo del fascio visualizzato da interfaccia

Per ottenere questa visualizzazione è stato necessario configurare diverse proprietà del grafico. Sulle proprietà dell'asse X è stato tolto l'autoscale, ed i valori minimo e massimo sono stati impostati rispettivamente a 0 e 39; sull'asse Y invece si è lasciato l'autoscale. Nelle proprietà della traccia si è lasciata la spunta su Chronological, ma la si è tolta da Concatenate Data, il Plot Mode è “plot n pts. & stop”, il Trace Type è Bar, e la variabile waveform è indicata sempre nel campo Y PV.

Per quanto riguarda l'emittanza, una prima interfaccia è quella dedicata alla parte di acquisizione, il cui scopo è quello di raccogliere i parametri da fornire al programma.

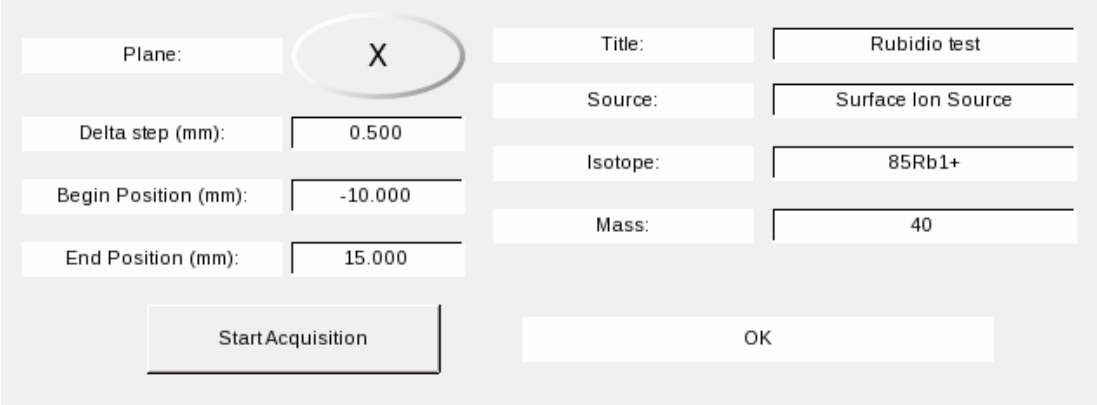

Figura 6.12 Interfaccia per comandare l’acquisizione della misura di emittanza

La seconda parte della stessa interfaccia è invece dedicata alla parte di calcolo, ed anch'essa serve per fornire all'apposito programma i parametri necessari al recupero dei dati acquisiti ed al calcolo.

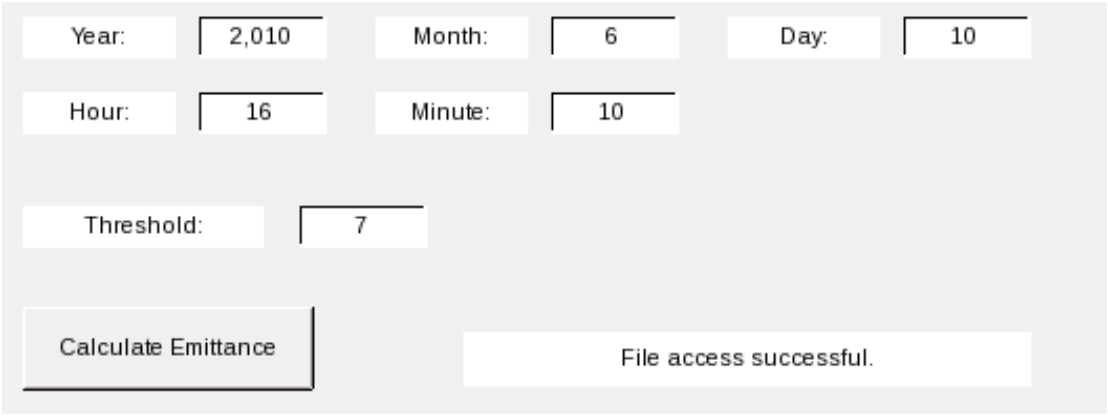

Figura 6.13 Interfaccia per comandare il calcolo dell’emittanza

L'output dei dati numerici viene visualizzato in una seconda interfaccia utente.

Date and Time	2010 06 10 - 16 10
Center Position	0.148 mm
Center Momentum	4.589 mrad
Beam Size	2.638 mm
Beam Divergence	9.287 mrad
Twiss Alpha	1.461
Twiss Beta	0.503 mm/mrad
Twiss Gamma	6.231 mrad/mm
Emittance RMS	13.842 mm*mrad
Beta rel.	0.003405
Emittance N RMS	0.0471 mm*mrad
Max Data	3,047

Figura 6.14 Interfaccia con i dati numerici in output

Una terza interfaccia grafica è invece dedicata alla visualizzazione dei grafici della misura d'emittanza.

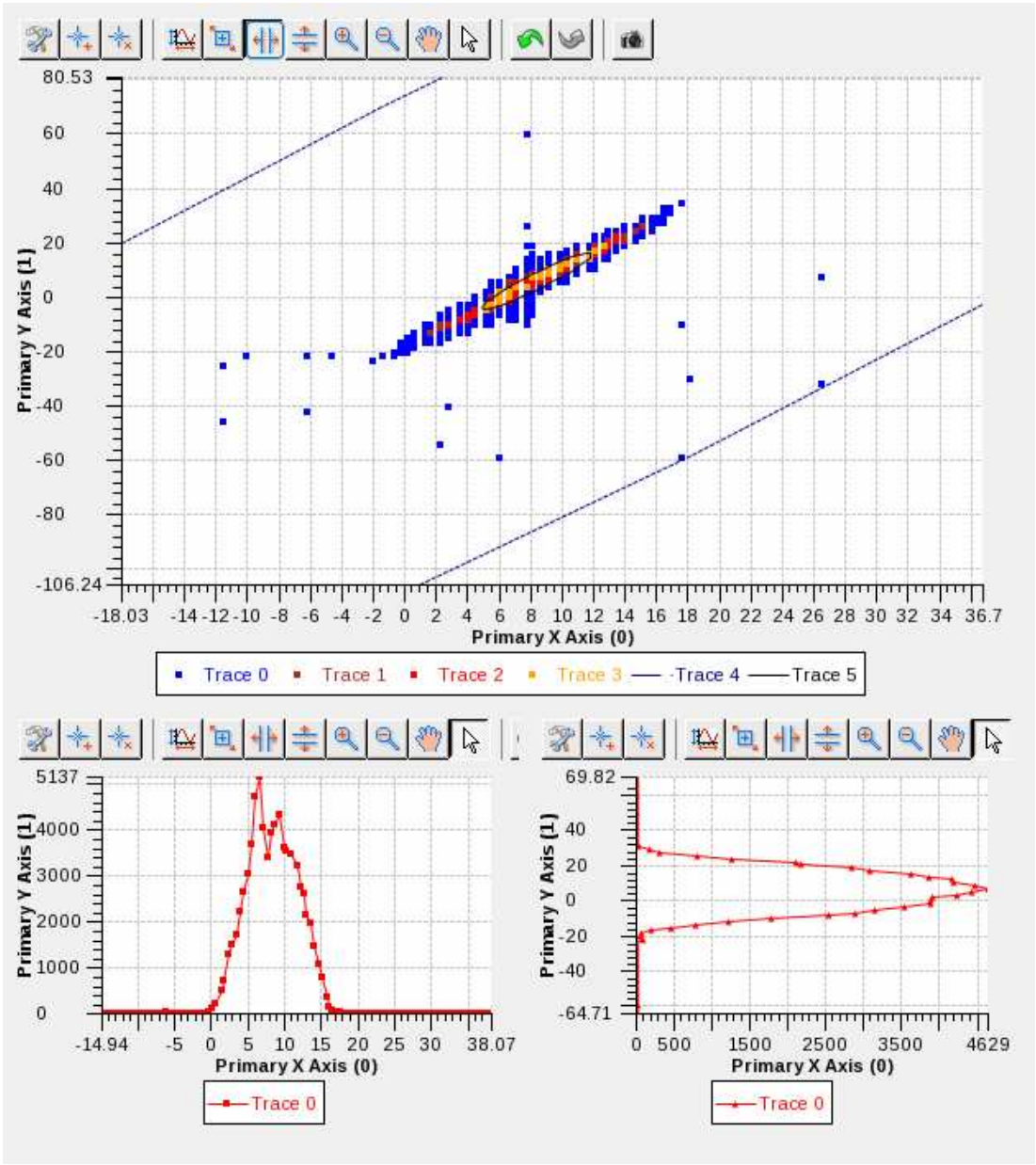

Figura 6.15 Interfaccia con l'output dei dati grafici della misura di emittanza.

Il primo grafico rappresenta lo spazio di fase del fascio, in cui i valori di corrente rilevati sono visualizzati con un colore diverso a seconda della loro intensità. Vengono poi disegnate le due ellissi di emittanza. Sull'asse X si ha la posizione in millimetri, mentre sull'asse Y si ha la posizione relativa in milliradianti del filo rispetto alla fessura.
Per la visualizzazione di questo grafico sono state utilizzate 6 tracce (4 per i punti, due per le ellissi), e quindi 12 variabili (X e Y per ogni traccia). Sulle proprietà di tutte le traccie sono state rimosse le spunte su Concatenate Data e Chronological ed il Plot Mode è "plot n pts."; l'unica differenza è che mentre nelle prime 4 traccie il Trace Type è Point, nelle ultime due (dedicate alle ellissi) esso è Solid Line.
Il due grafici in basso rappresentano invece, rispettivamente, la somma delle correnti lette per ogni posizione di acquisizione, e la somma delle correnti lette su ogni filo della griglia durante l'acquisizione. Anche questi sono

realizzati togliendo le proprietà Chronological e Concatenate Data, ed utilizzando “plot last n pts.”.
Un ulteriore particolare è che il Buffer Size di tutte le traccie di questi tre grafici deve corrispondere con la dimensione dei vettori da visualizzare.
Tutti i grafici sono navigabili con il solo utilizzo del mouse grazie ai pulsanti posti sopra ad ogni grafico. [12]

Conclusioni

La produzione di fasci di particelle radioattive è di importanza cruciale per applicazioni che vanno dallo studio della materia, allo studio dell'universo, fino ad alcune applicazioni mediche.
Il progetto SPES, volto alla produzione di fasci di questo tipo, prevede la costruzione di un impianto di tipo ISOL presso i Laboratori Nazionali di Legnaro dell'Istituto Nazionale di Fisica Nucleare.
Il Front-End su cui si è svolto il lavoro di tesi è una parte fondamentale di questo progetto, in quanto contiene il target di materiale fissile per la produzione degli atomi esotici, il sistema di ionizzazione e di estrazione del fascio, un sistema di trasporto ed un sistema di diagnostica.
Il sistema di diagnostica è una componente fondamentale di un acceleratore, in quanto permette di analizzare la struttura del fascio, di misurare determinate proprietà fisiche, e di effettuare studi di dinamica delle particelle. Ad esempio, permette lo studio della sorgente di ioni, misurandone l'efficienza (che dipende anche dal tipo di ione) o l'emittanza.
Consente inoltre di verificare se i parametri di trasporto sono correttamente impostati, o se c'è qualche problema lungo la linea di trasporto del fascio.
Il presente lavoro di tesi si è svolto quindi sul sistema di rivelazione di ioni del Front-End del progetto SPES, che è composto da una Faraday cup, un beam profiler, ed un emittance meter. La Faraday cup serve a misurare la corrente istantanea di fascio o a fare da integratore per lo studio delle efficienze di trasporto e ionizzazione, il beam profiler per garantire la corretta focalizzazione del fascio, e l'emittance meter per determinare il valore di emittanza della sorgente.

Per la realizzazione del sistema di acquisizione e controllo della diagnostica si è scelto di utilizzare un controller su bus VME, composto da una scheda con processore powerpc motorola e diverse schede di tipo Input/Output binario, ADC, e controller per motori passo-passo.
Come sistema operativo si è scelto di utilizzare VxWorks, un sistema operativo real time leggero ed efficiente, altamente customizzabile, e che effettua il boot da rete.

