

UNIVERSITÀ DEGLI STUDI DI ROMA

La Sapienza

Dipartimento di Fisica

Tesi di Laurea

**Il filtro in energia di primo livello
dell'esperimento L3 :
caratterizzazione dei moduli e
prove di blocchi funzionali**

Relatore:

Prof. Federico Cesaroni

Laureando:

Paolo Pacchiarotti mat. N43827

Tesina 1:

Teoria del moto browniano

Relatore: *Dott. Angelo Vulpiani*

Tesina 2:

**Principio di funzionamento di un refrigeratore
a diluizione di $He^3 - He^4$**

Relatore: *Dott. Fulvio Ricci*

Indice

Introduzione	iii
1 L'esperimento L3	1
1.1 LEP	1
1.2 L'esperimento L3	2
1.3 Il rivelatore	4
2 Il sistema di acquisizione dati di L3	13
2.1 Introduzione	13
2.2 Il filtro di primo livello di L3	20
2.3 Il filtro di secondo livello	21
2.4 Il filtro di terzo livello e il sistema di acquisizione dati	24
3 Il fitro in energia di primo livello	27
3.1 Segnali per il filtro in energia	27
3.2 Condizioni di "trigger"	33
3.3 Struttura generale del filtro	39
3.3.1 La conversione dei segnali	40
3.3.2 L'elaborazione numerica	42
3.3.3 Le temporizzazioni	45
3.4 Lettura dei Fera	48
3.5 Schema a blocchi della parte di elaborazione numerica	53
3.5.1 Filtro in energia	59
3.5.2 Filtro di fotone singolo	61
3.5.3 Filtro di <i>clusters</i>	63
3.5.4 Conteggio delle tracce ad alta energia	64
3.5.5 Filtro di luminosità	65
3.5.6 Filtro di "overflow"	65
3.5.7 Dati inviati al filtro di secondo livello	66
3.5.8 La decisione finale	66
3.6 Temporizzazione del filtro	67
4 Moduli e problemi connessi ai sincronismi	74

4.1 Premessa	74
4.2 Il modulo	74
4.3 Definizione di τ_1 e τ_2	78
4.4 Misura di τ_1 e τ_2	84
5 Le prove dei moduli	89
5.1 Introduzione	89
5.2 I programmi	90
5.3 L' <i>hardware</i> delle prove	95
5.4 Le prove sui Fera	96
5.4.1 Prove sulla linearit�	97
5.4.2 Prove sulla linearit� integrale	99
5.4.3 Prove dei piedistalli	99
6 Le prove dei blocchi funzionali	104
6.1 I blocchi funzionali	104
6.2 Le prove in laboratorio	105
6.2.1 Prova del gruppo 1 di conversione	105
6.2.2 Calcolo del vettore ELM $R\phi$	111
6.2.4 Filtro di <i>cluster</i> nel piano $R\phi$	116
7 Conclusioni	123
Bibliografia	125
Appendice A: Specifiche tecniche dei moduli	128
Appendice B: Listati dei Programmi	164
B1: Documentazione dei programmi di prova dei moduli	165
B2: Programmi di prova dei moduli	177
B3: Programmi di prova dei blocchi funzionali	230

Introduzione

La complessità attualmente raggiunta dai moderni apparati di misura negli esperimenti di fisica delle alte energie richiede lo sviluppo di sistemi di acquisizione dati altamente sofisticati.

Tradizionalmente questi sistemi avevano il solo compito di memorizzare adeguatamente l'insieme dei dati provenienti dal rivelatore ogniqualvolta si fosse individuata l'avvenuta interazione.

Questa semplice strategia diventa inapplicabile allorché la quantità di informazioni prodotta da un evento è tale, se ripetuta ad una frequenza elevata, da saturare le capacità di memorizzazione del più veloce sistema di calcolo. Diventa quindi necessario che il sistema di acquisizione svolga in tempo reale una funzione di riconoscimento delle "signature" che caratterizzino gli eventi interessanti dal punto di vista fisico, e soltanto per questi proceda alla registrazione delle informazioni.

Argomento della presente tesi è l'analisi del "filtro" (trigger) di energia al primo livello del sistema di acquisizione dati dell'esperimento L3. Tale esperimento è in fase di avanzato allestimento sull'anello di accumulazione per elettroni e positroni LEP.

Nel capitolo 1 vengono brevemente descritti il progetto LEP, l'esperimento L3 e le relative possibilità di ricerca nella fisica delle alte energie, mentre nel capitolo 2 viene illustrato il sistema di acquisizione dati e l'apparato di

filtraggio degli eventi in tale esperimento.

Il progetto e la descrizione del “filtro in energia di primo livello”, realizzato mediante dei moduli CAMAC (circa 300), sono l’argomento del terzo capitolo; le problematiche sorte nell’utilizzo di questi moduli e la caratterizzazione del loro comportamento nel tempo con opportuni parametri misurati in laboratorio sono affrontate nell’ambito del quarto capitolo. Per il controllo del funzionamento vari moduli, é stato sviluppato un pacchetto di software la cui struttura é discussa nel capitolo 5.

Infine nel sesto capitolo viene illustrata la realizzazione pratica e la prova in laboratorio di alcune parti del filtro in energia per verificarne la funzionalità.

1 L'esperimento L3

1.1 LEP

Il progetto LEP (Large Electron Positron storage ring), in fase di avanzata realizzazione al CERN (Organizzazione Europea per la Ricerca Nucleare) di Ginevra, consiste nella realizzazione di un anello di accumulazione nel quale pacchetti (bunches) di elettroni (e^-) e positroni (e^+) saranno accelerati e fatti collidere ad energie mai raggiunte finora in questo tipo di macchine. Sono previste due fasi di funzionamento della macchina LEP: durante la prima, che dovrebbe partire all'inizio del 1989, l'energia nel centro di massa raggiungerà i 100 GeV e con $4 + 4$ pacchetti di e^+ ed e^- , mentre nella seconda, detta LEP200, tale energia dovrebbe raggiungere i 200 GeV con $8 + 8$ pacchetti. Per minimizzare l'energia persa per radiazione di sincrotrone dagli elettroni e positroni, è stato necessario costruire un anello il cui raggio fosse sufficientemente grande. Infatti l'energia persa per emissione di sincrotrone va come [26]:

$$\left(\frac{E}{M_0}\right)^4 \frac{1}{R}$$

dove E ed M_0 sono rispettivamente energia e massa a riposo della particella ed R il raggio dell'anello di accelerazione. È quindi chiaro che solo un grosso acceleratore, con una circonferenza come LEP di ~ 27 Km, permette alte energie per particelle così poco massive come gli elettroni. D'altro canto però il fatto di utilizzare e^+ ed e^- consente di avere eventi molto più puliti rispetto

a quelli ottenuti con analoghe macchine protone anti-protone ($p\bar{p}$) perchè gli elettroni non presentano (fino ad ora non è stata osservata) struttura interna. Gli esperimenti su tale macchina dovrebbero permettere un'ulteriore verifica del "Modello Standard" di Weinberg e Salam dopo la recente scoperta dei bosoni Z^0 e $W^\pm$ all' SPS nonchè lo studio di fenomeni nuovi che potrebbero presentarsi.

1.2 L'esperimento L3

Sull' acceleratore LEP sono previste, nella prima fase, quattro sale sperimentali per altrettanti esperimenti : ALEPH, DELPHI, L3, OPAL.

L'esperimento L3 differisce dagli altri per alcune particolari peculiarità del rivelatore : infatti nel progetto dell'apparato è stata dedicata una speciale attenzione alla rivelazione di elettroni, fotoni e muoni prodotti su tutto l'angolo solido [15]. Nella prima fase di accensione di LEP il rivelatore permetterà di indagare in modo particolare, oltre che su eventuali processi del tutto imprevisti, sui seguenti fenomeni fisici [11,17] :

- i) Ricerca del bosone di Higgs
- ii) Spettroscopia del toponio
- iii) Conteggio del numero di generazioni di neutrini
- iv) Studio della Z^0

Diamo ora una breve e sommaria descrizione di questi processi anche se ciò esula dagli intenti di questo lavoro.

i) **Ricerca del bosone di Higgs (H^0).** Si cercano eventi del tipo :

$$Z^0 \rightarrow \gamma + H^0 \rightarrow \gamma + jets$$

Per una massa dell' Higgs $M_{H^0} \simeq 50$ GeV, tale evento è caratterizzato da un fotone di ~ 40 GeV e 2 fasci (jets) di adroni. Il bosone scalare neutro di Higgs, da cui discenderebbero le masse di tutte le particelle, è previsto dal meccanismo della "Rottura spontanea della simmetria".

ii) **Spettroscopia del toponio.** Il "modello standard" prevede l'esistenza di 6 quarks; mentre 5 di essi sono stati osservati come componenti di adroni, per il sesto, il "top" (t), non esistono ancora risultati sperimentali certi. Lo studio del decadimento dello stadio legato $t\bar{t}$ (toponio), forse possibile a LEP, dovrebbe fornire una nuova verifica della teoria. Uno dei probabili processi possibili di questo decadimento è :

$$e^+e^- \rightarrow (t\bar{t})' \rightarrow \gamma_1 + \gamma_2 + l^+l^-$$

Dato che $E_{\gamma_1} = 100$ MeV e $E_{\gamma_2} = 700$ MeV, per poter distinguere l'evento dal fondo c'è bisogno di un calorimetro elettromagnetico la cui risoluzione consenta, già a 100 MeV, di misurare l'energia dei fotoni con buona precisione. Ovviamente si deve anche rivelare la coppia di leptoni prodotta (l^+l^-). Essa può essere una coppia $\mu^+\mu^-$ o e^+e^- che possono essere rivelate rispettivamente con camere a muoni e camera di vertice immerse in campo magnetico.

- iii) **Conteggio del numero di generazioni di neutrini.** Consiste nel misurare la sezione d'urto del processo :

$$e^+e^- \rightarrow \gamma + Z^0 \rightarrow \gamma + \nu\bar{\nu}$$

che e' direttamente proporzionale al numero di generazioni di neutrini (N_ν). Tale sezione d'urto varia col variare dell'energia del γ ed è grande per energie piccole del fotone (dell'ordine di qualche GeV). Quindi per la buona riuscita dell'esperimento è necessario un calorimetro elettromagnetico che consenta di poter misurare l'energia del γ fino ad energie relativamente basse.

- iv) **Studio della Z^0 .** Consiste nella misura precisa della massa della Z^0 (allo 0.1 %) e nello studio della larghezza totale del suo decadimento ($\Gamma(Z^0)$) da cui si deduce il numero di famiglie di fermioni con massa minore di metà di quella della Z^0 . Infatti il modello Standard prevede $\Gamma(Z^0) \simeq 2.9$ GeV se esistono 6 leptoni e 6 quarks, mentre gli attuali risultati sperimentali danno per essa solo il limite superiore di 8.5 GeV. Si prevede che gli esperimenti previsti su LEP forniranno il suo valore con un errore di 50 MeV [17].

1.3 Il rivelatore

Nel progetto del rivelatore di L3 [15] si è cercato di ottimizzare la misura dell'energia di e e γ e dell'impulso dei μ , cosa fondamentale per lo studio dei processi fisici precedentemente menzionati.