Queste scelte hardware e software sono basate sulla necessità di avere determinismo nelle tempistiche di acquisizione ed elaborazione, e sul vantaggio di utilizzare degli standard consolidati.
Una prima parte del lavoro svolto è stata quindi l'installazine e la configurazione degli strumenti elettronici per la diagnostica e la configurazione/customizzazione del sistema operativo.

Dal punto di vista applicativo, in conformità con il resto del sistema di controlli, si è optato per il software EPICS in quanto è provato essere veloce e robusto, open source, ed utilizzabile come middleware per integrare hardware di tipo eterogeneo.
Si è quindi proceduto alla creazione di un Input/Output Controller EPICS per VxWorks con i driver per utilizzare tutte le schede hardware presenti nel crate, ed alla configurazione del database EPICS con tutti i record ed i link necessari. Per determinate funzioni, come i cambi di guadagno della coppa di Faraday o le correzioni sui dati *raw* del profilatore, si è reso necessario scrivere delle subroutine in C, la cui esecuzione è collegata al processameto dei record.
Per la messa in funzione e la corretta inizializzazione di tutto l'hardware e software, si è configurato uno script di avvio (startup command) che viene automaticamente eseguito nella riga di comando di VxWorks all'avvio del sistema.

Mentre le misure di corrente e del profilo possono essere visualizzate su interfaccia grafica senza il bisogno di pesanti elaborazioni intermedie, la misura di emittanza richiede calcoli più complessi.
Si è allora proceduto con la creazione di due applicativi client, scritti in linguaggio C, che accedono alle variabili EPICS tramite il protocollo Channel Access. Il primo gestisce l'acquisizione dei dati e li salva su un file, il secondo invece consente di leggere il file e di elaborare i dati in esso contenuti, scrivendo i risultati sia su file che su display.
Per concludere, parallelamente a tutto il lavoro di programmazione/configurazione, sono state create le interfacce operatore per il sistema.

Attualmente, l'intero sistema di diagnostica e le relative interfacce sono già in fase di utilizzo per i test di ionizzazione sulle sorgenti, con ottimi risultati.

Appendice A – Caratteristiche della scheda MVME3100

Sommario delle caratteristiche:

Feature	**Description**
Processor/Host Controller/Memory Controller	– Single 667/833 MHz MPC8540 PowerQUICC IIITM integrated processor (e500 core) – Integrated 256KB L2 cache/SRAM – Integrated four-channel DMA controller – Integrated PCI/PCI-X controller – Two integrated 10/100/1000 Ethernet controllers – Integrated 10/100 Ethernet controller – Integrated dual UART – Integrated I2C controller – Integrated programmable interrupt controller – Integrated local bus controller – Integrated DDR SDRAM controller
System Memory	– One SODIMM socket – Up to DDR333, ECC – 256MB or 512MB SODIMM
I2C Interface	– One 8KB VPD serial EEPROM – Two 64KB user configuration serial EEPROMs – One real-time clock (RTC) with removable battery – One temperature sensor – Interface to SPD(s) on SODIMM and P2 for RTM VPD
Flash	– 32MB to 256MB soldered Flash with two alternate 1MB boot sectors selectable via a hardware switch – Hardware switch or software bit write protection for entire logical bank

Feature	Description
PCI Interface	Bus A: – 66 MHz PCI-X (PCI-X 1.0b compliant) – One TSi148 VMEbus controller – One serial ATA (sATA) controller – Two PCI6520 PCI-X-to-PCI-X bridges (primary side) Bus B: – 33/66/100 MHz PCI/PCI-X (PCI 2.2 and PCI-X 1.0b compliant) – Two +3.3V/5V selectable VIO, 64-bit, single-wide PMC sites or one double-wide PMC site (PrPMC ANSI/VITA 32-2003 and PCI-X Auxiliary ANSI/VITA 39-2003 compliant) – One PCI6520 PCI-X-to-PCI-X bridge (secondary side) Bus C (-1263 version): – 33 MHz PCI (PCI 2.2 compliant) – One USB 2.0 controller – One PCI expansion connector for interface to PMCspan – One PCI6520 PCI-X-to-PCI-X bridge (secondary side)
I/O	– One front panel RJ45 connector with integrated LEDs for front I/O: one serial channel – One front panel RJ45 connector with integrated LEDs for front I/O: one 10/100/1000 Ethernet channel – One front panel external sATA data connector for front I/O: one sATA channel – One front panel USB Type A upright receptacle for front I/O: one USB 2.0 channel (-1263 version) – PMC site 1 front I/O and rear P2 I/O – PMC site 2 front I/O
Serial ATA	– One four-channel sATA controller: one channel for front-panel I/O, one channel for planar I/O, one channel for future rear P0 I/O, and one channel is not used – One planar data connector and one planar power connector for an interface to the sATA hard disk drive
USB (-1263 version)	– One four-channel USB 2.0 controller: one channel for front panel I/O and one channel for future rear P0 I/O. The other two channels are not used.
Ethernet	– Two 10/100/1000 MPC8540 Ethernet channels for front-panel I/O and rear P2 I/O – One 10/100 MPC8540 Ethernet channel for rear P2 I/O
Serial Interface	– One 16550-compatible, 9.6 to 115.2 KBAUD, MPC8540, asynchronous serial channel for front-panel I/O – One quad UART controller to provide four 16550-compatible, 9.6 to 115.2 KBAUD, asynchronous serial channels for rear P2 I/O

Feature	Description
Timers	– Four 32-bit MPC8540 timers – Four 32-bit timers in a PLD
Watchdog Timer	– One MPC8540 watchdog timer
VME Interface	– VME64 (ANSI/VITA 1-1994) compliant – VME64 Extensions (ANSI/VITA 1.1-1997) compliant – 2eSST (ANSI/VITA 1.5-2003) compliant – VITA 41.0, version 0.9 compliant – Two five-row P1 and P2 backplane connectors – One TSi148 VMEbus controller
Form Factor	– Standard 6U VME
Miscellaneous	– One front-panel reset/abort switch – Four front-panel status indicators: 10/100/1000 Ethernet link/speed and activity, board fail, and user software controlled LED – Six planar status indicators: one power supply status LED, two user software controlled LEDs, three sATA activity LEDs (one per channel) – One standard 16-pin JTAG/COP header – Boundary scan support – Switches for VME geographical addressing in a three-row backplane
Software Support	– VxWorks operating system – Linux operating system

Appendice B – Configurazione dell'IOC

Configurazione dell'applicazione

File app/configure/RELEASE

```
TEMPLATE_TOP=$(EPICS_BASE)/templates/makeBaseApp/top

EPICS_BASE=/home/confort/base-3.14.10
GENSUB=/home/confort/workspace/nicola/prova10/genSub
XYCOMIOC=/home/confort/workspace/nicola/prova10/xycomioc
MOTOR=/home/confort/workspace/nicola/prova10/infn-motor
```

File app/appApp/src/Makefile

```
TOP=../..

include $(TOP)/configure/CONFIG
#----------------------------------------
#  ADD MACRO DEFINITIONS AFTER THIS LINE
#============================

#============================
# Build the IOC application

PROD_IOC_vxWorks = app
# app.dbd will be created and installed
DBD += app.dbd

# app.dbd will be made up from these files:
app_DBD += base.dbd

# Include dbd files from all support applications:
app_DBD += genSubRecord.dbd
app_DBD += xy566Support.dbd
app_DBD += xydig.dbd
app_DBD += motorRecord.dbd
app_DBD += devInfnMtr.dbd

# Add all the support libraries needed by this IOC
app_LIBS += genSub
app_LIBS += xy566
```

```
app_LIBS += xydig
app_LIBS += InfnMtr
app_LIBS += motor

# app_registerRecordDeviceDriver.cpp derives from app.dbd
app_SRCS += app_registerRecordDeviceDriver.cpp
app_SRCS += beamProfileCorrection.c
app_SRCS += faradayCupGainSet.c

# Build the main IOC entry point on workstation OSs.
app_SRCS_DEFAULT += appMain.cpp
app_SRCS_vxWorks += -nil-

# Add support from base/src/vxWorks if needed
#app_OBJS_vxWorks += $(EPICS_BASE_BIN)/vxComLibrary

# Finally link to the EPICS Base libraries
app_LIBS += $(EPICS_BASE_IOC_LIBS)

#===========================

include $(TOP)/configure/RULES
```

Configurazione del Database

(file riportati in ordine alfabetico)

File app/appApp/Db/240port.db

```
record(bo, "$(P)port$(N)") {
  field(DTYP, "XYCOM 240 Port Direction")
  field(DESC, "Dir. $(N) for $(O1) to $(O8) of C${C}")
  field(SCAN, "Passive")
  field(OUT , "#C${C} S$(N) @" )
  field(PINI, "YES")
  field(VAL , 1 )
  field(ZNAM, "Input")
  field(ONAM, "Output")
  field(FLNK, "$(P)port$(N):fout1")
}

record(fanout, "$(P)port$(N):fout1") {
  field(LNK1, "$(P)out$(O1)")
  field(LNK2, "$(P)out$(O2)")
  field(LNK3, "$(P)out$(O3)")
  field(LNK4, "$(P)out$(O4)")
  field(LNK5, "$(P)out$(O5)")
  field(LNK6, "$(P)port$(N):fout2")
}

record(fanout, "$(P)port$(N):fout2") {
  field(LNK1, "TaDiagBmpr01_GainBit_out6")
  field(LNK2, "$(P)out$(O7)")
  field(LNK3, "$(P)out$(O8)")
}
```

File app/appApp/Db/common.substitutions

```
file db/xy566base.db
{
```

```
pattern
{C,P}
{1,"TaDiag_Adco01_"}
}
```

File app/appApp/Db/dbgensub.db

```
record(genSub,"$(PV)") {
             field(DESC,"General Subroutine Record")
             field(SCAN,"Passive")
             field(PINI,"NO")
             field(PHAS,"0")
             field(EVNT,"0")
             field(DISV,"1")
             field(LFLG,"IGNORE")
             field(EFLG,"ALWAYS")
             field(SDIS,"0.0000000000000000e+00")
             field(DISS,"NO_ALARM")
             field(PRIO,"LOW")
             field(FLNK,"TaDiagBmpr01_Hgri_CurrRd")
             field(SUBL,"0.0000000000000000e+00")
             field(BRSV,"NO_ALARM")
             field(INAM,"")
             field(SNAM,"beamProfileCorrection")
             field(UFA,"")
             field(UFVA,"")
             field(FTA,"SHORT")
             field(FTVA,"SHORT")
             field(NOA,"40")
             field(NOVA,"40")
             field(INPA,"TaDiagBmpr01_Hgri_CurrRaw")
             field(OUTA,"TaDiagBmpr01_Hgri_CurrRd")
      }
```

File app/appApp/Db/devInfnMtr.db

```
record(motor,"$(P)$(M)")
{
       field(DESC,"$(DESC)")
       field(DTYP,"$(DTYP)")
       field(DIR,"$(DIR)")
       field(VELO,"$(VELO)")
       field(OUT,"#C$(C) S$(S) @")
       field(MRES,"$(MRES)")
       field(PREC,"$(PREC)")
       field(EGU,"$(EGU)")
       field(DHLM,"$(DHLM)")
       field(DLLM,"$(DLLM)")
       field(INIT,"$(INIT)")
       field(TWV,"4000")
       field(RTRY,"0")
}
```

File app/appApp/Db/emittance_interface_in.db

```
record(ai, "TaDiagEmit01_PlanSt") {
  field( SCAN, "Passive")
```

```
}
record(ai, "TaDiagEmit01_DeltSt") {
  field( SCAN, "Passive")
}
record(ai, "TaDiagEmit01_PosiSt") {
  field( SCAN, "Passive")
}
record(ai, "TaDiagEmit01_PosfSt") {
  field( SCAN, "Passive")
}

record(stringin, "TaDiagEmit01_TitlSt") {
  field( SCAN, "Passive")
  field( DESC, "Title")
}

record(stringin, "TaDiagEmit01_StrgSt01") {
  field( SCAN, "Passive")
  field( DESC, "Source")
}

record(stringin, "TaDiagEmit01_StrgSt02") {
  field( SCAN, "Passive")
  field( DESC, "Isotope")
}

record(ai, "TaDiagEmit01_MassSt") {
  field( SCAN, "Passive")
  field( DESC, "Mass")
}

record(stringout, "TaDiagEmit01_StatSt01") {
  field( SCAN, "Passive")
}


record(ai, "TaDiagEmit01_DateSt01") {
  field( SCAN, "Passive")
}

record(ai, "TaDiagEmit01_DateSt02") {
  field( SCAN, "Passive")
}

record(ai, "TaDiagEmit01_DateSt03") {
  field( SCAN, "Passive")
}

record(ai, "TaDiagEmit01_DateSt04") {
  field( SCAN, "Passive")
}

record(ai, "TaDiagEmit01_DateSt05") {
  field( SCAN, "Passive")
}

record(ai, "TaDiagEmit01_ThreSt") {
  field( SCAN, "Passive")
}

record(stringout, "TaDiagEmit01_StatSt02") {
  field( SCAN, "Passive")
}
```

File app/appApp/Db/emittance_interface_out.db

```
record(waveform, "TaDiagEmit01_Intr01x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 31)
}

record(waveform, "TaDiagEmit01_Intr01y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 31)
}

record(waveform, "TaDiagEmit01_Intr02x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 31)
}

record(waveform, "TaDiagEmit01_Intr02y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 31)
}

record(waveform, "TaDiagEmit01_Intr03x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
}

record(waveform, "TaDiagEmit01_Intr03y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
}

record(waveform, "TaDiagEmit01_Intr04x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
}

record(waveform, "TaDiagEmit01_Intr04y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
}

record(waveform, "TaDiagEmit01_Intr05x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
}

record(waveform, "TaDiagEmit01_Intr05y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
}

record(waveform, "TaDiagEmit01_Intr06x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
```

```
}

record(waveform, "TaDiagEmit01_Intr06y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 2000)
}

record(waveform, "TaDiagEmit01_Intr07x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 500)
}

record(waveform, "TaDiagEmit01_Intr07y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 500)
}

record(waveform, "TaDiagEmit01_Intr08x") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 80)
}

record(waveform, "TaDiagEmit01_Intr08y") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, 80)
}

record(stringout, "TaDiagEmit01:IntrRd01") {
  field( SCAN, "Passive")
  field( DESC, "Date and Time")
}

record(ao, "TaDiagEmit01:IntrRd02") {
  field( SCAN, "Passive")
  field( DESC, "Center position")
  field( PREC, "3")
  field( EGU, "mm")
}

record(ao, "TaDiagEmit01:IntrRd03") {
  field( SCAN, "Passive")
  field( DESC, "Center momentum")
  field( PREC, "3")
  field( EGU, "mrad")
}

record(ao, "TaDiagEmit01:IntrRd04") {
  field( SCAN, "Passive")
  field( DESC, "Beam size")
  field( PREC, "3")
  field( EGU, "mm")
}

record(ao, "TaDiagEmit01:IntrRd05") {
  field( SCAN, "Passive")
  field( DESC, "Beam divergence")
  field( PREC, "3")
  field( EGU, "mrad")
}

record(ao, "TaDiagEmit01:IntrRd06") {
  field( SCAN, "Passive")
```

```
  field( DESC, "Twiss Alpha")
  field( PREC, "3")
}

record(ao, "TaDiagEmit01:IntrRd07") {
  field( SCAN, "Passive")
  field( DESC, "Twiss Beta")
  field( PREC, "3")
  field( EGU, "mm/mrad")
}

record(ao, "TaDiagEmit01:IntrRd08") {
  field( SCAN, "Passive")
  field( DESC, "Twiss Gamma")
  field( PREC, "3")
  field( EGU, "mrad/mm")
}

record(ao, "TaDiagEmit01:IntrRd09") {
  field( SCAN, "Passive")
  field( DESC, "Emittance RMS")
  field( PREC, "3")
  field( EGU, "mm*mrad")
}

record(ao, "TaDiagEmit01:IntrRd10") {
  field( SCAN, "Passive")
  field( DESC, "Beta rel.")
  field( PREC, "6")
}

record(ao, "TaDiagEmit01:IntrRd11") {
  field( SCAN, "Passive")
  field( DESC, "Emittance N rms")
  field( PREC, "4")
  field( EGU, "mm*mrad")
}

record(ao, "TaDiagEmit01:IntrRd12") {
  field( SCAN, "Passive")
  field( DESC, "Max Data")
  field( PREC, "0")
}
```

File app/appApp/Db/emittancemeter.substitutions

```
file db/xy566gain.db
{
pattern
{C,P,CH,N}
{1,"TaDiagEmit01_", "Hgri01_", 3}
{1,"TaDiagEmit01_", "Hgri02_", 4}
{1,"TaDiag", "Emit01_Henc_", 5}
{1,"TaDiagEmit01_", "Vgri01_", 6}
{1,"TaDiagEmit01_", "Vgri02_", 7}
{1,"TaDiag", "Emit01_Venc_", 8}
}

file db/xy566wave.db
{
pattern
{C,P,CH,N,L,FL}
{1,"TaDiagEmit01_", "Hgri01_CurrRd", 3, 40, ""}
{1,"TaDiagEmit01_", "Hgri02_CurrRd", 4, 40, ""}
```

```
{1,"TaDiagEmit01_", "Vgri01_CurrRd", 6, 40, ""}
{1,"TaDiagEmit01_", "Vgri02_CurrRd", 7, 40, ""}
}

file db/xy566val.db
{
pattern
{C,P,CH,N}
{1,"TaDiag", "Emit01_HencRd", 5}
{1,"TaDiag", "Emit01_VencRd", 8}
}

file db/devInfnMtr.db
{
pattern
{            DESC,                       DTYP,  P,           M,
      C,S,DIR,VELO,MRES,PREC,EGU,DHLM,DLLM,INIT}
{"INFN Stepper Axis 2","INFN-
Mtr","TaDiagEmit01_","Motr01",1,1,0,400,1,1,"steps",8388607,-8388608,"0"}
{"INFN Stepper Axis 3","INFN-
Mtr","TaDiagEmit01_","Motr02",1,2,0,400,1,1,"steps",8388607,-8388608,"0"}
}
```

File app/appApp/Db/faradaycup.substitutions

```
file db/xy566gain.db
{
pattern
{C,P,CH,N}
{1,"TaDiag", "Facp01_", 2}
}

file db/xy566val.db
{
pattern
{C,P,CH,N}
{1,"TaDiag", "Facp01_CurrRd", 2}
}

file db/FCaverage.db
{
pattern
{P,CH,CH2}
{"TaDiag", "Facp01_CurrRd", "Facp01"}
}

file db/240port.db
{
pattern
{C,P,            N, O1,O2,O3,O4,O5,O6,O7,O8}
{3,"TaDiagFacp01_GainBit_", 0, 0, 1, 2, 3, 4, 5, 6, 7}
}

file db/xybo.db
{
pattern
{C,P,DTYP,PINI,N}
{3,"TaDiagFacp01_GainBit_", "XYCOM 240", "NO", 0}
{3,"TaDiagFacp01_GainBit_", "XYCOM 240", "NO", 1}
{4,"TaDiagFacp01_", "XYCOM 220", "YES", 0}
}
```

File app/appApp/Db/FCaverage.db

```
record(calc, "$(P)$(CH)a")
{
  field(DESC, "amplification_test")
  field(SCAN,".1 second")
  field(CALC, "A*B")
  field(INPA, "$(P)$(CH)")
  field(INPB, "$(P)$(CH)Gain")
  field(PREC, "3")
  field( EGU , "nA")
  field( MDEL , "-1")
}

record(ai, "$(P)$(CH)Gain")
{
  field(DESC, "amplification_test")
  field(VAL, "1")
}

record(compress, "$(P)$(CH)b") {
  field( DESC, "test_average")
  field( SCAN, ".1 second")
  field( N   , "100")
  field( NSAM, "6")
  field( ALG , "Average")
  field( INP , "$(P)$(CH)a")
}

record(compress, "$(P)$(CH)c") {
  field( DESC, "test_average")
  field( SCAN, "1 second")
  field( N   , "3")
  field( NSAM, "2")
  field( ALG , "N to 1 Average")
  field( INP , "$(P)$(CH)b")
}

record(compress, "$(P)$(CH)d") {
  field( DESC, "test_average")
  field( SCAN, "1 second")
  field( N   , "2")
  field( ALG , "N to 1 Average")
  field( INP , "$(P)$(CH)c")
  field( EGU , "nA")
}

record(bi, "$(P)$(CH2):InteSt") {
  field( DESC, "Start/Stop the integration")
}

record(calc, "$(P)$(CH2)_InteRd")
{
       field(DESC, "FC Integration")
       field(SCAN,".1 second")
       field(CALC, "A + (B*C*D*E)")
       field(INPA, "$(P)$(CH2)_InteRd.VAL  NPP NMS")
       field(INPB, "$(P)$(CH2):InteSt")
       field(INPC, "0.1")
       field(INPD, "0.000000001")
       field(INPE, "$(P)$(CH)a")
       field(EGU, "Coulombs")
}

record(ai, "$(P)$(CH2)_HwGainSt00")
```

```
{
  field(DESC, "Hardware Gain Set Value")
  field( SCAN, "")
  field(VAL, "0")
  field(FLNK, "$(P)$(CH2)_HwGainSt01")
}


record(genSub,"$(P)$(CH2)_HwGainSt01") {
       field(DESC,"General Subroutine Record")
       field(SCAN,"Passive")
       field(PINI,"NO")
       field(PHAS,"0")
       field(EVNT,"0")
       field(DISV,"1")
       field(LFLG,"IGNORE")
       field(EFLG,"ALWAYS")
       field(SDIS,"0.0000000000000000e+00")
       field(DISS,"NO_ALARM")
       field(PRIO,"LOW")
       field(FLNK,"$(P)$(CH2)_HwGain_fanout")
       field(SUBL,"0.0000000000000000e+00")
       field(BRSV,"NO_ALARM")
       field(INAM,"")
       field(SNAM,"faradayCupGainSet")
       field(UFA,"")
       field(UFB,"")
       field(UFC,"")
       field(UFD,"")
       field(UFE,"")
       field(UFF,"")
       field(UFG,"")
       field(UFH,"")
       field(UFI,"")
       field(UFJ,"")
       field(UFVA,"")
       field(UFVB,"")
       field(UFVC,"")
       field(UFVD,"")
       field(UFVE,"")
       field(UFVF,"")
       field(UFVG,"")
       field(UFVH,"")
       field(UFVI,"")
       field(UFVJ,"")
       field(FTA,"DOUBLE")
       field(FTB,"DOUBLE")
       field(FTC,"DOUBLE")
       field(FTD,"DOUBLE")
       field(FTE,"DOUBLE")
       field(FTF,"DOUBLE")
       field(FTG,"DOUBLE")
       field(FTH,"DOUBLE")
       field(FTI,"DOUBLE")
       field(FTJ,"DOUBLE")
       field(FTVA,"ENUM")
       field(FTVB,"ENUM")
       field(FTVC,"ENUM")
       field(FTVD,"DOUBLE")
       field(FTVE,"STRING")
       field(FTVF,"DOUBLE")
       field(FTVG,"DOUBLE")
       field(FTVH,"DOUBLE")
       field(FTVI,"DOUBLE")
       field(FTVJ,"DOUBLE")
       field(NOA,"1")
       field(NOB,"1")
```

```
       field(NOC,"1")
       field(NOD,"1")
       field(NOE,"1")
       field(NOF,"1")
       field(NOG,"1")
       field(NOH,"1")
       field(NOI,"1")
       field(NOJ,"1")
       field(NOVA,"1")
       field(NOVB,"1")
       field(NOVC,"1")
       field(NOVD,"1")
       field(NOVE,"1")
       field(NOVF,"1")
       field(NOVG,"1")
       field(NOVH,"1")
       field(NOVI,"1")
       field(NOVJ,"1")
       field(INPA,"$(P)$(CH2)_HwGainSt00")
       field(INPB,"0.000000000000000e+00")
       field(INPC,"0.000000000000000e+00")
       field(INPD,"0.000000000000000e+00")
       field(INPE,"0.000000000000000e+00")
       field(INPF,"0.0000000000000000e+00")
       field(INPG,"0.0000000000000000e+00")
       field(INPH,"0.0000000000000000e+00")
       field(INPI,"0.0000000000000000e+00")
       field(INPJ,"0.0000000000000000e+00")
       field(OUTA,"$(P)$(CH2)_GainBit_port0")
       field(OUTB,"$(P)$(CH2)_GainBit_out0")
       field(OUTC,"$(P)$(CH2)_GainBit_out1")
       field(OUTD,"$(P)$(CH2)_CurrRdGain")
       field(OUTE,"$(P)$(CH2)_GainStatus")
       field(OUTF,"0.000000000000000e+00")
       field(OUTG,"0.000000000000000e+00")
       field(OUTH,"0.000000000000000e+00")
       field(OUTI,"0.000000000000000e+00")
       field(OUTJ,"0.000000000000000e+00")
}

record(stringout, "$(P)$(CH2)_GainStatus")
{
  field(DESC, "FC Gain Status")
  field(VAL, "Gain NOT set.")
}

record(fanout, "$(P)$(CH2)_HwGain_fanout") {
  field(LNK1, "$(P)$(CH2)_GainBit_port0")
  field(LNK2, "$(P)$(CH2)_GainBit_out0")
  field(LNK3, "$(P)$(CH2)_GainBit_out1")
  field(LNK4, "$(P)$(CH2)_CurrRdGain")
  field(LNK5, "$(P)$(CH2)_GainStatus")
}
```