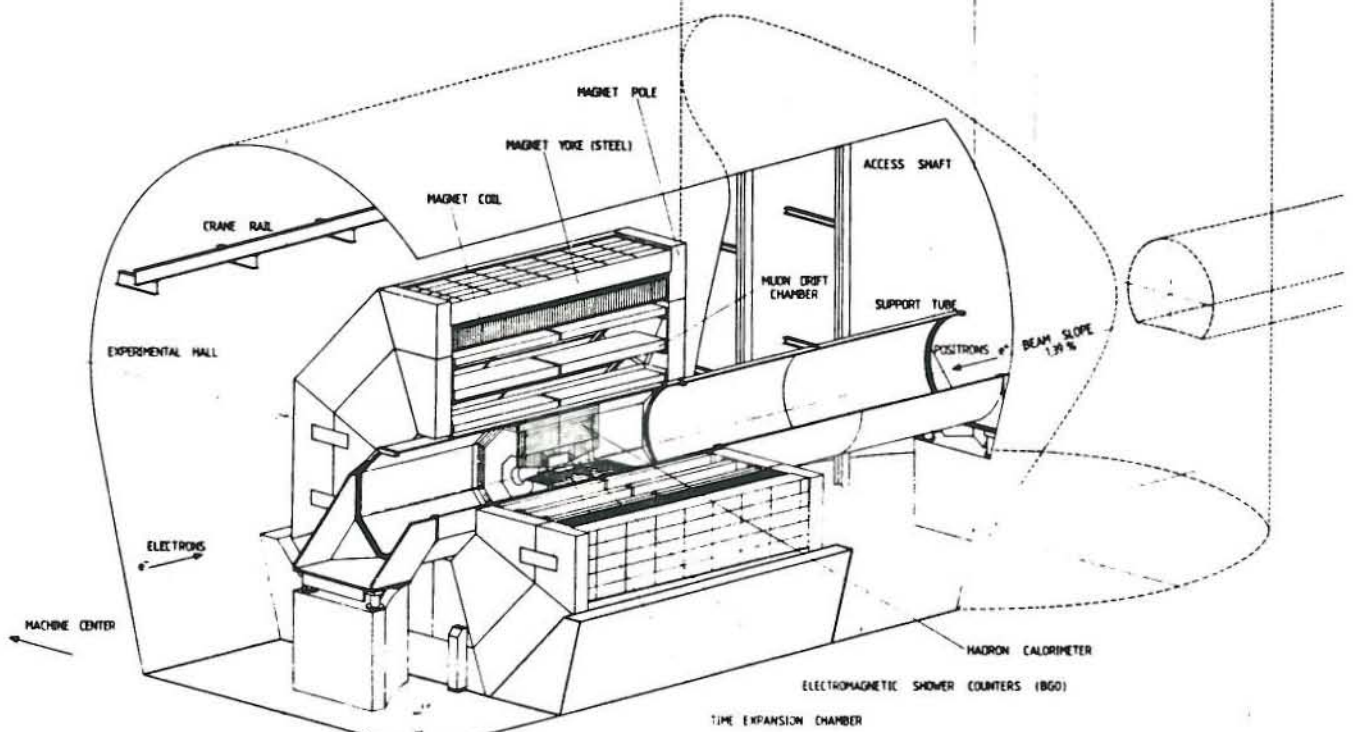


Fig. 1.1 il rivelatore di L3

Per avere una buona risoluzione, l'energia di e e γ deve essere misurata il più vicino possibile al vertice di interazione affinché non si degradi nell'attraversare i materiali che costituiscono il detector. Inoltre per ottenere una buona precisione nella misura dell'impulso dei μ c'è bisogno di un adatto rivelatore di grande volume immerso in campo magnetico. Il rivelatore dell'esperimento L3 è quindi costituito da :

- Una piccola camera di vertice con una alta risoluzione delle tracce.
- Calorimetri, ovvero le parti dedicate alla misura dell'energia delle particelle, molto compatti e posti subito intorno al vertice di interazione.
- Rivelatore di grande volume con all'interno vari piani di camere a deriva per poter misurare con precisione l'impulso dei muoni.

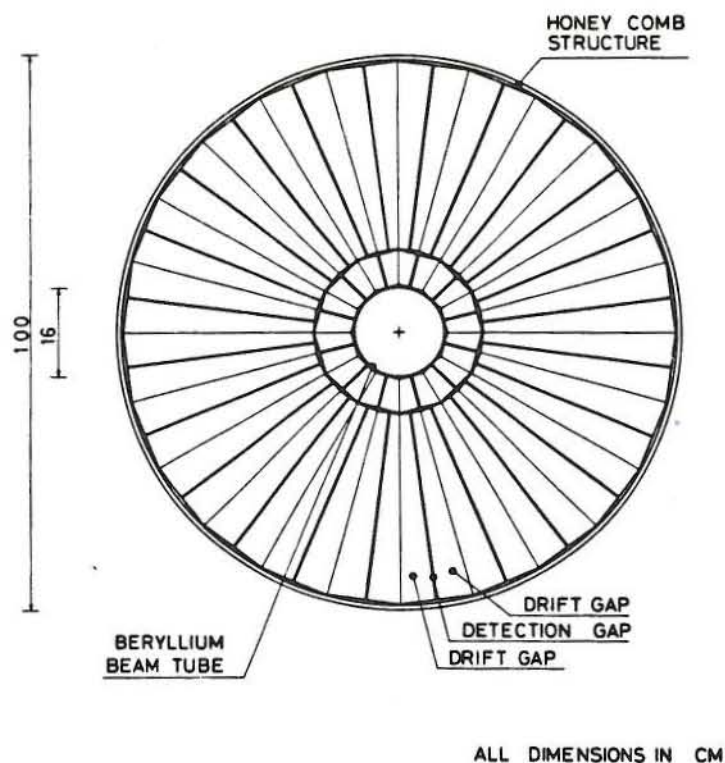


Fig. 1.2 La TEC

- Una coppia di rivelatori di luminosità per controllare con precisione la luminosità dell'acceleratore.

L'intero rivelatore è immerso in un campo magnetico solenoidale con l'asse coincidente con la direzione del fascio. In figura 1.1 è mostrata una visione del progetto del rivelatore.

Il rivelatore di vertice. Il rivelatore più interno è la camera di vertice il cui compito è di rivelare le tracce di particelle cariche vicino al punto di interazione. In questa maniera si possono distinguere i fotoni dagli elettroni, misurare la carica di quest'ultimi e stabilire la vita media, dell'ordine di 10^{-13} secondi, delle particelle di lunga vita. Nell'esperimento L3 essa consiste nella TEC, Time Expansion Chamber (fig. 1.2), una particolare cam-

era a migrazione cilindrica con risoluzione delle tracce di $30\ \mu\text{m}$. È detta così perchè sfrutta l'idea di utilizzare 2 campi elettrici, uno debole in cui le cariche, prodotte dal passaggio di particelle ionizzanti, migrano a bassa velocità (Time Expansion), ed uno di *detection* dove le stesse cariche vengono accelerate e focalizzate su di un anodo e quindi rivelate.

Il calorimetro elettromagnetico. Subito intorno alla TEC si estende, da 50 cm fino ad 80 cm di raggio, il calorimetro elettromagnetico (ELM) formato da 12000 cristalli di BGO (Ortogerminato di bismuto, $\text{Bi}_4\text{Ge}_3\text{O}_{12}$). La sua forma è quella di un cilindro con due "tappi" agli estremi (fig. 1.3) e con l'asse diretto lungo la direzione del fascio. Il cilindro è formato da 8000 cristalli a forma di tronco di piramide a sezione quadrata con l'asse diretto verso il vertice di interazione. I due "tappi" sono formati da 2000 cristalli ciascuno: anche in questo caso sono dei tronchi di piramide diretti verso il vertice di interazione ma hanno sezione rettangolare. I cristalli sono tutti lunghi 24 cm; quelli che formano il cilindro hanno sezioni variabili da $2 \times 2\ \text{cm}^2$ a $3 \times 3\ \text{cm}^2$. Queste dimensioni trasversali permettono una elevata frammentazione del calorimetro elettromagnetico e quindi un'ottima risoluzione spaziale.

I cristalli di BGO per il fatto di essere intrinsecamente scintillanti e di avere una breve lunghezza di radiazione (1.13 cm) sono particolarmente adatti alla realizzazione di calorimetri di ottime prestazioni e molto compatti, come richiesto dall'esperimento L3. Infatti [15] la risoluzione energetica di

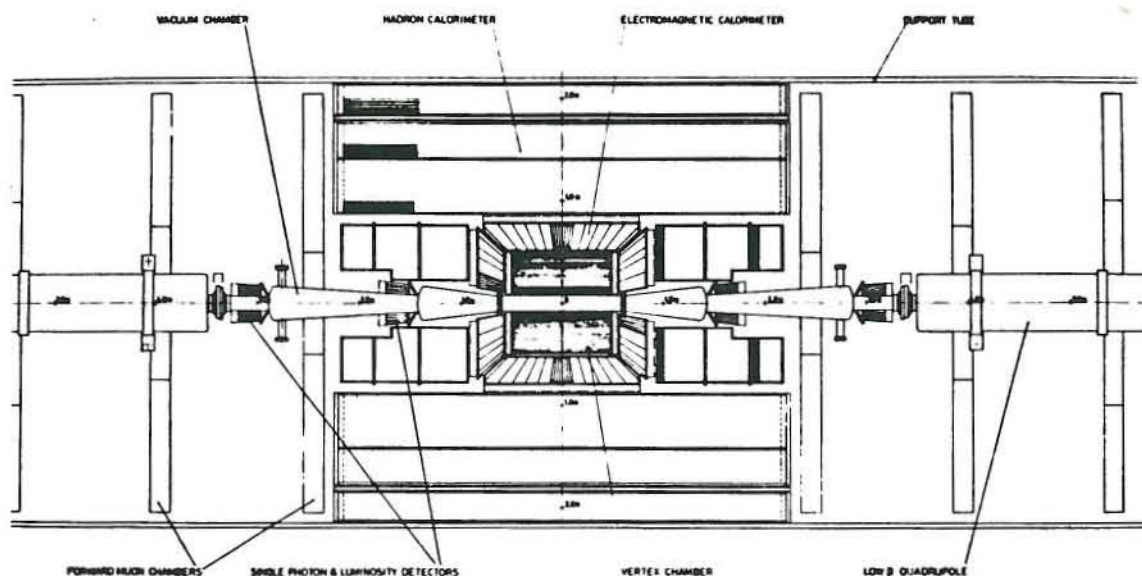


Fig. 1.3 I rivelatori interni al tubo di supporto

un cristallo di BGO è

$$\frac{\sigma}{E} = \frac{0.5\%}{\sqrt{E}} \text{ per } E < 1 \text{ GeV}$$

mentre è praticamente costante e pari all'1% per $1 \text{ GeV} \leq E \leq 50 \text{ GeV}$ se si sceglie un'opportuna lunghezza del cristallo. In figura 1.4 è mostrato come questo sia possibile con un cristallo della lunghezza di 24 cm pari a 22 lunghezze di radiazione.