File app/appApp/Db/Makefile

```
TOP=../..
include $(TOP)/configure/CONFIG
#----------------------------------------
#  ADD MACRO DEFINITIONS AFTER THIS LINE

#----------------------------------------------------
#  Optimization of db files using dbst (DEFAULT: NO)
```

```
#DB_OPT = YES

DB += xy566base.db
DB += xy566gain.db
DB += xy566val.db
DB += xy566wave.db
DB += 240port.db
DB += xybi.db
DB += xybo.db
DB += devInfnMtr.db
DB += FCaverage.db
DB += waveform.db
DB += dbgensub.db
DB += emittance_interface_in.db
DB += emittance_interface_out.db

DB += common.substitutions
DB += faradaycup.substitutions
DB += beamprofiler.substitutions
DB += emittancemeter.substitutions
DB += provasubgen.substitutions

include $(TOP)/configure/RULES
```

File app/appApp/Db/provasubgen.substitutions

```
file db/dbgensub.db
{
pattern
{PV}
{"TaDiagBmpr01_Hgri_CurrElab"}
}
```

File app/appApp/Db/waveform.db

```
record(waveform, "$(PV)") {
  field(SCAN, "Passive")
  field( FTVL, "DOUBLE")
  field( NELM, $(L))
}
```

File app/appApp/Db/xy566base.db

```
record(bi, "$(P)IntrRd") {
  field( DTYP, "XYCOM 566")
  field( INP , "#C$(C) S0 @")
  field( SCAN, "1 second")
  field( ZNAM, "Stopped")
  field( ONAM, "Running")
}

record(bo, "$(P)IntrSt") {
  field( DTYP, "XYCOM 566")
  field( OUT , "#C$(C) S0 @")
  field( SCAN, ".1 second")
  field( ZNAM, "Stop")
  field( ONAM, "Run")
```

```
  field( RVAL, 1 )
  field( PINI, "YES")
}
```

File app/appApp/Db/xy566gain.db

```
record(mbbo, "$(P)$(CH)GainSt") {
  field( DTYP, "XYCOM 566 Gain")
  field( OUT , "#C$(C) S$(N) @")
  field( ZRST, "Gain A")
  field( ONST, "Gain B")
  field( TWST, "Gain C")
  field( THST, "Gain D")
  field( ZRVL, 0)
  field( ONVL, 1)
  field( TWVL, 2)
  field( THVL, 3)
  field( VAL , 0 )
  field( PINI, "YES")
}
```

File app/appApp/Db/xy566val.db

```
record(ai, "$(P)$(CH)") {
  field( DESC, "Card $(C) channel $(N)")
  field( DTYP, "XYCOM 566")
  field( SCAN, "I/O Intr")
  field( INP , "#C$(C) S$(N) @")
  field( ASLO, "0.00244")
  field( EGU , "nA")
  field( MDEL , "-1")
}
```

File app/appApp/Db/xy566wave.db

```
record(waveform, "$(P)$(CH)") {
  field( DESC, "Card $(C) channel $(N)")
  field( DTYP, "XYCOM 566")
  field( SCAN, "I/O Intr")
  field( INP , "#C$(C) S$(N) @")
  field( FTVL, "DOUBLE")
  field( NELM, $(L))
  field( FLNK, "$(FL)")
}
```

File app/appApp/Db/xybi.db

```
record(bi, "$(P)in$(N)") {
  field(DTYP, "$(DTYP)")
  field(SCAN, "$(SCAN)")
  field(INP , "#C${C} S$(N) @" )
  field(ZNAM, "Low")
  field(ONAM, "High")
}
```

File app/appApp/Db/xybo.db

```
record(bo, "$(P)out$(N)") {
  field(DTYP, "$(DTYP)")
  field(SCAN, "Passive")
  field(OUT , "#C${C} S$(N) @" )
  field(PINI, "${PINI}")
  field(VAL , 0 )
  field(ZNAM, "Low")
  field(ONAM, "High")
}
```

Subroutines

File app/appApp/src/beamProfileCorrection.c

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

#include <dbEvent.h>
#include <dbDefs.h>
#include <dbCommon.h>
#include <registryFunction.h>
#include <epicsExport.h>
#include <recSup.h>
#include <genSubRecord.h>
#include <pinfo.h>

long beamProfileCorrection( genSubRecord *pgsub )
{
  short *ptr;

  ptr = (short *)pgsub->a;
  int i=0;
  short num[40];
  for (i=0; i<40;i++)
  {
    num[i]= *(ptr++);
  }

  num[11]=(num[10]+num[12])/2;
  num[13]=(num[12]+num[14])/2;
  num[21]=(num[20]+num[22])/2;
  num[23]=(num[22]+num[24])/2;
  num[29]=(num[28]+num[30])/2;
  num[31]=(num[30]+num[32])/2;

  memcpy( pgsub ->vala, num, 40*sizeof(short));
  return(0);
}

epicsRegisterFunction (beamProfileCorrection);
```

File app/appApp/src/faradayCupGainSet.c

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

#include <dbEvent.h>
#include <dbDefs.h>
#include <dbCommon.h>
#include <registryFunction.h>
#include <epicsExport.h>
#include <recSup.h>
#include <genSubRecord.h>
#include <pinfo.h>

long faradayCupGainSet( genSubRecord *pgsub )
{
  double *gain;
  gain = (double *)pgsub->a;
  double gainval= *gain;

  short port0, out0, out1;
  double gaincalc;
  char status[40];

  port0=1;
  if ((gainval>3.5)&&(gainval<4.5))
  {
    out0=1;
    out1=1;
    gaincalc=1000;
    strcpy(status, "Gain D. Max=10.000nA. Min=20nA.");
  }
  else if ((gainval>2.5)&&(gainval<3.5))
  {
    out0=0;
    out1=1;
    gaincalc=100;
    strcpy(status, "Gain C. Max=1.000nA. Min=2nA.");
  }
  else if ((gainval>1.5)&&(gainval<2.5))
  {
    out0=1;
    out1=0;
    gaincalc=10;
    strcpy(status, "Gain B. Max=100nA. Min=200pA.");
  }
  else if ((gainval>0.5)&&(gainval<1.5))
  {
    out0=0;
    out1=0;
    gaincalc=1;
    strcpy(status, "Gain A. Max=10nA. Min=10pA.");
  }
  else
  {
    out0=0;
    out1=0;
    gaincalc=1;
    strcpy(status, "Gain NOT set.");
  }

  memcpy( pgsub->vala, &port0, sizeof(short));
  memcpy( pgsub->valb, &out0, sizeof(short));
  memcpy( pgsub->valc, &out1, sizeof(short));
```

```
  memcpy( pgsub->vald, &gaincalc, sizeof(double));
  memcpy( pgsub->vale, &status, 40*sizeof(char));

  return(0);
}

epicsRegisterFunction (faradayCupGainSet);
```

Appendice C – Parametri di avvio di VxWorks

I parametri di boot impostati possono essere visualizzati con il comando "p" sul prompt di boot di VxWorks e possono essere cambiati con il comando "c".

I parametri utilizzati sono:

boot device: mottsec0
processor number: 0
host name: mylinuxpc
file name: /home/epics/workspace/nicola/vxImage1bis/default/vxWorks
inet on ethernet: 10.5.0.23
inet on backplane:
host inet: 10.5.0.28
gateway inet:
user: confort
ftp password:
flags: 0x0
target name:
startup script:
/home/confort/workspace/nicola/prova10/app/iocBoot/iocapp/st.cmd
other:

Dove:
- file name è la posizione dell'immagine di vxWorks sul computer host.
- inet on ethernet è l'indirizzo ip della scheda VME.
- host inet è l'indirizzo ip del computer host da cui fare il bootstrap.
- startup script è la posizione dello script sul computer host.

Appendice D – Script di avvio

File app/iocBoot/iocapp/st.cmd

```
ifconfig "mottsec0 10.5.0.23 netmask 255.255.255.0"

cd "/home/confort/workspace/nicola/prova10/app/iocBoot/iocapp"

< cdCommands
putenv "EPICS_TS_NTP_INET=10.5.0.254"
putenv "EPICS_TIMEZONE=CET::-60:032802:103102"

cd topbin

ld < app.munch

## Register all support components
cd top
dbLoadDatabase "dbd/app.dbd"
app_registerRecordDeviceDriver pdbbase

############### ADC card 1 ####################

xycom566setup(1, 0xf800, 0xa00000, 3, 0xf2, 1)

stc566seqmulti(1, 2, 0x100, 0x4800)

seq566set(1, 0, 40, 1, 1, 40)
seq566set(1, 1, 40, 2, 1, 40)
seq566set(1, 2, 1, 3, 1, 1)
seq566set(1, 3, 40, 4, 1, 40)
seq566set(1, 4, 40, 5, 1, 40)
seq566set(1, 5, 1, 6, 1, 1)
seq566set(1, 6, 40, 7, 1, 40)
seq566set(1, 7, 40, 8, 1, 40)
seq566set(1, 8, 1, 9, 1, 1)

xycom566finish()

############# Xycom 240 ##################
xycom240setup(3,0xe000)

############# Xycom 220 ##################
xycom220setup(4,0x3000)

############ Stepper Motor #################
```

```
# INFN Stepper motor module 1,VME A16 addr 0x5000,update at 10 Hz
infnModConfig(1,0x5000,10)

## Load record instances
dbLoadTemplate("db/common.substitutions")
dbLoadTemplate("db/faradaycup.substitutions")
dbLoadTemplate("db/beamprofiler.substitutions")
dbLoadTemplate("db/provasubgen.substitutions")
dbLoadTemplate("db/emittancemeter.substitutions")
dbLoadRecords("db/emittance_interface_in.db")
dbLoadRecords("db/emittance_interface_out.db")

cd startup
iocInit
```