Il calorimetro adronico. Il calorimetro adronico si estende fino a 213 cm dall'asse del rivelatore e permette di misurare l'energia e la posizione degli sciame adronici. È stato realizzato con una struttura cellulare detta a torri (tronchi di piramide); ogni torre ha l'asse diretto verso il vertice di interazione (v. fig. 1.5) ed ha approssimativamente la dimensione di uno

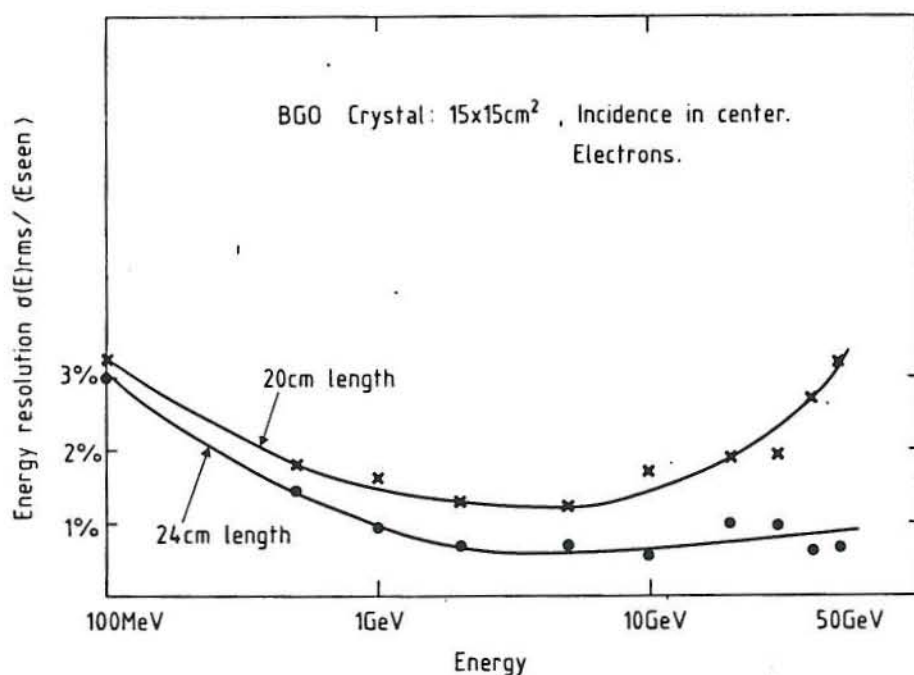


Fig. 1.4 Risoluzione energetica in funzione della lunghezza del cristallo

sciame adronico. La torre viene costruita alternando trasversalmente strati passivi di uranio, dove si sviluppa lo sciame, a strati attivi di camere a fili in regime proporzionale. La risoluzione energetica prevista per questo tipo di calorimetro è [15]:

$$\frac{\sigma}{E} = \frac{50\%}{\sqrt{E}} + 10\% \quad E \text{ in GeV}$$

Come per il rivelatore elettromagnetico l'insieme di queste torri, assume la forma di un cilindro con due "tappi" il cui asse coincide con la direzione del fascio.

Per una migliore installazione e manutenzione, i calorimetri e la camera di vertice sono contenuti in un tubo di supporto, che permette eventualmente di "sfilare" i rivelatori centrali dal resto del rivelatore.

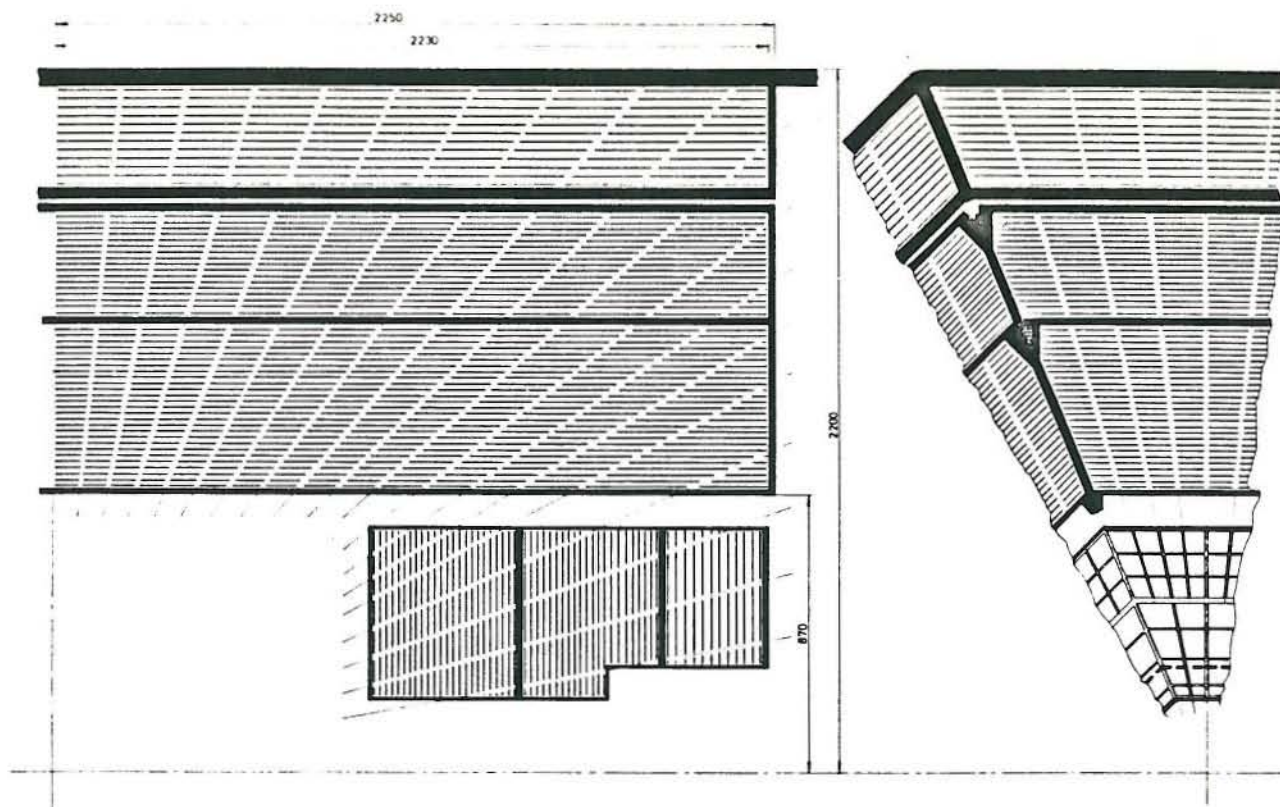


Fig. 1.5 Struttura a torri del calorimetro adronico

Le camere a muoni. Fuori dal tubo di supporto c'è il rivelatore per i muoni; l'impulso dei μ viene misurato con 3 piani di camere a deriva. I piani sono disposti lungo i lati di un prisma ottagonale in 3 strati e posti ad una distanza dal centro di interazione via via maggiore (fig. 1.6) (il più esterno è a 563 cm).

Le camere a deriva che compongono i 3 strati sono di due tipi :

- le "*p-chambers*" che, misurando la curvatura delle traiettorie dovuta al campo magnetico, permettono la misura dell'impulso
- le "*z-chambers*" da cui si può ottenere l'informazione riguardante la coordinata z (l'asse z corrisponde alla direzione del fascio).

Date le dimensioni dell'apparato è richiesto un sistema di allineamento

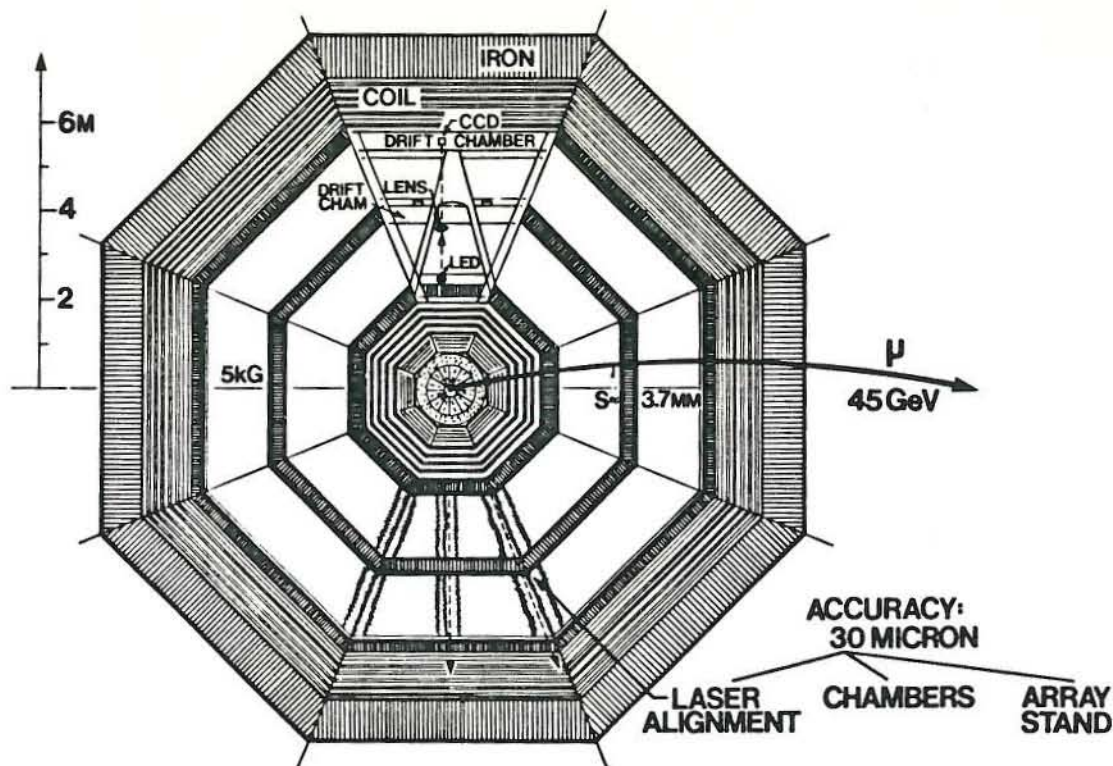


Fig. 1.6 Le camere a muoni

delle camere. Per mezzo di fasci laser si controlla la posizione delle camere e si correggono eventuali spostamenti utilizzando dei microattuatori. In questa maniera si riesce ad avere una risoluzione dell'impulso a 45 GeV pari a $\frac{\delta p}{p} = 1\%$.

I due rivelatori di luminosità, posti simmetricamente a circa 3 mt. dal vertice di interazione subito intorno al tubo del fascio (fig. 1.3), sono composti da due "corone" di cristalli di BGO alle quali vengono anteposte delle camere proporzionali per determinare le traiettorie delle particelle cariche. Mediante questi rivelatori ci si propone di misurare la luminosità della macchina all'1% e di rivelare eventi che producono 1 o 2 fotoni con traiettorie molto prossime all'asse del fascio.

Il rivelatore di L3 è interamente contenuto in un magnete solenoidale, capace di generare un campo di 5 kGauss, la cui forma è un prisma a base ottagonale con l'asse posto lungo la direzione del fascio (fig. 1.7).

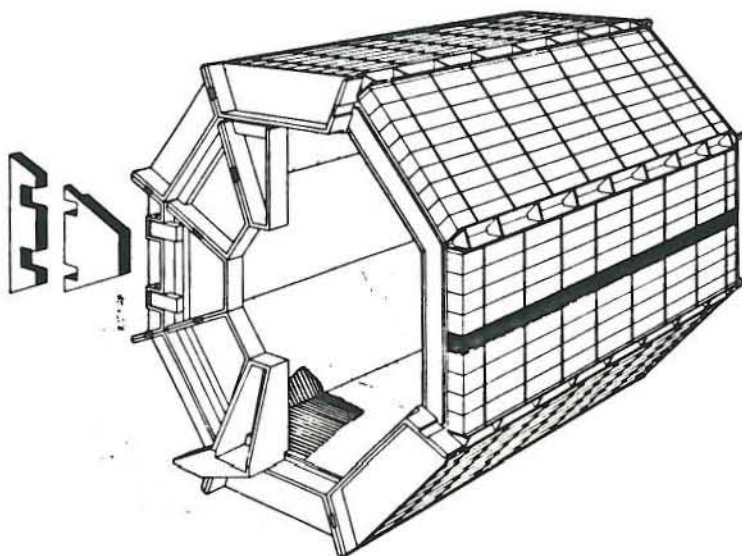


Fig. 1.7 Visione schematica del magnete

2 Il sistema di acquisizione dati di L3

2.1 Introduzione

In un esperimento di fisica delle alte energie si deve gestire una gran mole di dati: in LEP i pacchetti di elettroni e positroni vengono fatti collidere ogni $22 \mu s$ e ad ogni collisione può venir prodotta un'informazione pari a 100 kilobytes.