Appendice E – Programmi per la misura di emittanza

File acquisition.c

```
#define TIMEOUT 1.0
#define SCA_OK 1
#define SCA_ERR 0
#define MAX_STRING 40

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cadef.h>

#include "acquisition.h"

/* Function prototypes */
int main(int argc, char **argv);

/* Global variables */
double timeout=TIMEOUT;
long year, month, day, hour, minute;
int stat;
chid pCh1,pCh2,pCh3,pCh4;
char comando[255];
char tempstring[40];
struct dbr_time_double * pTD;
int plane;
double deltaum, posinium, posfinum;
long deltastep, posinistep, posfinstep;
char status[40];
int mass;
int endswitch1,endswitch2,movn1,movn2;
unsigned nBytes;
char path[255];
char comando[255];
long templong;
int one=1;
short motormoving;
int motorposition;
clock_t endwait;
short  datagrid1[40];
short  datagrid2[40];
int i;
int iterations;
int tempint;
```

```
short tempshort;
double tempdouble;
int pvSpecified=0;
char name[MAX_STRING];
char value[MAX_STRING];
double data;
int    shortdata;
short  shortarray[40];


/****************** main ************************/
int main(int argc, char **argv)
{

  /* Initialize */
    stat=ca_context_create(ca_disable_preemptive_callback);
    if(stat != ECA_NORMAL) {
       printf ("ca_context_create failed:\n%s\n",ca_message(stat));
       exit(1);
    }

    /********************** READ MAIN PARAMETERS ***************/
    stat=ca_create_channel (indata01,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
/*Plane*/
    stat=ca_create_channel (indata02,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh2);
/*Delta*/
    stat=ca_create_channel (indata03,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh3);
/*Initial Position*/
    stat=ca_create_channel (indata04,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh4);
/*Final Position*/
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_SHORT,pCh1,&plane);
    stat=ca_get(DBR_DOUBLE,pCh2,&deltaum);
    stat=ca_get(DBR_DOUBLE,pCh3,&posinium);
    stat=ca_get(DBR_DOUBLE,pCh4,&posfinum);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_clear_channel(pCh2);
    stat=ca_clear_channel(pCh3);
    stat=ca_clear_channel(pCh4);
    stat=ca_create_channel (indata08,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_SHORT,pCh1,&mass);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);

    /********************** CHECK PARAMETERS *****************/
    if ((plane!=0)&&(plane!=1))
    {
      strcpy(status,"Plane Error.");
      stat=ca_create_channel
(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
      return(0);
    }
    if ((deltaum<0.01)||(deltaum>10))
    {
      strcpy(status,"Delta Error.");
      stat=ca_create_channel
(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
```

```
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
      return(0);
    }
    if ((posinium<-70)||(posinium>20))
    {
      strcpy(status,"Initial Position Error.");
      stat=ca_create_channel
(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
      return(0);
    }
    if ((posfinum<-20)||(posfinum>30)|| (posfinum<posinium+deltaum*10))
    {
      strcpy(status,"Final Position Error.");
      stat=ca_create_channel
(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
      return(0);
    }
    if ((mass<1)||(mass>1000))
    {
      strcpy(status,"Mass Error.");
      stat=ca_create_channel
(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
      return(0);
    }

    /*************** CHECK MOTORS POSITION ***************/
    stat=ca_create_channel (X_motorLLS,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
/*X_motorLLS*/
    stat=ca_create_channel
(X_motorMOVN,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh2); /*X_motorMOVN*/
    stat=ca_create_channel (Y_motorLLS,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh3);
/*Y_motorLLS*/
    stat=ca_create_channel
(Y_motorMOVN,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh4); /*Y_motorMOVN*/
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_SHORT,pCh1,&endswitch1);
    stat=ca_get(DBR_SHORT,pCh2,&movn1);
    stat=ca_get(DBR_SHORT,pCh3,&endswitch2);
    stat=ca_get(DBR_SHORT,pCh4,&movn2);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_clear_channel(pCh2);
    stat=ca_clear_channel(pCh3);
    stat=ca_clear_channel(pCh4);
    /*
    if ((endswitch1!=1)||(endswitch2!=1)|| (movn1!=0)||(movn2!=0))
    {
      strcpy(status,"Motor Position Error.");
      stat=ca_create_channel
(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
```

```
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
      return(0);
    }
    */


    /*************** START WRITING RAWFILE *********************/
    stat=ca_create_channel(X_encoder,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    nBytes = dbr_size_n ( DBR_TIME_DOUBLE, 1 );
    pTD = ( struct dbr_time_double * ) malloc ( nBytes );
    if ( ! pTD ) {
        fprintf ( stderr, "insufficient memory to complete request\n" );
        return -1;
    }
    stat = ca_array_get ( DBR_TIME_DOUBLE, 1, pCh1, pTD );
    stat=ca_pend_io(timeout);

    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "%Y", & pTD-
>stamp );
    year=atoi(tempstring);
    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "%m", & pTD-
>stamp );
    month=atoi(tempstring);
    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "%d", & pTD-
>stamp );
    day=atoi(tempstring);
    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "%H", & pTD-
>stamp );
    hour=atoi(tempstring);
    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "%M", & pTD-
>stamp );
    minute=atoi(tempstring);
    strcpy(path,folder1);
    strcpy(comando, "mkdir ");
    strcat(comando,path);
    system(comando);
    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "/%Y%m%d", &
pTD->stamp );
    strcat(comando,tempstring);
    strcat(path,tempstring);
    system(comando);
    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "/%H%M", & pTD-
>stamp );
    strcat(comando,tempstring);
    strcat(path,tempstring); /*Copy of the folder path */
    system(comando); /*Create folder Data/<date>/<time> */

    strcpy(comando,path);
    strcat(comando,"/");
    strcat(comando,file1);
    FILE *outputfile = NULL;
    outputfile = fopen(comando,"w");
    if (outputfile==NULL)
    {
      strcpy(status,"Cannot create output file.");
      stat=ca_create_channel
(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      ca_context_destroy();
```

```
      return(0);
    }

    fprintf(outputfile,"EMITTANCE MEASUREMENT\n");
    epicsTimeToStrftime ( tempstring, sizeof ( tempstring ), "%Y %m %d - %H
%M", & pTD->stamp );
    fprintf(outputfile,"%s\n",tempstring);


    ca_clear_channel ( pCh1 );
    free ( pTD );


    /*************** READ/WRITE OTHER PARAMETERS ************************/
    stat=ca_create_channel (indata05,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
/*Title*/
    stat=ca_create_channel (indata06,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh2);
/*Source*/
    stat=ca_create_channel (indata07,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh3);
/*Isotope*/
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    fprintf(outputfile,"TITLE: %s\n",tempstring);
    stat=ca_get(DBR_STRING,pCh2,&tempstring);
    stat=ca_pend_io(timeout);
    fprintf(outputfile,"SOURCE: %s\n",tempstring);
    stat=ca_get(DBR_STRING,pCh3,&tempstring);
    stat=ca_pend_io(timeout);
    fprintf(outputfile,"ISOTOPE: %s\n",tempstring);
    stat=ca_clear_channel(pCh1);
    stat=ca_clear_channel(pCh2);
    stat=ca_clear_channel(pCh3);
    fprintf(outputfile,"MASS: %i\n",mass);

    stat=ca_create_channel (TaVoltage,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_LONG,pCh1,&templong);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"VOLTAGE: %li\n",templong);

    stat=ca_create_channel (TaPuller,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"PULLER: %s\n",tempstring);
    stat=ca_create_channel (TaVacuum1,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"VACUUM1: %s\n",tempstring);
    stat=ca_create_channel (TaVacuum2,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"VACUUM2: %s\n",tempstring);
    stat=ca_create_channel (TaHeat,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"HEAT: %s\n",tempstring);
    stat=ca_create_channel (TaLine,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
```

```
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"LINE: %s\n",tempstring);
    stat=ca_create_channel (TaMagnet,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"MAGNET: %s\n",tempstring);
    stat=ca_create_channel (TaOven,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"OVEN: %s\n",tempstring);
    stat=ca_create_channel (TaAnode,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_STRING,pCh1,&tempstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    fprintf(outputfile,"ANODE: %s\n",tempstring);


    /************* SETTING MOTOR AND ACQUISITION ***************/
    char encoder[40];
    char grid1[40];
    char grid2[40];
    char motorLLS[40];
    char motorHLS[40];
    char motorVELO[40];
    char motorMOVN[40];
    char motorTWV[40];
    char motorTWR[40];
    char motorTWF[40];
    printf("%d\n",plane);
    if (plane==0)
    {
      fprintf(outputfile,"X plane\n");
      strcpy(encoder,X_encoder);
      strcpy(grid1,X_grid1);
      strcpy(grid2,X_grid2);
      strcpy(motorLLS,X_motorLLS);
      strcpy(motorHLS,X_motorHLS);
      strcpy(motorVELO,X_motorVELO);
      strcpy(motorMOVN,X_motorMOVN);
      strcpy(motorTWV,X_motorTWV);
      strcpy(motorTWR,X_motorTWR);
      strcpy(motorTWF,X_motorTWF);
    }
    else
    {
      fprintf(outputfile,"Y plane\n");
      strcpy(encoder,Y_encoder);
      strcpy(grid1,Y_grid1);
      strcpy(grid2,Y_grid2);
      strcpy(motorLLS,Y_motorLLS);
      strcpy(motorHLS,Y_motorHLS);
      strcpy(motorVELO,Y_motorVELO);
      strcpy(motorMOVN,Y_motorMOVN);
      strcpy(motorTWV,Y_motorTWV);
      strcpy(motorTWR,Y_motorTWR);
      strcpy(motorTWF,Y_motorTWF);
    }

    deltastep=deltaum*400;
```

```
    posinistep=posinium*25.9+2428;
    posfinstep=posfinum*25.9+2428;

    fprintf(outputfile,"%li delta\n",deltastep);
    fprintf(outputfile,"%li initial position\n",posinistep);
    fprintf(outputfile,"%li final position\n",posfinstep);

    /************* DATA ACQUISITION ******************/
    /*Take the slit to the initial position*/
    templong=(posinium+70)*400;    stat=ca_create_channel
(motorTWV,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&templong);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_create_channel (motorTWF,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_SHORT,pCh1,&one);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    do {
      endwait = clock () + 0.1 * CLOCKS_PER_SEC ; /* 0.1 seconds delay to be
sure that the MOVN signal is done by the Stepper Motor board */
      while (clock() < endwait)
       {
       }
    stat=ca_create_channel (motorMOVN,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_SHORT,pCh1,&motormoving);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    }
    while (motormoving==1);

    /*Data Acquisition*/
    iterations=0;
    fprintf(outputfile,"RAW DATA:\n");
    templong=deltastep;
    stat=ca_create_channel (motorTWV,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&templong);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_create_channel (encoder,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh2); /*
Encoder */
    stat=ca_create_channel (grid1,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh3); /*
Grid 1 */
    stat=ca_create_channel (grid2,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh4); /*
Grid 2 */
    stat=ca_pend_io(timeout);
    do {
      /*Read raw data and write to file*/
      stat=ca_get(DBR_SHORT,pCh2,&motorposition);
      stat=ca_array_get(DBR_SHORT,40,pCh3,&datagrid1);
      stat=ca_array_get(DBR_SHORT,40,pCh4,&datagrid2);
      stat=ca_pend_io(timeout);
      fprintf(outputfile,"%i\n",motorposition);
      for (i=0;i<40;i++)
      {
       fprintf(outputfile,"%5i",datagrid1[i]);
      }
      for (i=0;i<40;i++)
      {
       fprintf(outputfile,"%5i",datagrid2[i]);
      }
      fprintf(outputfile,"\n");
      iterations++;
```

```
      /*Send motor forward*/
      stat=ca_create_channel (motorTWF,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_SHORT,pCh1,&one);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
      do {
      endwait = clock () + 0.1 * CLOCKS_PER_SEC ; /* 0.1 seconds delay to be
sure that the MOVN signal is done by the Stepper Motor board */
      while (clock() < endwait)
       {
       }
    stat=ca_create_channel (motorMOVN,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_SHORT,pCh1,&motormoving);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    }
    while (motormoving==1); /*Keep cicling until motor stops*/

    /*Check if acquisition is done*/
    i=0;
    stat=ca_create_channel (motorHLS,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
/*motorHLS*/
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_SHORT,pCh1,&tempint);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    if ((motorposition>posfinstep)|| (iterations>500)||(tempint!=0))
      i=1;


    } while (i==0);

    stat=ca_clear_channel(pCh2);
    stat=ca_clear_channel(pCh3);
    stat=ca_clear_channel(pCh4);

    /************ FINISHES TO WRITE RAWFILE **************/
    fprintf(outputfile,"END DATA\n");
    fprintf(outputfile,"%i iterations",iterations);



    /********** TAKE THE MOTOR BACK ********************/
    templong=50000;
    stat=ca_create_channel(motorTWV,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&templong);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_create_channel(motorTWR,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_SHORT,pCh1,&one);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);


    /*************** SET EPICS CALCULATION VARIABLES **********/
    stat=ca_create_channel
(datetime1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&year);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_create_channel
(datetime2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
```

```
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&month);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_create_channel
(datetime3Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&day);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_create_channel
(datetime4Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&hour);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);
    stat=ca_create_channel
(datetime5Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_LONG,pCh1,&minute);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh1);

  /* Exit */
  strcpy(status,"OK.");

stat=ca_create_channel(status1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh1);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh1,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh1);
    ca_context_destroy();
    return(0);

}
```