Con una qualsiasi macchina acceleratrice, a causa della piccola sezione d'urto degli elettroni, non viene sempre creata un'interazione in cui l'energia, concentrata in un volume piccolissimo, diventa materia generando nuove particelle: è stato calcolato [10,24] che in LEP si potrà in media ottenere al massimo qualche interazione utile per secondo.

Il compito del "sistema di filtro e acquisizione dati" è quello di memorizzare su un supporto di massa, di solito il nastro magnetico, gli "eventi buoni" scartando il più possibile il "fondo" formato dalle mancate interazioni e^+e^- , dai raggi cosmici, dalla radiazione di sincrotrone etc. In particolare il filtro (trigger) che effettua il riconoscimento degli "eventi buoni" ed avverte il sistema di acquisizione dati di scartare o memorizzare l'evento, deve essere abbastanza flessibile da accettare processi fisici nuovi ed inaspettati, nonché tale da adattarsi ad eventuali modifiche ed ampliamenti del rivelatore.

La disponibilità di risorse finanziarie ed umane pone ovvi limiti alla progettazione e realizzazione del sistema. Il compromesso tra costo e prestazioni

porta di solito a dei sistemi di acquisizione dati con 3-4 stadi successivi di filtro ed una struttura ad albero per il flusso dei dati, provenendo questi ultimi dai vari sub-rivelatori la cui elettronica di lettura spesso costituisce dei veri e propri sistemi di acquisizione dati elementari.

Il progetto di un sistema viene sviluppato tenendo conto dei seguenti fattori :

1. Frequenza di collisione.
2. Frequenza degli "eventi buoni" aspettata.
3. Tempo morto.
4. Costo.

Nell'esperimento L3, il sistema di filtro e acquisizione dati deve elaborare gli eventi alla frequenza del fascio (50 kHz), scartarne gran parte e raggiungere la frequenza aspettata degli eventi buoni che risulta essere di circa 1 Hz, cioè quella del picco della Z^0 . Ciò deve essere fatto non solo mantenendo il tempo morto, ovvero il tempo in cui non si possono acquisire nuovi dati, il più piccolo possibile per sfruttare al massimo il prezioso tempo di accensione dell' acceleratore, ma anche con costi minimi e compatibili col bilancio dell'esperimento.

Per quanto detto è stata scelta per il sistema di filtro e acquisizione dati di L3 una struttura come quella mostrata in figura 2.1 : i primi due livelli di filtro, molto veloci, elaborano una quantità di informazione (*trigger data*) inferiore a quella globale che invece viene elaborata dal terzo livello

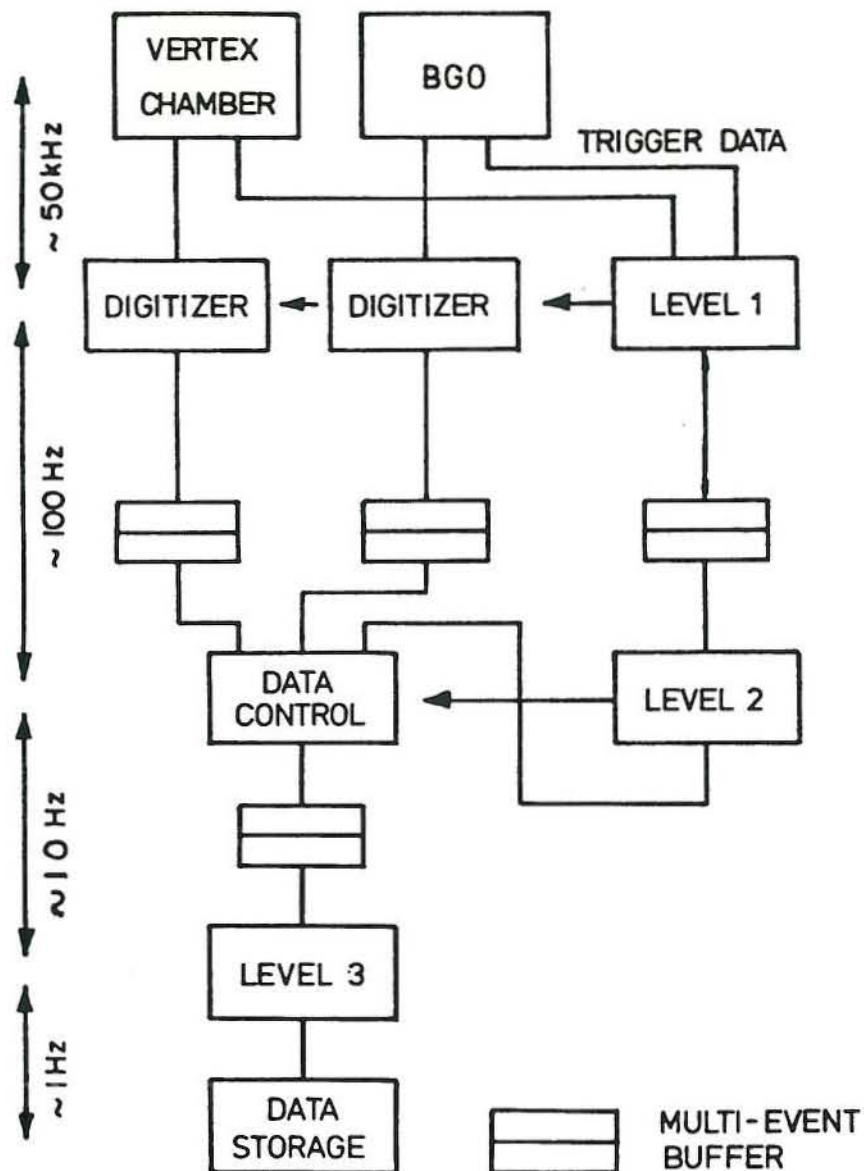


Fig. 2.1 Schema a blocchi del sistema acquisizione dati di L3

e poi, se l'evento è "buono", memorizzata su nastro. Questa struttura ha permesso di contenere i costi del sistema : bastano solo un migliaio di canali "veloci" per formare i "trigger data", mentre l'acquisizione dell'intera informazione richiede la conversione analogica-numerica di circa 100'000 segnali che avviene però soltanto alla frequenza di accettazione del filtro di primo livello ($100 \div 500$ Hz) e che quindi può essere fatta con elettronica più lenta

e perciò meno costosa.

Ad ogni collisione iniziano le conversioni sia dei "trigger data" che dei "dati principali"; dopo qualche μs gli ADC veloci finiscono di convertire i "trigger data" e può iniziare l'elaborazione del filtro di primo livello che sfrutta algoritmi molto semplici. Nonostante tale semplicità circa il 99% degli eventi risulta scartato: ci si aspetta [24] che dopo questa prima operazione di filtraggio la frequenza di eventi accettati in media sarà pari a 100 Hz o al massimo 500 Hz in condizioni particolarmente sfavorevoli. Nel caso in cui un evento è scartato viene fermata la conversione dei "dati principali" e si fa in modo che tutto sia pronto per la successiva collisione. Da quanto detto il filtro di secondo livello viene attivato al massimo ogni 2 ms; tale filtro utilizza algoritmi più complessi riducendo ulteriormente la frequenza di eventi accettati a 10 Hz. Infine il terzo livello, avendo un fattore di riduzione pari a 10, porta la frequenza degli eventi ad 1 Hz che è proprio quella aspettata per collisioni significative.

La struttura descritta consente oltre che il contenimento dei costi, di ottenere una buona flessibilità e funzionalità : il maggior tempo di calcolo a disposizione dei filtri a secondo e terzo livello (rispettivamente di 2 ms e 20 ms) permette l'uso di processori programmabili ed emulatori invece dei velocissimi e costosissimi *hardware processors* utilizzati per il primo livello. Per quanto riguarda specificamente il costo, c'è da tener presente che nell'esperimento L3 il secondo livello di filtro costa la metà del solo filtro in

energia di primo livello.

Dato che il tempo a disposizione per l'elaborazione di primo livello è pari a $22 \mu s$, per poter effettuare gli algoritmi di filtraggio in tempo, è necessario un *hardware processor* che sfrutti il più possibile il calcolo parallelo [23]. Quando un evento è accettato da questo filtro si può calcolare il tempo morto che si crea nel sistema: in tal caso il filtro a primo livello non elabora né acquisisce ulteriori eventi finché non viene terminata la conversione dei dati di tutti gli elementi del rivelatore e non ne vengono trasferiti i risultati in apposite memorie. Poiché questa operazione è lenta per ragioni di costo e viene effettuata in circa $500 \mu s$, tenendo conto che la frequenza media aspettata al primo livello è pari a 100 Hz , il tempo morto in percentuale, che coincide con la percentuale di eventi persi, è di $500 \mu s \times 100 \text{ Hz} = 5\%$. Infatti, dato che si può supporre che la probabilità nel tempo (λ) di avere un evento buono sia costante poichè dipende da alcuni parametri dell'acceleratore (luminosità, energia dei pacchetti), la probabilità di avere n eventi in un intervallo di tempo t segue la distribuzione di Poisson e quindi il numero medio di eventi persi dal filtro nel tempo t è pari al valor medio λt della poissoniana. Una delle condizioni che deve rispettare il sistema di acquisizione dati è di non peggiorare ulteriormente in forma significativa tale tempo morto.

Precedentemente si è visto che il filtro a secondo livello ha a disposizione per il calcolo un tempo medio di 2 ms : è quindi possibile utilizzare processori veloci programmabili come gli XOP del CERN. Anche se progettati in

maniera da effettuare tutti i calcoli nei limiti di tempo previsti, bisogna sempre utilizzare memorie tampone per i “trigger data” per evitare di perdere informazioni nel caso in cui si verificano due o più accettazioni consecutive del filtro a primo livello. Le memorie tampone dovranno essere multi-evento cioè capaci di memorizzare più eventi. Per calcolare la “profondità” di queste memorie si possono utilizzare i risultati della teoria delle code. Infatti si può descrivere una fase del processo di acquisizione in termini di una coda in ingresso con una data distribuzione (ad. es. una poissoniana con valor medio λ , dove nel nostro caso λ è il tempo medio di ingresso alla memoria tampone), uno stadio tampone profondo k eventi e uno stadio di elaborazione il cui tempo medio di calcolo è μ . Il tempo morto per questo processo [19] è dato dalla formula :

$$tempo\ morto(\%) = \frac{(1 - \eta)\eta^k}{1 - \eta^{k+1}} \quad con \quad \eta = \frac{\mu}{\lambda}.$$

In tabella 2.1 è mostrato il tempo morto in funzione di diversi valori di η e k .

Nel nostro caso alla massima frequenza media di accettazione prevista del primo livello (500 Hz) il tempo morto dovuto al tempo di conversione è del 25%, quindi, se $\eta = 0.8$ con $k = 8$, come previsto nel progetto, si ha un tempo morto aggiuntivo del solo 3%. Invece a 100 Hz di accettazione del primo livello si ha $\eta = 2\ ms \times 100\ Hz = 0.2$ e, con la stessa memoria profonda 8 eventi, (v. tabella 2.1) praticamente il tempo morto addizionale al sistema è trascurabile.

η	Buffer capacity			
	1	2	3	4
0.1	9.1	0.9	–	–
0.2	16.6	3.2	0.6	–
0.4	28.5	10.3	3.9	1.6
0.6	37.5	18.5	10.1	5.7
0.8	44.4	26.2	17.4	12.2
1.0	50.0	33.3	25.0	20.0
2.0	66.6	57.1	53.3	51.6

Tabella 2.1 Tempo morto in funzione di η e k

Il terzo livello di filtro, composto da vari emulatori in parallelo, ha a disposizione 20 ms di tempo di elaborazione. Poichè gli emulatori permettono di eseguire *online* gli algoritmi propri fino ad ora dell'analisi *offline*, a questo livello viene fatto un filtraggio molto sofisticato tale che non dovrebbero essere richieste ulteriori analisi almeno per i processi aspettati.

Come tra primo e secondo livello, anche lungo il flusso dei dati principali vengono poste delle memorie tampone per mantenere il tempo morto nei limiti richiesti. Data la complessità dell'intero sistema, per controllarne l'efficienza ed il funzionamento sono stati usati degli appositi programmi di simulazione; infatti simulando il flusso di dati proveniente dal rivelatore si può controllare l'efficienza degli algoritmi di filtraggio nei tre livelli, calcolare

Beam Crossing Rate	44.982kHz	
Level-1 Accept Rate	100Hz(Typ.)	500Hz(Max.)
Encoding and Buffering Time	<500 μ s	
Level-2 Accept Rate	10Hz(Typ.)	50Hz(Max.)
Level-3 Accept Rate	1Hz(Typ.)	5Hz(Max.)
Tape Writing Rate	1Hz(Typ.)	5Hz(Max.)
System Dead Time	5%(Typ.)	25%(Max.)
Muon Chamber Readout Throughput	500Hz	
Hadron Calorimeter Readout Throughput	500Hz	
BGO Readout Throughput	500Hz	
TEC Readout Throughput	50Hz	
Level-1 Trigger Throughput	45kHz	
Level-2 Trigger Throughput	500Hz	(4 x XOP's)
Level-3 Trigger Throughput	50Hz	(3 x 3081/E's)
Level-1 Trigger Processing Time	<21 μ s	
Level-2 Trigger Processing Time	<8ms	
Level-3 Trigger Processing Time	<60ms	

Tabella 2.2 Principali parametri del sistema acquisizione dati di L3

sia la profondità delle memorie tampone che i tempi di trasferimento delle informazioni, ottenere infine il valore totale del tempo morto.

In tabella 2.2 sono riassunti i principali parametri del filtro nel suo insieme e del sistema di acquisizione dati di L3 [10].

2.2 Il filtro di primo livello di L3

Nel capitolo 1 abbiamo visto che gli stati finali principali dei processi in studio sono caratterizzati dalla creazione di elettroni, muoni, fotoni (per cui tra l'altro il rivelatore è ottimizzato) e *jets* adronici. Questo corrisponde, nel rivelatore, a depositi di energia nei calorimetri provocati da e, γ , e adroni, e alla rilevazione delle tracce generate da e e μ nelle camere di vertice e a muoni rispettivamente. Per il filtro di primo livello si può ottenere l'analisi

degli eventi con algoritmi relativamente semplici come : calcolare la quantità di energia rilasciata nel detector e controllare eventuali depositi isolati di energia nei cristalli di BGO (filtro di energia e fotone singolo), controllare se ci sono almeno più di due tracce di particelle cariche (filtro di TEC), verificare l'esistenza di tracce di muoni provenienti dal vertice di interazione (filtro di muoni) [14]. Sono quindi necessari 3 tipi di filtro di 1° livello ognuno specifico per uno o due sub-rivelatori. Comunque già il solo filtro di energia, di cui parleremo in maniera dettagliata nel capitolo 3, fornisce una selezione sufficiente degli eventi, anche se non completa: infatti potrà accadere che alcuni eventi saranno riconosciuti contemporaneamente da più di un tipo di filtro a primo livello.

2.3 Il filtro di secondo livello

Il filtro di secondo livello [22] (fig. 2.2) si basa su l'XOP (eXpandable Online Processor) [25] un processore molto veloce e microprogrammabile.

L'XOP, sviluppato presso la *Data handling Division* del CERN, può eseguire microistruzioni su parole di 16 bit in 100 ns ed essere interfacciato sia allo standard FASTBUS [8], per quanto riguarda il trasferimento dati, sia al VME tramite il quale viene microprogrammato. Nell'esperimento L3 il filtro di secondo livello è formato da 4 di questi calcolatori usati in parallelo che, a turno, elaborano i dati e mandano i risultati al livello successivo. I 4 XOP vengono posti tra 2 *crate* FASTBUS dove sono alloggiate le memorie per

del filtro di primo livello, ed è pari a 8 ms (4×2 ms) grazie all'utilizzo dei 4 XOP in parallelo.

Finita l'elaborazione gli XOP trasferiscono un'informazione a 2 DSM (Dual Slave Memory) che sono memorie a doppia porta FASTBUS. Tale informazione è costituita dal numero dell'evento, dalla decisione (si/no) del primo e del secondo livello e da tutti i dati d'ingresso su cui è avvenuta l'elaborazione. L'informazione contenuta nella prima DSM sarà utilizzata dal "ricostruttore di eventi" (vedi par. 2.4) per accettare o scartare l'evento; quella contenuta nella seconda servirà per controllare il funzionamento dei filtri di primo e secondo livello : questa memoria infatti è collegata ad un calcolatore VAX 750 dedicato al controllo dei filtri. Dato che i 4 XOP lavorano in parallelo e che il tempo di elaborazione può essere diverso da un evento all'altro, può accadere che gli eventi elaborati non siano più in ordine cronologico. Per ricostruire tale ordine la memoria della DSM viene divisa in blocchi ed ogni XOP può conoscere dal numero dell'evento elaborato il blocco in cui memorizzarlo.

Inoltre gli XOP inviano l'informazione riguardante la decisione ed il numero dell'evento al DRP (Data Reduction Processor) (v. fig. 2.2). Il DRP è un compattatore di dati e riduce, per ogni evento, gli 1.5 Megabytes di informazioni provenienti dalla TEC a 63 Kilobytes. In caso di evento scartato la compattazione non sarà eseguita.

2.4 Il filtro di terzo livello e il sistema di acquisizione dati

Il sistema di acquisizione dati e il terzo livello di filtro di L3 [10] è interamente realizzato utilizzando il protocollo FASTBUS [8] (fig. 2.3).

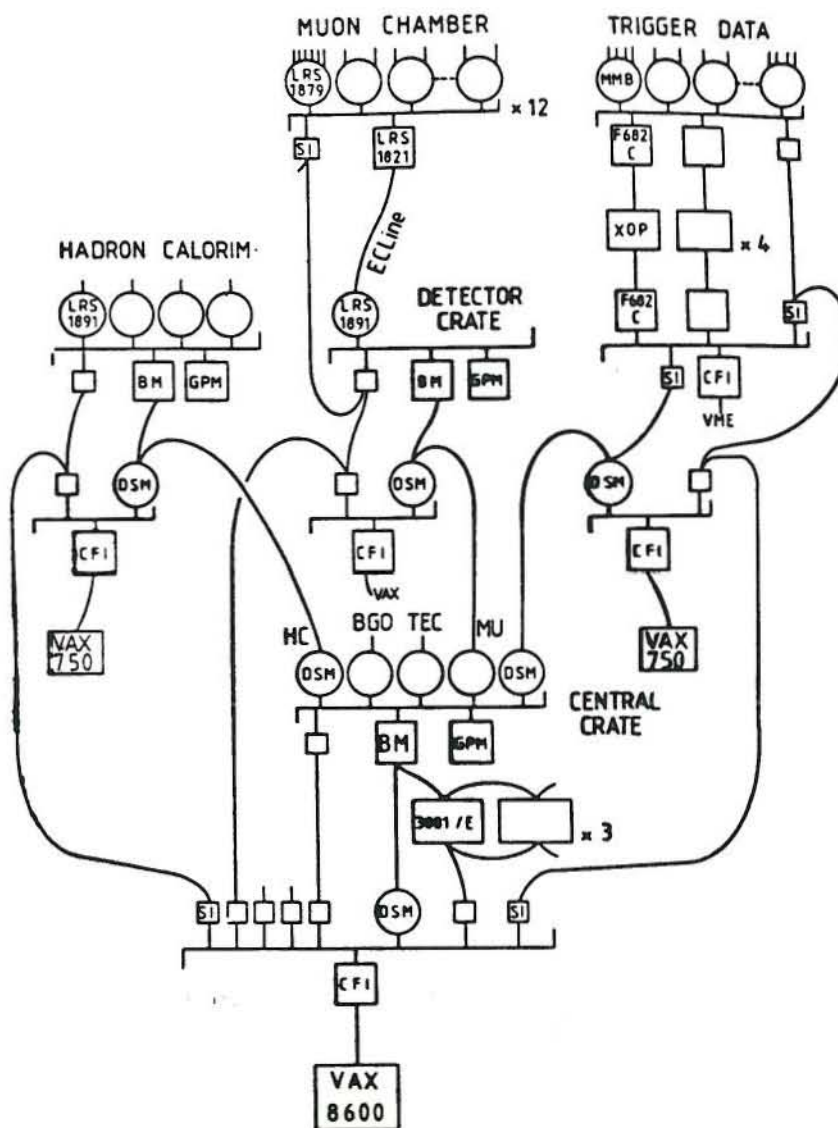


Fig. 2.3 Il DAQ e il terzo livello di filtro

Tale sistema, dopo le conversioni, è organizzato in maniera che esiste un *crate* Fastbus dove alloggiano le memorie tampone a doppia porta (LRS 1891) in cui vengono immagazzinati tutti i dati provenienti da un singolo

sub-rivelatore. I dati di ogni sub-rivelatore riguardanti lo stesso evento vengono riordinati e poi trasferiti insieme da un ricostruttore parziale di eventi ad altre due memorie tampone a doppia porta (DSM). Il ricostruttore è formato da un GPM (Fastbus General Purpose Master) e da un modulo BM (Fastbus Block Mover) il quale sotto il controllo del GPM, riesce a trasferire blocchi di dati ad alta velocità. La prima DSM può essere letta dal VAX 750 dedicato al controllo dei sub-rivelatori. La seconda si trova nel *crate* centrale e verrà utilizzata da un altro "ricostruttore di eventi". Compito di tale ricostruttore è di fondere insieme i dati provenienti dai 4 sub-rivelatori in modo da "ricostruire" l'evento completo. Per fare ciò vengono utilizzati di nuovo un GPM ed un BM che accedono ai dati di 5 DSM. Nelle prime quattro DSM sono memorizzati i dati provenienti dai quattro sub-rivelatori corredati del numero dell'evento, mentre nella quinta sono presenti i risultati delle decisioni ed i dati elaborati dai filtri di primo e secondo livello. Sfruttando le informazioni contenute nella quinta DSM il "ricostruttore" deciderà se preparare l'evento completo o cancellarlo dalle memorie semplicemente muovendone i puntatori di lettura e scrittura. A questo punto l'informazione completa riguardante un evento "buono" è trasferita dal "ricostruttore" al filtro di terzo livello.