File acquisition.h

```
char folder1[] = "Data";
char file1[] = "rawdata";

long maxsamples = 500;

char indata01[] = "TaDiagEmit01_PlanSt";
char indata02[] = "TaDiagEmit01_DeltSt";
char indata03[] = "TaDiagEmit01_PosiSt";
char indata04[] = "TaDiagEmit01_PosfSt";
char indata05[] = "TaDiagEmit01_TitlSt";
char indata06[] = "TaDiagEmit01_StrgSt01";
char indata07[] = "TaDiagEmit01_StrgSt02";
char indata08[] = "TaDiagEmit01_MassSt";
char status1Set[] = "TaDiagEmit01_StatSt01";

char TaVoltage[] = "TaSourExtr_Hvps_VoltRd01";
char TaPuller[] = "TaSourExtr_PosiRd01";
char TaVacuum1[] = "TaVacuCham_PosiRd01";
char TaVacuum2[] = "TaVacuCham_PosiRd02";
char TaHeat[] = "TaHeat_Hcps_CurrRd";
char TaLine[] = "TaIoniSurf_Hcps_CurrRd";
char TaMagnet[] = "TaBtraMagn_Hcps_CurrRd";
char TaOven[] = "TaIoniOven_Hcps:Powr";
char TaAnode[] = "TaIoniPlas_Hvps_VoltRd";

char X_encoder[] = "TaDiagEmit01_HencRd.RVAL";
```

```
char X_grid1[] = "TaDiagEmit01_Hgri01_CurrRd";
char X_grid2[] = "TaDiagEmit01_Hgri02_CurrRd";
char X_motorLLS[] = "TaDiagEmit01_Motr01.LLS";
char X_motorHLS[] = "TaDiagEmit01_Motr01.HLS";
char X_motorVELO[] = "TaDiagEmit01_Motr01.VELO";
char X_motorMOVN[] = "TaDiagEmit01_Motr01.MOVN";
char X_motorTWV[] = "TaDiagEmit01_Motr01.TWV";
char X_motorTWR[] = "TaDiagEmit01_Motr01.TWR";
char X_motorTWF[] = "TaDiagEmit01_Motr01.TWF";

char Y_encoder[] = "TaDiagEmit01_VencRd.RVAL";
char Y_grid1[] = "TaDiagEmit01_Vgri01_CurrRd";
char Y_grid2[] = "TaDiagEmit01_Vgri02_CurrRd";
char Y_motorLLS[] = "TaDiagEmit01_Motr02.LLS";
char Y_motorHLS[] = "TaDiagEmit01_Motr02.HLS";
char Y_motorVELO[] = "TaDiagEmit01_Motr02.VELO";
char Y_motorMOVN[] = "TaDiagEmit01_Motr02.MOVN";
char Y_motorTWV[] = "TaDiagEmit01_Motr02.TWV";
char Y_motorTWR[] = "TaDiagEmit01_Motr02.TWR";
char Y_motorTWF[] = "TaDiagEmit01_Motr02.TWF";

char datetime1Set[] = "TaDiagEmit01_DateSt01";
char datetime2Set[] = "TaDiagEmit01_DateSt02";
char datetime3Set[] = "TaDiagEmit01_DateSt03";
char datetime4Set[] = "TaDiagEmit01_DateSt04";
char datetime5Set[] = "TaDiagEmit01_DateSt05";
```