Il filtro di terzo livello è formato da 3 emulatori 3081/*E* [20] che lavorano in parallelo (come gli XOP del 2° livello).

Il 3081/*E* emula le principali funzioni di elaborazione del calcolatore

IBM 3081. Essendo un emulatore e non un calcolatore non ha un sistema operativo nè dispositivi di I/O sofisticati, ma può eseguire programmi il cui microcodice può venir generato da linguaggi ad alto livello, come il FORTRAN. In particolare si possono prima sviluppare i programmi di analisi degli eventi in FORTRAN su di un computer IBM e poi trasferirli, una volta compilati e tradotti, sul 3081/E. Questo fatto rende questo tipo di elaboratori molto utile per l'analisi *online* degli eventi perchè è possibile utilizzare tutti gli algoritmi dell'analisi *offline* senza effettuare onerose modifiche ed adattamenti o peggio traduzioni in microcodice dei programmi. Tra le caratteristiche del 3081/E ricordiamo che : può gestire l'aritmetica in doppia precisione, ha fino a 7 Megabytes di memoria, ha una velocità di calcolo di 4 MFLOPS (MFLOP : 1 milione di operazioni in virgola mobile al secondo), e può comunicare con l'esterno tramite interfaccia FASTBUS e VME.

Dei tre emulatori che formano il filtro di terzo livello, quando uno è inattivo riceve i dati dal BM, li elabora e, se l'evento è buono, trasferisce i dati al calcolatore principale di acquisizione (VAX 8800) che si occupa poi di trascriverli su nastro. Dato che il tempo di calcolo a disposizione per ogni elaboratore è di 3×20 ms, con l'utilizzo delle DSM, che sono memorie tampone, si mantiene il tempo morto nei limiti richiesti. Ci si aspetta una frequenza di eventi accettati dal terzo livello al massimo di 5 Hz, frequenza che non pone ancora nessun problema per la registrazione su nastro anche di eventi con grossa mole di dati.

3 Il filtro in energia di primo livello

3.1 Segnali per il filtro in energia

Si è visto (2.1) che per il filtro in energia e di fotone singolo, vengono utilizzati dei segnali analogici “dedicati” provenienti dai calorimetri elettromagnetico (ELM) ed adronico (HAD), dagli scintillatori (SCIN), dal rivelatore di luminosità (LUM) e dalla camera di vertice (TEC). Nel capitolo 1 non si è fatto cenno agli scintillatori, posti tra i due calorimetri, perchè le informazioni ottenibili da essi sono utilizzate solo dai filtri di primo e secondo livello.

Ogni segnale è generato da un segmento più o meno grande di ogni sub-rivelatore ed è ottenuto sommando più segnali elementari: in questa maniera si ha un'informazione meno dettagliata dell'evento ma è possibile elaborarla nei tempi richiesti e con costi accettabili.

Ciascun segmento del rivelatore viene individuato con un sistema di coordinate sferiche con origine nel punto di interazione, come mostrato in figura 3.1 e 3.2 in cui sono riportate le proiezioni sui piani xy ($R\phi$) e yz ($R\theta$) del rivelatore dove l'asse z coincide con l'asse del fascio.

I segmenti del calorimetro elettromagnetico sono ottenuti dividendo quest'ultimo in 16 intervalli in θ (fig. 3.2) e 32 in ϕ (fig. 3.1). Dato che tutti gli intervalli in ϕ sono uguali e che ci sono 160 cristalli di BGO in ogni

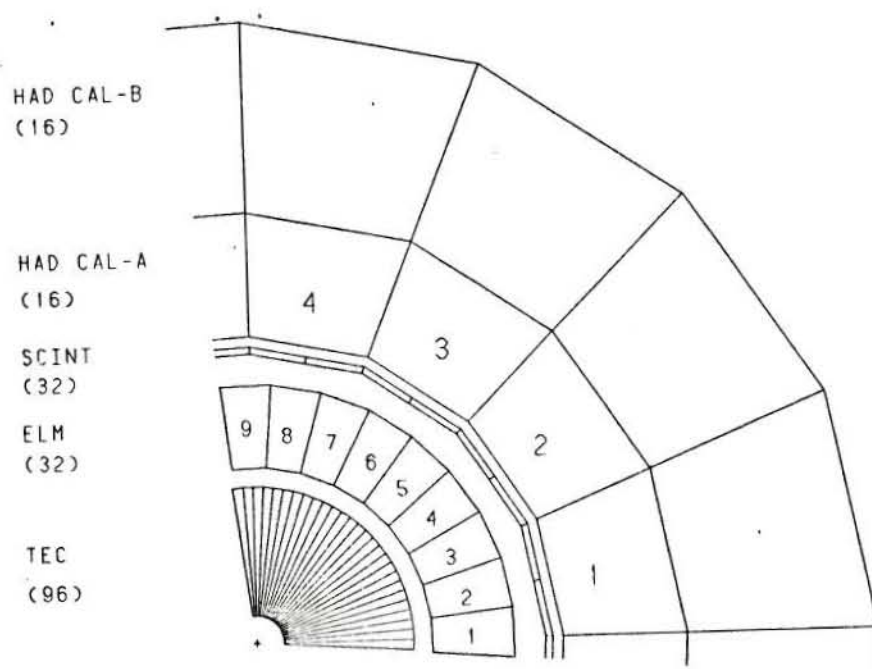


Fig. 3.1 proiezione XY del rivelatore ($R\phi$)

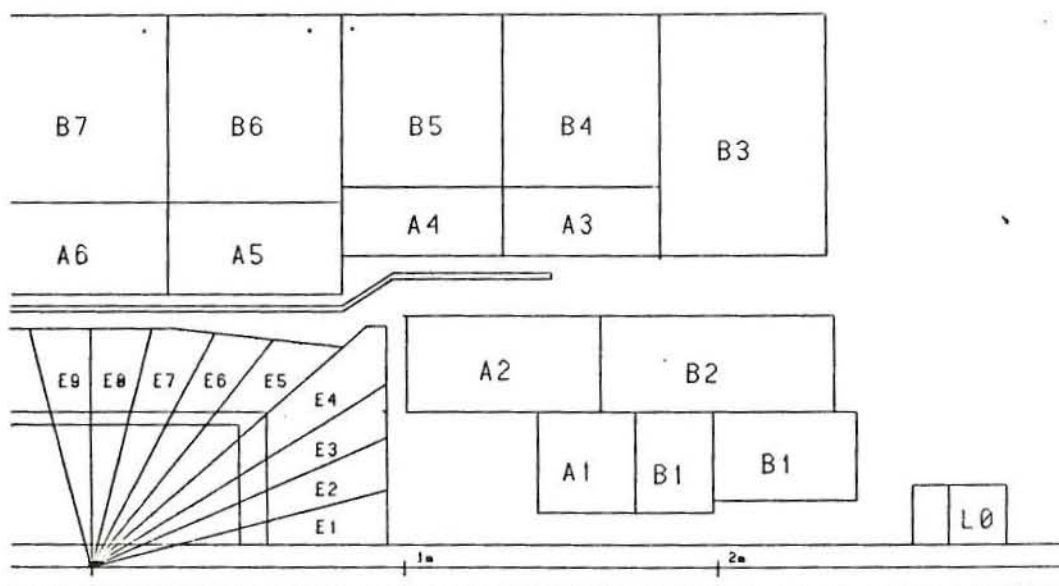


Fig. 3.2 proiezione YZ del rivelatore ($R\theta$)

piano $R\phi$, ogni segmento ϕ contiene circa 5 cristalli. I limiti di ogni intervallo in ϕ sono dati dalla formula :

$$-4.50^\circ + \Delta\phi \cdot (n - 1) < \phi_n < -4.50^\circ + \Delta\phi \cdot n$$

$$\text{dove} \quad \Delta\phi = 11.25^\circ$$

La segmentazione in θ , mostrata in tabella 3.1, copre il cilindro e i “tappi” del calorimetro elettromagnetico. Ogni intervallo contiene in questo caso 6 cristalli e perciò ogni segnale utilizzato dai primi due livelli di filtro è formato dalla somma analogica di quelli generati da $5 \times 6 = 30$ cristalli di BGO.

Per il calorimetro adronico si prelevano due segnali da un solo modulo fisico del calorimetro. Infatti tali segnali corrispondono a due differenti campionature in profondità del calorimetro stesso (parte interna A, parte esterna B). Il cilindro viene diviso in 9 anelli (fig. 3.2): i 7 più centrali corrispondono a 7 intervalli in θ sia per la parte A che per la parte B mentre i due più esterni corrispondono a due intervalli θ solo per la parte B. I “tappi” sono formati da 4 segmenti in θ sia per A che per B. Quindi ci sono in totale 11 intervalli in θ per la parte A e 13 intervalli per B come mostrato in tabella 3.2 e 3.4. Nel piano $R\phi$ il cilindro è diviso in 16 elementi i cui limiti sono :

$$\Delta\phi \cdot (n - 1) < \phi_n < \Delta\phi \cdot n$$

$$\text{dove} \quad \Delta\phi = 22.5^\circ$$

$\theta - \text{bin}$	$\theta_{\min}$	$\theta_{\max}$
E(1, ϕ)	5.0°	14.4°
E(2, ϕ)	14.4°	23.7°
E(3, ϕ)	23.7°	33.0°
E(4, ϕ)	33.0°	42.4°
E(5, ϕ)	42.4°	52.2°
E(6, ϕ)	52.2°	63.6°
E(7, ϕ)	63.6°	76.4°
E(8, ϕ)	76.4°	90.0°
E(9, ϕ)	90.0°	103.6°
E(10, ϕ)	103.6°	116.4°
E(11, ϕ)	116.4°	127.8°
E(12, ϕ)	127.8°	137.6°
E(13, ϕ)	137.6°	147.0°
E(14, ϕ)	147.0°	156.3°
E(15, ϕ)	156.3°	165.6°
E(16, ϕ)	165.6°	175.0°

Tabella 3.1 Segmentazione θ del calorimetro elettromagnetico (indicato con E)

$\theta - \text{bin}$	$\theta_{\min}$	$\theta_{\max}$
A(1, ϕ)	5°	17°
A(2, ϕ)	17°	36°
A(3, ϕ)	29°	42°
A(4, ϕ)	38°	56°
A(5, ϕ)	50°	76°
A(6, ϕ)	74°	106°
A(7, ϕ)	104°	130°
A(8, ϕ)	124°	142°
A(9, ϕ)	138°	151°
A(10, ϕ)	144°	163°
A(11, ϕ)	163°	175°

Tabella 3.2 Segmentazione θ della parte A del calorimetro adronico (indicato con A)

θ - bin	$\theta_{\min}$	$\theta_{\max}$
B(1, ϕ)	5°	13°
B(2, ϕ)	13°	27°
B(3, ϕ)	23°	43°
B(4, ϕ)	34°	52°
B(5, ϕ)	43°	65°
B(6, ϕ)	57°	81°
B(7, ϕ)	76°	104°
B(8, ϕ)	99°	123°
B(9, ϕ)	115°	137°
B(10, ϕ)	128°	146°
B(11, ϕ)	137°	157°
B(12, ϕ)	153°	167°
B(13, ϕ)	167°	175°

Tabella 3.3 Segmentazione θ della parte B del calorimetro adronico (indicato con B)