File calculation.c

```
#define TIMEOUT 1.0
#define SCA_OK 1
#define SCA_ERR 0
#define MAX_STRING 40

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cadef.h>
#include <ctype.h>
#include <math.h>

#include "calculation.h"

/* Function prototypes */
int main(int argc, char **argv);

/* Global variables */
long year, month, day, hour, minute;
long threshold;
double timeout=TIMEOUT;
char status[40];
char path[255];
char comando[255];
char tempstring[5];
char inputstring[500];
long templong;
int i,j;
int stat;
chid pCh;

long iterations;

#include "readfilefunctions.c"
```

```
/****************** main *********************/
int main(int argc, char **argv)
{
  /* Initialize */
    stat=ca_context_create (ca_disable_preemptive_callback);
    if(stat != ECA_NORMAL) {
       printf ("ca_context_create failed:\n%s\n", ca_message(stat));
       exit(1);
    }

  /*************** READ DATE AND TIME **************/
  /* Search */
    stat=ca_create_channel
(datetime1Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_LONG,pCh,&year);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

    stat=ca_create_channel
(datetime2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_LONG,pCh,&month);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

    stat=ca_create_channel
(datetime3Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_LONG,pCh,&day);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

    stat=ca_create_channel
(datetime4Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_LONG,pCh,&hour);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

    stat=ca_create_channel
(datetime5Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_LONG,pCh,&minute);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

  /************ READ THRESHOLD ***********/
    stat=ca_create_channel
(thresholdSet,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_get(DBR_LONG,pCh,&threshold);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

  /************  CHECK PARAMETERS ***********/
    if ((year<1900)||(year>2500) ||(month>12)||(month<1) ||(day>31)||(day<1)
||(hour>23)||(hour<0)|| (minute>59)||(minute<0)||
(threshold>4095)||(threshold<0))
    {
      strcpy(status,"Parameter Error.");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
```

```
      ca_context_destroy();
      return(0);
    }
    else
    {
      strcpy(status,"Parameters Ok.");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
    }

    /**************** ACCESS RAWDATA FILE *************/
    /*creation of the folder path*/
    path[0] = '0' + (year / 1000);
    templong = year%1000;
    path[1] = '0' + (templong / 100);
    templong = templong%100;
    path[2] = '0' + (templong / 10);
    templong = templong%10;
    path[3] = '0' + (templong / 1);
    path[4] = '0' + (month / 10);
    templong = month%10;
    path[5] = '0' + (templong / 1);
    path[6] = '0' + (day / 10);
    templong = day%10;
    path[7] = '0' + (templong / 1);
    path[8] = '/';
    path[9] = '0' + (hour / 10);
    templong = hour%10;
    path[10] = '0' + (templong / 1);
    path[11] = '0' + (minute / 10);
    templong = minute%10;
    path[12] = '0' + (templong / 1);
    tempstring[0] = '/';
    strcpy(comando, folder1);
    strcat(comando,tempstring);
    strcat(comando,path);
    strcat(comando,tempstring);
    strcpy(path, comando); /*saves a copy of the folder path*/
    strcat(comando,file1);

    /*Access to the rawfile*/
    FILE *textfile = NULL;
    textfile = fopen(comando,"r");
    if (textfile==NULL)
    {
      strcpy(status,"Cannot find requested file.");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
      return(0);
    }
    else
    {
      strcpy(status,"File access successful.");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
```

```
  stat=ca_clear_channel(pCh);
}


/********* CHECKING THE RAWDATA FILE ***********/
int line=1;
/*ROW ONE, checks the first word*/
fscanf(textfile, "%s", inputstring);
if (strcmp(inputstring, "EMITTANCE") != 0)
{launchlineerror(line);
return(0);}
nextline(textfile);

/*ROW TWO, checks the first digit*/
line++;
i = fgetc(textfile);
if (isdigit(i)==0)
{launchlineerror(line);
return(0);}
nextline(textfile);

/*ROW THREE, checks the title tag*/
line++;
fscanf(textfile, "%s", inputstring);
if (strcmp(inputstring, "TITLE:") != 0)
{launchlineerror(line);
return(0);}
nextline(textfile);

/*ROW FOUR, checks the source tag*/
line++;
fscanf(textfile, "%s", inputstring);
if (strcmp(inputstring, "SOURCE:") != 0)
{launchlineerror(line);
return(0);}
nextline(textfile);

/*ROW FIVE, checks the isotope tag*/
line++;
fscanf(textfile, "%s", inputstring);
if (strcmp(inputstring, "ISOTOPE:") != 0)
{launchlineerror(line);
return(0);}
nextline(textfile);

/*ROW SIX, checks the mass tag*/
line++;
fscanf(textfile, "%s", inputstring);
if (strcmp(inputstring, "MASS:") != 0)
{launchlineerror(line);
return(0);}
nextline(textfile);

/*ROW SEVEN, checks the voltage tag*/
line++;
fscanf(textfile, "%s", inputstring);
if (strcmp(inputstring, "VOLTAGE:") != 0)
{launchlineerror(line);
return(0);}
nextline(textfile);

/*ROW EIGHT, checks the puller tag*/
line++;
fscanf(textfile, "%s", inputstring);
if (strcmp(inputstring, "PULLER:") != 0)
{launchlineerror(line);
return(0);}
```

```
    nextline(textfile);

    /*ROW NINE, checks the vacuum1 tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "VACUUM1:") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW TEN, checks the vacuum2 tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "VACUUM2:") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW ELEVEN, checks the heat tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "HEAT:") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW TWELVE, checks the line tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "LINE:") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW THIRTEEN, checks the magnet tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "MAGNET:") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW FOURTEEN, checks the oven tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "OVEN:") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW FIFTEEN, checks the anode tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "ANODE:") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW SIXTEEN, checks the first digit*/
    line++;
    i = fgetc(textfile);
    if ((i != 'X')&&(i != 'Y'))
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW SEVENTEEN, checks the first digit*/
```

```
    line++;
    i = fgetc(textfile);
    if (isdigit(i)==0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW EIGHTEEN, checks the first digit*/
    line++;
    i = fgetc(textfile);
    if (isdigit(i)==0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW NINETEEN, checks the first digit*/
    line++;
    i = fgetc(textfile);
    if (isdigit(i)==0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*ROW TWENTY, checks the raw tag*/
    line++;
    fscanf(textfile, "%s", inputstring);
    if (strcmp(inputstring, "RAW") != 0)
    {launchlineerror(line);
    return(0);}
    nextline(textfile);

    /*DATA ROWS, checks the first char of each row*/
    line++;
    templong=0; /*to count the number of acquisitions*/
    i = fgetc(textfile);
    while (i != 'E')
    {
      if (isdigit(i)==0)
      {launchlineerror(line);
      return(0);}
      nextline(textfile);

      line++;
      i = fgetc(textfile);
      if (i!=' ')
      {launchlineerror(line);
      return(0);}
      nextline(textfile);

      templong++;
      line++;
      i = fgetc(textfile);
    }
    nextline(textfile);

    /*LAST ROW, number of iterations*/
    fscanf(textfile, "%li", &iterations);
    if (iterations != templong)
    {
      strcpy(status,"Data Error. Wrong iteration number");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
```

```
  return(0);
}
fclose(textfile);

/********** VARIABLES DECLARATION *********/
int mass;
long voltage;
int plane;
int deltastep;
long posinistep;
long posfinstep;
int rawdata[iterations][wiresnumber];
int maxdata;
int positionsstep[iterations];
double positionsmm[iterations]; /*equivalente variabile xd*/
double wiresstepmrad; /*equivalente variabile asd*/
double wiresmrad[wiresnumber]; /*equivalente variabile xpd*/
long px[iterations]; /*Sum of currents for every position*/
long pxp[wiresnumber]; /*Sum of the currents for every wire*/

double Intr01x[Intr01size];
double Intr01y[Intr01size];
double Intr02x[Intr01size];
double Intr02y[Intr01size];
double Intr03x[Intr03size];
double Intr03y[Intr03size];
double Intr04x[Intr03size];
double Intr04y[Intr03size];
double Intr05x[Intr03size];
double Intr05y[Intr03size];
double Intr06x[Intr03size];
double Intr06y[Intr03size];
double Intr07x[Intr07size];
double Intr07y[Intr07size];
double Intr08x[wiresnumber];
double Intr08y[wiresnumber];
int Intr03num=0;
int Intr04num=0;
int Intr05num=0;
int Intr06num=0;


long sumw=0; /*Sum of all the currents*/
double xave=0; /*Average X*/
double xpave=0; /*Average X'*/
double sx2ave=0; /*Average X^2*/
double sxp2ave=0; /*Average X'^2*/
double sxxpave=0; /*Average XX'*/
double x2ave=0; /* <X^2> - <X>^2 */
double xp2ave=0; /* <X'^2> - <X'>^2 */
double xxpave=0; /* <XX'> - <X><X'> */
double emitrms=0; /* sqrt( x2ave * xp2ave - xxpave^2 ) */
double beamsize=0;
double beamdivergence=0;
double twissalpha=0;
double twissbeta=0;
double twissgamma=0;
double emitcalc=0; /*used in calculation*/
double emitmax=0; /*max value of emitcalc*/
double relthreshold=0;
double betarelat=0;
double emitNrms=0;
double emitNmax=0;



/********** READING THE RAWDATA FILE **********/
```

```
    /*Access to the rawfile again*/
    textfile = NULL;
    textfile = fopen(comando,"r");
    if (textfile==NULL)
    {
      strcpy(status,"Cannot find requested file.(2)");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
      return(0);
    }

    nextline(textfile); /*EMITTANCE MEASUREMENT*/
    nextline(textfile); /*Date*/
    nextline(textfile); /*Title*/
    nextline(textfile); /*Source*/
    nextline(textfile); /*Isotope*/

    fscanf(textfile, "%s", inputstring); /*MASS:*/
    readline(textfile, tempstring);
    mass=atoi(tempstring);

    fscanf(textfile, "%s", inputstring); /*VOLTAGE:*/
    readline(textfile, tempstring);
    voltage=atol(tempstring);

    nextline(textfile); /*Puller*/
    nextline(textfile); /*Vacuum1*/
    nextline(textfile); /*Vacuum2*/
    nextline(textfile); /*Heat*/
    nextline(textfile); /*Line*/
    nextline(textfile); /*Magnet*/
    nextline(textfile); /*Oven*/
    nextline(textfile); /*Anode*/

    plane = fgetc(textfile); /*PLANE*/
    nextline(textfile);

    fscanf(textfile, "%s", inputstring); /*DELTA:*/
    deltastep=atol(inputstring);
    nextline(textfile);

    fscanf(textfile, "%s", inputstring); /*INITIAL POSITION:*/
    posinistep=atol(inputstring);
    nextline(textfile);

    fscanf(textfile, "%s", inputstring); /*FINAL POSITION:*/
    posfinstep=atol(inputstring);
    nextline(textfile);

    nextline(textfile); /*Raw data:*/

    for (i=0;i<iterations;i++)
    {
      fscanf(textfile, "%s", inputstring); /*Slit Position*/
      positionsstep[i]=atoi(inputstring);
      for (j=0;j<wiresnumber;j++)
      {
       fscanf(textfile, "%s", inputstring); /*Value*/
       rawdata[i][j]=atoi(inputstring);
      }
    }
```

```
    fclose(textfile);

    /********** CORRECTIONS *************/
    for (i=0;i<iterations;i++)
    {
      rawdata[i][38]=(rawdata[i][37]*3+rawdata[i][40])/4; /*filo strano*/
      rawdata[i][39]=(rawdata[i][37]+rawdata[i][40])/2; /*filo rotto vicino
al salto*/
      rawdata[i][57]=(rawdata[i][56]+rawdata[i][58])/2; /*filo rotto 58; */
    }

    /*********** THRESHOLD ************/
    for (i=0;i<iterations;i++)
      for (j=0;j<wiresnumber;j++)
       if (rawdata[i][j]<threshold)
         rawdata[i][j]=0;

    /***** PRELIMINAR  CALCULATIONS  ********/
    /*Passo dei fili in mrad*/
    if (plane=='X')
    {wiresstepmrad=3.406;}
    else
    {wiresstepmrad=3.401;}
    for (j=0;j<wiresnumber; j++)
    {
      wiresmrad[j]=wiresstepmrad*(j-40+2)/2;
      pxp[j]=0;
    }

    /*Step -> mm conversion */
    for (i=0; i<iterations; i++)
    {
      /*positionsmm[i]=-(positionsstep[i]*0.003125 - 30);*/ /******** ALPI
MODE********/
      positionsmm[i]=-((positionsstep[i] - 615)/25.9 - 70);    /*********
SPES MODE***********/
      px[i]=0;
    }

    /********  CALCULATIONS  **********/
    maxdata=0;
    for (i=0; i<iterations; i++)
    {
      for (j=0; j<wiresnumber; j++)
      {
       /*searches the max value*/
       if (rawdata[i][j]>maxdata)
         maxdata=rawdata[i][j];

       px[i]=px[i]+rawdata[i][j];
       pxp[j]=pxp[j]+rawdata[i][j];

       sumw= sumw+rawdata[i][j];
       xave= xave+rawdata[i][j]*positionsmm[i];
       xpave= xpave+rawdata[i][j]*wiresmrad[j];
       sx2ave= sx2ave+rawdata[i][j]*positionsmm[i]*positionsmm[i];
       sxp2ave= sxp2ave+rawdata[i][j]*wiresmrad[j]*wiresmrad[j];
       sxxpave= sxxpave+rawdata[i][j]*positionsmm[i]*wiresmrad[j];

      }
    }

    if (sumw<=0)
    {
      strcpy(status,"Data Error. SumW < 0 .");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
```

```
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
      return(0);
    }

    xave = xave / sumw;
    xpave=xpave / sumw;
    sx2ave=sx2ave / sumw;
    sxp2ave=sxp2ave / sumw;
    sxxpave=sxxpave / sumw;
    x2ave=sx2ave - (xave * xave);
    xp2ave=sxp2ave - (xpave * xpave);
    xxpave=sxxpave - (xave * xpave);
    emitrms = sqrt(x2ave * xp2ave - xxpave * xxpave);

    beamsize = sqrt(x2ave);
    beamdivergence = sqrt(xp2ave);

    twissalpha = -xxpave / emitrms;
    twissbeta = x2ave / emitrms;
    twissgamma = xp2ave / emitrms;

    for (i=0; i<iterations; i++)
    {
      for (j=0; j<wiresnumber; j++)
      {
       /*calculates emittance and searches for its max value*/
       if (rawdata[i][j]>0)
       {
         emitcalc = twissgamma * pow((positionsmm[i] - xave),2) + 2 *
twissalpha * (positionsmm[i] - xave) * (wiresmrad[j] - xpave) + twissbeta *
pow((wiresmrad[j]-xpave),2);
        if (emitcalc > emitmax)
          emitmax = emitcalc;
       }
      }
    }

    relthreshold = threshold; /*typecasting necessary for the division in
the next line*/
    relthreshold = relthreshold / maxdata;
    /*printf("emitmax = %f\n",emitmax);
    printf("rel threshold = %f\n",relthreshold);*/

    betarelat=voltage/1000;
    betarelat=(betarelat/mass)/931494;
    betarelat=pow(1+betarelat,2);
    betarelat=sqrt(1-1/betarelat);
    printf("betarelat = %f\n",betarelat);

    emitNrms = betarelat * emitrms;
    emitNmax = betarelat * emitmax;


    /******** CREATE OUTPUT FILE **************/
    /* Open rawdata file again*/
    textfile = fopen(comando,"r");
    if (textfile==NULL)
    {
      strcpy(status,"Cannot find requested file.(2)");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
```

```
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
      return(0);
    }

    /* Create emittancedata file */
    strcpy(comando,path);
    strcat(comando,file2);
    FILE *outputfile = NULL;
    outputfile = fopen(comando,"w");
    if (outputfile==NULL)
    {
      strcpy (status,"Cannot create output file.");
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
      return(0);
    }

    readline(textfile,inputstring); /*EMITTANCE MEASUREMENT*/
    fprintf(outputfile,"%s OUTPUT\n",inputstring);
    readline(textfile,inputstring); /*DATE*/
    fprintf(outputfile,"%s\n",inputstring);

    /*Write date to interface PV*/
    stat=ca_create_channel (outdata1,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_STRING,pCh,&inputstring);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);


    for (i=0;i<14;i++) /*TITLE, SOURCE, ..... , ANODE, PLANE*/
    {
      readline(textfile,inputstring);
      fprintf(outputfile,"%s\n",inputstring);
    }

    fprintf(outputfile, "INITIAL POSITION: %f mm\n",positionsmm[0]);
    fprintf(outputfile, "FINAL POSITION: %f mm\n",positionsmm[iterations-
1]);
    fprintf(outputfile, "ITERATIONS: %li\n",iterations);
    fprintf(outputfile, "THRESHOLD: %li\n",threshold);
    fprintf(outputfile, "\n");
    fprintf(outputfile, "MAX DATA: %i\n",maxdata);
    fprintf(outputfile, "Data integral: %li\n",sumw);
    fprintf(outputfile, "CENTER POSITION: %f mm\n",xave);
    fprintf(outputfile, "CENTER MOMENTUN: %f mrad\n",xpave);
    fprintf(outputfile, "Average pos^2: %f mm^2\n",x2ave);
    fprintf(outputfile, "Average mom^2: %f mrad^2\n",xp2ave);
    fprintf(outputfile, "Average pos*mom: %f mm*mrad\n",xxpave);
    fprintf(outputfile, "EMITTANCE RMS: %f mm*mrad\n",emitrms);
    fprintf(outputfile, "BEAM SIZE: %f mm\n",beamsize);
    fprintf(outputfile, "BEAM DIVERGENCE: %f mrad\n",beamdivergence);
    fprintf(outputfile, "TWISS ALPHA: %f\n",twissalpha);
    fprintf(outputfile, "TWISS BETA: %f mm/mrad\n",twissbeta);
    fprintf(outputfile, "TWISS GAMMA: %f mrad/mm\n",twissgamma);
    fprintf(outputfile, "BETA RELAT.: %f \n",betarelat);
    fprintf(outputfile, "EMITTANCE N RMS: %f  mm*mrad\n",emitNrms);
    fprintf(outputfile, "EMITTANCE MAX: %f  mm*mrad\n",emitmax);
    fprintf(outputfile, "EMITTANCE N MAX: %f  mm*mrad\n",emitNmax);
    fprintf(outputfile, "Rel. threshold: %f %%\n",relthreshold*100);
```

```
    fprintf(outputfile, "Ratio Max/Rms: %f\n",(emitmax/emitrms));

    /*Write datas to interface PVs*/
    stat=ca_create_channel (outdata2,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&xave);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata3,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&xpave);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata4,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&beamsize);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata5,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&beamdivergence);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata6,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&twissalpha);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata7,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&twissbeta);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata8,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&twissgamma);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata9,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&emitrms);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata10,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&betarelat);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata11,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_DOUBLE,pCh,&emitNrms);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (outdata12,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_put(DBR_SHORT,pCh,&maxdata);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

    fclose(textfile);
    fclose(outputfile);

    /*********** CREATE GRAPHICS ************/
    for (i=0; i<iterations; i++)
    {
      for (j=0; j<wiresnumber; j++)
      {
```

```
      if ((rawdata[i][j]>0) && (rawdata[i][j]<=(maxdata/4)))
      {
        Intr03x[Intr03num]=positionsmm[i];
        Intr03y[Intr03num]=wiresmrad[j];
        Intr03num++;
      }
      else if ((rawdata[i][j]>(maxdata/4)) && (rawdata[i][j]<=(maxdata/2)))
      {
        Intr04x[Intr04num]=positionsmm[i];
        Intr04y[Intr04num]=wiresmrad[j];
        Intr04num++;
      }
      else if ((rawdata[i][j]>(maxdata/2)) &&
(rawdata[i][j]<=(3*maxdata/4)))
      {
        Intr05x[Intr05num]=positionsmm[i];
        Intr05y[Intr05num]=wiresmrad[j];
        Intr05num++;
      }
      else if (rawdata[i][j]>(3*maxdata/4))
      {
        Intr06x[Intr06num]=positionsmm[i];
        Intr06y[Intr06num]=wiresmrad[j];
        Intr06num++;
      }

     }
   }

   /* Ellipsis data */
   double deltang=2 * M_PI / 30;
   double ang=0;
   for (i=0;i<31;i++)
   {
     Intr01x[i]=sqrt(emitrms * twissbeta) * cos(ang) + xave;
     Intr01y[i]=sqrt(emitrms / twissbeta) * (sin(ang) - twissalpha *
cos(ang)) + xpave;
     Intr02x[i]=sqrt(emitmax * twissbeta) * cos(ang) + xave;
     Intr02y[i]=sqrt(emitmax / twissbeta) * (sin(ang) - twissalpha *
cos(ang)) + xpave;
     ang = ang + deltang;
   }

   if ((Intr03num>Intr03size)|| (Intr04num>Intr03size)||
(Intr05num>Intr03size)|| (Intr06num>Intr03size))
   {
     strcpy(status,"Buffer Error. Too much graphic data.");
     stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
     stat=ca_pend_io(timeout);
     stat=ca_put(DBR_STRING,pCh,&status);
     stat=ca_pend_io(timeout);
     stat=ca_clear_channel(pCh);
     ca_context_destroy();
     return(0);
   }
   if (iterations>Intr07size)
   {
     strcpy (status,"Buffer Error. Too many iterations.");
     stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
     stat=ca_pend_io(timeout);
     stat=ca_put(DBR_STRING,pCh,&status);
     stat=ca_pend_io(timeout);
     stat=ca_clear_channel(pCh);
     ca_context_destroy();
     return(0);
```

```
    }

    /*Fill graphic arrays*/
    for (i=Intr03num;i<Intr03size;i++)
    {
      Intr03x[i]=Intr03x[Intr03num-1];
      Intr03y[i]=Intr03y[Intr03num-1];
    }
    for (i=Intr04num;i<Intr03size;i++)
    {
      Intr04x[i]=Intr04x[Intr04num-1];
      Intr04y[i]=Intr04y[Intr04num-1];
    }
    for (i=Intr05num;i<Intr03size;i++)
    {
      Intr05x[i]=Intr05x[Intr05num-1];
      Intr05y[i]=Intr05y[Intr05num-1];
    }
    for (i=Intr06num;i<Intr03size;i++)
    {
      Intr06x[i]=Intr06x[Intr06num-1];
      Intr06y[i]=Intr06y[Intr06num-1];
    }
    for (i=0;i<iterations;i++)
    {
      Intr07x[i]=positionsmm[i];
      Intr07y[i]=px[i];
    }
    for (i=iterations;i<Intr07size;i++)
    {
      Intr07x[i]=Intr07x[iterations-1];
      Intr07y[i]=Intr07y[iterations-1];
    }
    for (i=0;i<wiresnumber;i++)
    {
      Intr08x[i]=pxp[i];
      Intr08y[i]=wiresmrad[i];
    }

    /*Write graphic arrays to PVs*/
    stat=ca_create_channel (profile1,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr01size,pCh,&Intr01x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile2,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr01size,pCh,&Intr01y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile3,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr01size,pCh,&Intr02x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile4,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr01size,pCh,&Intr02y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile5,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr03x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile6,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
```

```
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr03y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile7,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr04x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile8,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr04y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile9,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr05x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile10,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr05y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile11,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr06x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile12,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr03size,pCh,&Intr06y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile13,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr07size,pCh,&Intr07x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile14,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,Intr07size,pCh,&Intr07y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile15,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,wiresnumber,pCh,&Intr08x);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);
    stat=ca_create_channel (profile16,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
    stat=ca_pend_io(timeout);
    stat=ca_array_put(DBR_DOUBLE,wiresnumber,pCh,&Intr08y);
    stat=ca_pend_io(timeout);
    stat=ca_clear_channel(pCh);

    ca_context_destroy();

    return(0);
}
```