ϕ - bin	$\phi_{\min}$	$\phi_{\max}$
A(θ , 1), B(θ , 1)	0.0°	45.0°
A(θ , 2), B(θ , 2)	22.5°	67.5°
A(θ , 3), B(θ , 3)	45.0°	90.0°
A(θ , 4), B(θ , 4)	67.5°	112.5°
A(θ , 5), B(θ , 5)	90.0°	135.0°
A(θ , 6), B(θ , 6)	112.5°	157.5°
A(θ , 7), B(θ , 7)	135.0°	180.0°
A(θ , 8), B(θ , 8)	157.5°	202.5°
A(θ , 9), B(θ , 9)	180.0°	225.0°
A(θ ,10), B(θ ,10)	202.5°	247.5°
A(θ ,11), B(θ ,11)	225.0°	270.0°
A(θ ,12), B(θ ,12)	247.5°	292.5°
A(θ ,13), B(θ ,13)	270.0°	315.0°
A(θ ,14), B(θ ,14)	292.5°	337.5°
A(θ ,15), B(θ ,15)	315.0°	360.0°
A(θ ,16), B(θ ,16)	337.5°	22.5°

Tabella 3.4 Segmentazione ϕ dei "tappi" del calorimetro adronico parti A e B

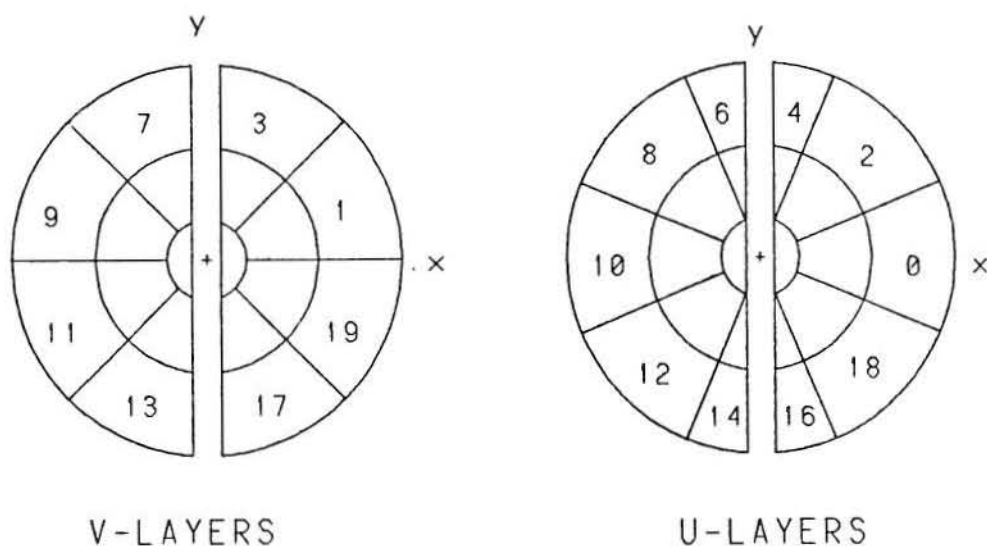


Fig. 3.3 Segmentazione dei "tappi" dell'HAD

Per i "tappi" la suddivisione è ben più complessa in quanto ci sono in totale 20 piani di fili alternati di strati-U e strati-V con segmentazione sfasata (fig. 3.3): ogni piano di fili copre 45° in ϕ e quindi segmenti adiacenti si sovrappongono. La figura 3.3 ne mostra la numerazione e la loro corrispondenza con i segmenti in ϕ è illustrata in tabella 3.4.

Nel rivelatore di luminosità vengono generati 16×2 segnali corrispondenti a 16 intervalli in ϕ e 2 in θ . L'angolo θ individua le regioni avanti ed indietro del rivelatore.

Dai contatori a scintillazione sono prelevate 32 informazioni da un bit. Queste informazioni, provenienti da 32 segmenti in ϕ e date dall'OR logico di 3 di questi contatori, vengono utilizzate nel filtro di *clusters* per avere

ulteriori informazioni sulla creazione e direzione dei fasci adronici.

Infine la camera di vertice TEC (Time Expansion Chamber) fornisce 32 informazioni da un bit formate dall'OR di 6 suoi segmenti in ϕ . I gruppi di 6 segmenti sono ampi quanto due segmenti in ϕ del calorimetro elettromagnetico e forniscono informazioni relative al rilevamento di una o più tracce in quella regione. In questa maniera possiamo effettuare delle correlazioni tra tracce nella TEC e depositi di energia nel calorimetro elettromagnetico.

In totale il filtro in energia di primo livello deve elaborare :

- 512 segnali generati dal calorimetro elettromagnetico e codificati in parole da 11 bit,
 - 384 segnali dal calorimetro adronico codificati in parole da 10 bit,
 - 32 segnali dal rivelatore di luminosità codificati in parole da 11 bit,
 - 32 segnali dagli scintillatori codificati in parole da 1 bit,
 - 32 segnali dal rivelatore di vertice codificati in parole da 1 bit,
- per un totale di 9.6 Kilobyte di informazioni.

3.2 Condizioni di "trigger"

Le condizioni di "trigger", cioè le condizioni in presenza delle quali si decide di accettare un evento, vengono stabilite considerando le caratteristiche degli stati finali dei processi fisici in studio. Gran parte di questi è caratterizzata dal rilascio di energia nei calorimetri o dal fatto di generare un fotone singolo ($Z^0 \rightarrow \gamma + \nu\bar{\nu}$) o fasci di adroni a bassa energia. Queste

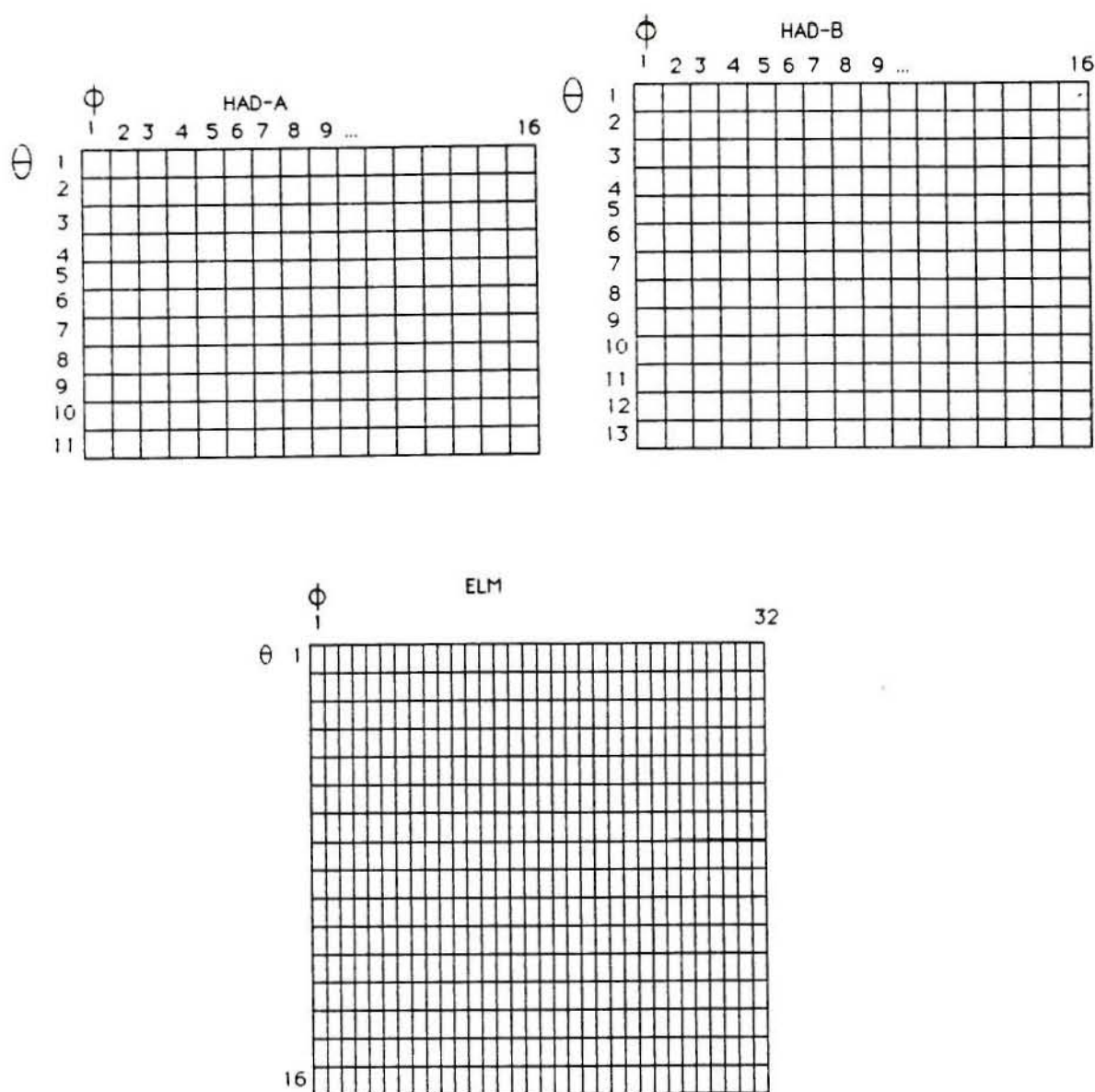


Fig. 3.4 Le matrici HAD-A, HAD-B, ELM

considerazioni portano a scegliere un insieme di condizioni che permettano un opportuno filtraggio degli eventi il quale però non sia estremamente selettivo in modo da non scartare processi fisici inaspettati. Per questo motivo si tengono liberi molti parametri, come le soglie minime delle energie rilasciate, che potranno essere "aggiustati" al momento della presa dei dati.

Tenendo conto della segmentazione effettuata nei calorimetri, si possono

formare tre matrici [12], due per il calorimetro adronico ed una per quello elettromagnetico in cui ogni elemento rappresenta l'energia rilasciata nel segmento θ_i, ϕ_j : in seguito si farà riferimento a queste matrici rispettivamente con i nomi HAD-A, HAD-B, ELM (fig. 3.4). Sommandone gli elementi per righe e per colonne si formano 6 vettori le cui componenti rappresentano la distribuzione in energia nei piani $R\theta$ e $R\phi$: HAD A- $R\theta$, HAD B- $R\theta$, ELM- $R\theta$ sono quelli ottenuti sommando le colonne delle matrici e HAD A- $R\phi$, HAD B- $R\phi$, ELM- $R\phi$ quelli sommando le righe; le componenti di ognuno di essi verranno indicate ponendo un indice al nome del vettore. Dopo il calcolo di questi vettori, nel filtro di primo livello, si verifica se le condizioni di "trigger" sono soddisfatte per ogni evento.

Tali condizioni vengono stabilite [13] considerando i seguenti fattori :

1. Energia :

Per molte reazioni e^+e^- c'è un rilascio di energia nei calorimetri superiore ad una certa percentuale (10÷20%) dell'energia nel centro di massa, quindi nel filtro vengono calcolati e controllati vari bilanci energetici relativi ai due calorimetri.