File calculation.h

```
char datetime1Set[] = "TaDiagEmit01_DateSt01";
char datetime2Set[] = "TaDiagEmit01_DateSt02";
char datetime3Set[] = "TaDiagEmit01_DateSt03";
```

```
char datetime4Set[] = "TaDiagEmit01_DateSt04";
char datetime5Set[] = "TaDiagEmit01_DateSt05";
char thresholdSet[] = "TaDiagEmit01_ThreSt";
char status2Set[] = "TaDiagEmit01_StatSt02";

char folder1[] = "Data";
char file1[] = "rawdata";
char file2[] = "emittancedata";
char file3[] = "graphdata";

int wiresnumber = 80;

/*Output Graph Data*/
char profile1[] = "TaDiagEmit01_Intr01x";
char profile2[] = "TaDiagEmit01_Intr01y";
char profile3[] = "TaDiagEmit01_Intr02x";
char profile4[] = "TaDiagEmit01_Intr02y";
char profile5[] = "TaDiagEmit01_Intr03x";
char profile6[] = "TaDiagEmit01_Intr03y";
char profile7[] = "TaDiagEmit01_Intr04x";
char profile8[] = "TaDiagEmit01_Intr04y";
char profile9[] = "TaDiagEmit01_Intr05x";
char profile10[] = "TaDiagEmit01_Intr05y";
char profile11[] = "TaDiagEmit01_Intr06x";
char profile12[] = "TaDiagEmit01_Intr06y";
char profile13[] = "TaDiagEmit01_Intr07x";
char profile14[] = "TaDiagEmit01_Intr07y";
char profile15[] = "TaDiagEmit01_Intr08x";
char profile16[] = "TaDiagEmit01_Intr08y";
int Intr01size=31;
int Intr03size=2000;
int Intr07size=500;

char outdata1[] = "TaDiagEmit01:IntrRd01";
char outdata2[] = "TaDiagEmit01:IntrRd02";
char outdata3[] = "TaDiagEmit01:IntrRd03";
char outdata4[] = "TaDiagEmit01:IntrRd04";
char outdata5[] = "TaDiagEmit01:IntrRd05";
char outdata6[] = "TaDiagEmit01:IntrRd06";
char outdata7[] = "TaDiagEmit01:IntrRd07";
char outdata8[] = "TaDiagEmit01:IntrRd08";
char outdata9[] = "TaDiagEmit01:IntrRd09";
char outdata10[] = "TaDiagEmit01:IntrRd10";
char outdata11[] = "TaDiagEmit01:IntrRd11";
char outdata12[] = "TaDiagEmit01:IntrRd12";
```

File readfilefunctions.c

```
void itoa(int n, int len, char *buf)
{
  int i;
  char sign = ' ';
  if (0 > n) {
    n = -n;
    sign = '-';
  }
  for (i = len; i > 0; i--) {
    if (n > 0 || len == i) {
      buf[i-1] = '0' + (n % 10);
      n = n / 10;
    } else {
      buf[i-1] = sign;
      sign = ' ';
    }
```

```
  }
  if (strlen(buf)>len)
    buf[len]='\0';
}


void launchlineerror(int i)
{
      strcpy(status,"Error reading file, line:");
      itoa(i,5,tempstring);
      strcat(status,tempstring);
      stat=ca_create_channel
(status2Set,NULL,NULL,CA_PRIORITY_DEFAULT,&pCh);
      stat=ca_pend_io(timeout);
      stat=ca_put(DBR_STRING,pCh,&status);
      stat=ca_pend_io(timeout);
      stat=ca_clear_channel(pCh);
      ca_context_destroy();
}

void nextline(FILE *rdf)
{
  int c;
  do
  {c=fgetc(rdf);}
  while (c != '\n');
}

void readline(FILE *rdf, char *buf)
{
  int c;
  int i=0;
  c=fgetc(rdf);
  while (c != '\n')
    {buf[i]=c;
      i++;
      c=fgetc(rdf);
    }
  buf[i]='\0';
}
```

Comandi per la compilazione

```
/usr/bin/gcc -c   -D_POSIX_C_SOURCE=199506L -D_POSIX_THREADS -
D_XOPEN_SOURCE=500           -D_X86_  -DUNIX  -D_BSD_SOURCE -Dlinux  -
D_REENTRANT -ansi   -O3   -Wall      -m32     -g  -I. -I/home/confort/base-
3.14.10/include/os/Linux -I/home/confort/base-3.14.10/include
acquisition.c
/usr/bin/g++ -o acquisition  -L/home/confort/base-3.14.10/lib/linux-x86 -
Wl,-rpath,/home/confort/base-3.14.10/lib/linux-x86       -m32
acquisition.o    -lcas -lgdd -lasHost -ldbStaticHost -lregistryIoc -lca -
lCom

/usr/bin/gcc -c   -D_POSIX_C_SOURCE=199506L -D_POSIX_THREADS -
D_XOPEN_SOURCE=500           -D_X86_  -DUNIX  -D_BSD_SOURCE -Dlinux  -
D_REENTRANT -ansi   -O3   -Wall      -m32     -g  -I. -I/home/confort/base-
3.14.10/include/os/Linux -I/home/confort/base-3.14.10/include
calculation.c
/usr/bin/g++ -o calculation  -L/home/confort/base-3.14.10/lib/linux-x86 -
Wl,-rpath,/home/confort/base-3.14.10/lib/linux-x86       -m32
calculation.o    -lcas -lgdd -lasHost -ldbStaticHost -lregistryIoc -lca -
lCom
```

Bibliografia e Sitografia

[1] http://www.sapere.it/tca/MainApp?srvc=vr&url=/5/1012_1

[2] www.infn.it/comunicazione/mostre/pdf%20schede%20bassa/scheda4.pdf

[3] Mirko Tessarolo - Sviluppo di un sistema per la movimentazione di sorgenti radioattive – Tesi di laurea in Ingegneria Meccatronica, A.A. 2008/2009, Università degli studi di Trento

[4] Mirlo Libralato - Studio elettro-termo-strutturale del sistema di estrazione e ionizzazione del progetto SPES – Tesi di laurea in Ingegneria Meccanica, A.A. 2008/2009, Università degli studi di Padova

[5] www.eurisol.org/site02/doc/brochureEurisol_100709_italian.pdf

[6] Maurizio Montis - Utilizzo di software EPICS per lo sviluppo del sistema di controllo dell'apparato di produzione di ioni del progetto SPES – Tesi di laurea in Ingegneria dell'Automazione, A.A. 2009/2010, Università degli studi di Padova

[7] http://www.aps.anl.gov/epics

[8] *APS Training*: http://www.aps.anl.gov/epics/docs/GSWE.php

[9] *Channel Access Reference Manual*: http://www.aps.anl.gov/epics/base/R3-14/10-docs/CAref.html

[10] *Record Reference Manual*: http://www.aps.anl.gov/epics/wiki/index.php/RRM_3-14

[11] *Application Developer's Guide*: http://www.aps.anl.gov/epics/base/R3-14/10-docs/AppDevGuide.pdf

[12] http://css.desy.de/content/index_eng.html

[13] http://en.wikipedia.org/wiki/Faraday_cup

[14] Lecture Notes on Beam Instrumentation and Diagnostics, Peter Forck. Joint University Accelerator SchoolC, 2009.

[15] Introduction to linear accelerators, Thomas P. Wangler. Los. Alamos Report, LA-UR-94-125 (1994)

[16] Emittance Meter design details for the SPES project, Jacobo Montaño. http://www.lnl.infn.it/~spes/

[17] http://en.wikipedia.org/wiki/VMEbus

[18] The VMEbus Handbook, expanded third edition, Wade D. Peterson. VITA, 1993.

[19] http://www.vita.com

[20] MVME3100 Programmer's Guide

[21] XVME-566 Manual, Xycom

[22] XVME-240 Manual, Xycom

[23] XVME-220 Manual, Xycom

[24] http://en.wikipedia.org/wiki/VxWorks

[25] VxWorks 5.4 Programmer's Guide, WindRiver, 1999.

[26] VxWorks 5.4 Reference Manual, WindRiver, 1999.

[27] *XycomIoc Repository*: http://epics.hg.sourceforge.net/hgweb/epics/xycomioc/summary

[28] *NTP documentation*: http://www.ntp.org/documentation.html

[29] http://en.wikiversity.org/wiki/Category:C_programming_language

[30] http://www.cplusplus.com/reference/

[31] http://programmazione.html.it/guide/lezione/1116/le-funzioni-fprintf-e-fscanf/