In particolare si calcola :

- L'energia totale depositata nel calorimetro elettromagnetico (ELM) e la sua somma con quella rivelata nel calorimetro adronico (HAD). Ciò si ottiene sommando tutte le componenti dei rispettivi vettori $R\theta$ (o $R\phi$) (SLA: Small and Large Angle). Si è in presenza di eventi

- buoni se l'energia totale è superiore ad una certa soglia. Inoltre si calcola anche la sola energia depositata nel calorimetro elettromagnetico per poter selezionare eventi a stati finali non adronici.
- L'energia depositata a grandi θ (Large Angles, LA) misurata sia nel calorimetro elettromagnetico che in tutto il rivelatore per cui si può porre una soglia più bassa che nel precedente caso senza accettare eventi spuri perchè si rimuove la regione a piccolo θ , più vicina all'asse del fascio, soggetta al fondo causato dalle interazione dei pacchetti con il gas residuo nel tubo dell'acceleratore. In questo caso il calcolo viene fatto sommando le componenti HAD-A $R\theta_i$, HAD-B $R\theta_i$ ed ELM $R\theta_i$ corrispondenti a grandi θ .
 - Il bilancio energetico $E_z = \sum E(\theta_i) \cos(\theta_i)$ lungo la direzione del fascio per ambedue i calorimetri calcolato su tutti i valori di θ_i il quale permette di accettare gli eventi ad energia mancante (*missing energy*). Infatti il bilancio E_z è nullo se il rivelatore riesce a misurare l'energia di tutte le particelle prodotte in un interazione "buona", mentre è diverso da zero se ci sono asimmetrie che testimoniano la presenza di particelle, come i neutrini, la cui energia non può essere rivelata.
 - Il rapporto, $(\frac{ELM}{ELM + HAD})$, tra l'energia misurata nel calorimetro elettromagnetico e l'energia totale che consente di selezionare eventi i cui stati finali siano adronici o con rilascio di energia nel solo

calorimetro elettromagnetico.

2. Clusters:

anche se le condizioni riportate al punto 1 non sono soddisfatte, si può decidere di accettare l'evento se l'energia, al di sopra di una certa soglia, è in massima parte depositata su un ristretto numero di coni di piccola apertura angolare (*clusters*). Per cercare i *clusters* nel piano $R\phi$ si analizzano i relativi vettori calcolati: si trova un *cluster* se l'energia rilasciata nel calorimetro elettromagnetico e nelle parti A e B di quello adronico è consistente con quella prodotta da una particella proveniente dal punto di interazione ed è al di sopra di un certo valore di soglia. Per tale ricerca debbono venir utilizzate anche le informazioni provenienti dalla TEC e dagli Scintillatori che, dando ulteriori notizie sulle direzioni delle tracce, permettono di separare l'evento dal fondo. Per il piano $R\theta$ viene eseguito un calcolo analogo. La decisione di accettare o scartare l'evento dipenderà dal numero di *clusters* trovati nelle 2 proiezioni.

3. Conteggio di tracce ad alta energia:

si può avere un evento significativo se uno o più elementi delle matrici ammonta ad un certo valore. Ogni elemento, cioè l'energia depositata in ogni segmento, è comparato con una soglia che potrà essere funzione di θ ed essere differente per i due calorimetri. Tutte le tracce la cui energia è superiore a suddetta soglia sono contate e se il loro numero è maggiore di un certo valore si considera l'evento buono.

4. Fotone singolo :

anche se l'energia rilasciata è molto bassa, si può accettare l'evento se la distribuzione in energia è compatibile con un evento di fotone singolo (principalmente $Z_0 \rightarrow \gamma + \nu\bar{\nu}$ e, si spera, $Z_0 \rightarrow H_0 + \gamma$). In questo caso tutta l'energia elettromagnetica è confinata in una piccola regione del calorimetro elettromagnetico. Il fotone può colpire i segmenti del calorimetro in uno dei tre modi mostrati in figura 3.5.

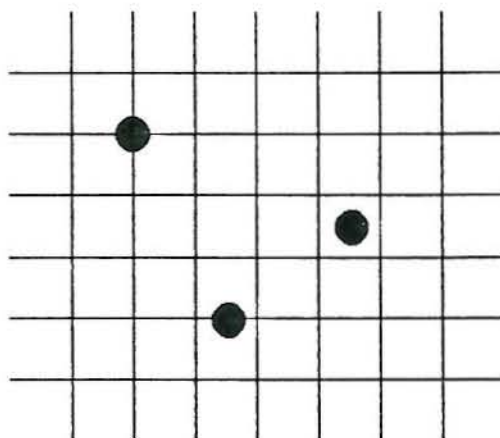


Fig. 3.5 Modi in cui il fotone colpisce i segmenti

Per verificare se questa condizione è soddisfatta si considerano tutte le possibili somme di due righe adiacenti e tutte le possibili somme di due colonne adiacenti della relativa matrice e le si confrontano con l'energia elettromagnetica totale. Si verifica un evento di fotone singolo se almeno due di queste somme, una per le righe ed una per le colonne, corrispondono a gran parte dell'energia rilasciata, si pensa più dell'80%. Per evitare di accettare il fondo ci sarà una soglia minima per l'energia

elettromagnetica totale, al di sotto della quale non sarà attivo il filtro di fotone singolo.

5. Rivelatore di luminosità:

se il rivelatore di luminosità, anteriore e posteriore, rivela delle tracce ad alta energia con una forte correlazione avanti - indietro, allora si è in presenza di eventi di Bhaba (cioè $e^+e^- \rightarrow e^+e^-$) che devono essere accettati. Una ulteriore condizione di “trigger” ricavabile dal rivelatore di luminosità è quella di *single tag*, ovvero selezionare eventi che depositano l'energia in uno dei due rivelatori di luminosità e nei calorimetri.

6. Overflow :

il filtro in energia di primo livello deve essere progettato in modo che non si verifichino “overflow” fino ad un energia totale di 200 GeV. Solo in casi eccezionali, raggi cosmici altamente energetici o moduli malfunzionanti, potrà accadere un “overflow”. Se tale condizione si verifica l'evento dovrà essere comunque accettato per poter effettuare ulteriori controlli nel secondo e terzo livello di filtro.

3.3 Struttura generale del filtro

Il filtro in energia di primo livello è stato progettato utilizzando 9 tipi di moduli CAMAC [3]. Lo standard CAMAC è stato scelto per la reperibilità sul mercato di alcuni di questi moduli e per la sua versatilità e semplicità. Il “dataway” CAMAC viene usato solo per il controllo e l'inizializzazione delle

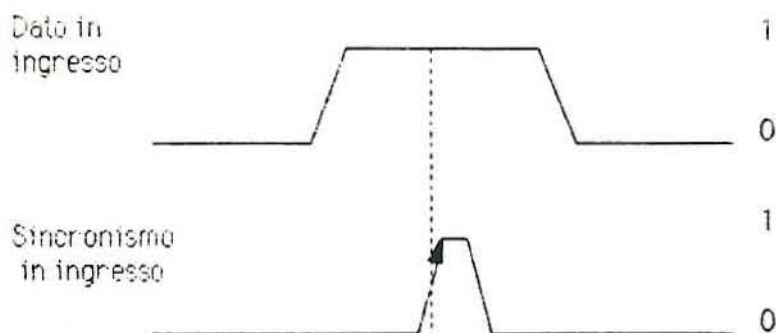


Fig. 3.6 Il protocollo di trasmissione dati ECL differenziale

operazioni del filtro mentre per il trasferimento veloce dei dati da modulo a modulo è utilizzato, via pannello frontale, un “bus” ECL differenziale.

Il protocollo di tale “bus” [2] prevede che il dato, costituito da una parola di 16 bit, sia accompagnato da un segnale (fig. 3.6) il cui fronte di salita può essere utilizzato come sincronismo.

Utilizzando tali moduli in cascata è stato progettato un *hardware processor*, cioè un processore a logica cablata, costituito da 3 parti fondamentali :

1. La conversione analogico-digitale veloce dei segnali.
2. L’elaborazione numerica delle informazioni.
3. La temporizzazione dell’intero sistema.

Infatti con dei moduli commerciali vengono effettuate le conversioni analogico-digitali dei segnali provenienti dai sub-rivelatori e le informazioni numeriche ottenute sono elaborate da un insieme di moduli che, lavorando in “pipeline”, necessitano di un’accurata temporizzazione globale.

3.3.1 La conversione dei segnali

La necessità di riuscire a “vedere” gli eventi di fotone singolo fino ad ener-

gie del fotone dell'ordine di 100 MeV, poichè in questo caso la sezione d'urto del processo $Z^0 \rightarrow \gamma + \nu\bar{\nu}$ è grande, comporta che i segnali del calorimetro elettromagnetico siano convertiti perlomeno con parole di 11 cifre binarie. Infatti se si considera comunque buono un evento in cui l'energia depositata in un segmento del calorimetro elettromagnetico sia almeno pari a 50 GeV, si ottiene una risoluzione nel convertitore analogico-digitale di $50 \text{ GeV} / (2^{11} \simeq 2000) = 25 \text{ MeV}$ per bit.

Per il calorimetro adronico, non essendoci questa necessità stringente ed essendo la risoluzione energetica ben minore di quella dei cristalli di BGO, una conversione con 10 bit, a cui corrisponde una risoluzione di 50 MeV per bit, risulta più che sufficiente.

Così è stato scelto un ADC che potesse convertire i segnali generati nei due calorimetri in 10 od 11 bit in tempi brevi : il Fast Encoding e Readout ADC (FERA) della LeCroy Research Systems. Il FERA è dotato di 16 canali i cui tempi di conversione, per parole di 10 o 11 bit, sono rispettivamente pari a $4.8 \mu\text{s}$ e $8.5 \mu\text{s}$; dopo la conversione, i risultati possono venir letti sequenzialmente attraverso un "bus" ECL differenziale al massimo uno ogni 100 ns.

Poichè il ritardo ingresso-uscita del modulo più lento usato nella costruzione dello *hardware processor* è di 60 ns, e tenendo conto del fatto che la decisione finale deve essere fornita prima del momento in cui si verifica la nuova collisione, per la temporizzazione di tutti i moduli è stato scelto un

periodo di 60 ns.

Per poter utilizzare tale periodo anche nella lettura dei FERA, si è pensato di usarne due in parallelo [12], leggerne i risultati da ciascuno ogni 120 ns e commutare le 2 uscite ogni 60 ns su di un'unico "bus" mediante il modulo "Bus Switch" realizzato presso il Laboratorio di Elettronica della sezione di Roma dell'INFN. In realtà per poter suddividere i FERA in soli 9 gruppi di conversione si sono utilizzate per ogni "Bus Switch" due coppie di FERA in parallelo. Da 8 di questi 9 gruppi di conversione si possono prelevare sequenzialmente le informazioni digitalizzate $\theta_i\phi_j$ relative a agli elementi di 2 righe di ogni matrice per essere inviate alla parte di elaborazione numerica del filtro di primo livello. Il flusso di informazioni generato da ogni gruppo di conversione è schematizzato in figura 3.7.

3.3.2 L'elaborazione numerica

Per poter connettere la parte di conversione del filtro con quella di elaborazione numerica si utilizzano quindi nove cavi: otto sono usati per il trasferimento delle informazioni riguardanti l'energia depositata nei segmenti del calorimetro elettromagnetico ed adronico, il nono per quella depositata nei rivelatori di luminosità. Poichè, come si vedrà in seguito, la conversione dei segnali provenienti dal calorimetro adronico finisce $4.3 \mu s$ prima di quella del calorimetro elettromagnetico, tempo più che sufficiente per la lettura delle informazioni nei Fera relative agli elementi delle due matrici HAD-A e HAD-B, è possibile utilizzare gli stessi cavi per il trasferimento anche dei

	HAD A			HAD B			ELM			
gruppo	1	16-17	32	33	48-49	64	65	96-97	128	t
0							0,1	0,16-17,1	17,16	In passi di 60 ns
1							1,1	1,32-16,1	16,32	
2				1,1	1,16-13,1	13,16	2,1	2,32-15,1	15,32	
3	1,1	1,16-11,1	11,16	2,1	2,16-12,1	12,16	3,1	3,32-14,1	14,32	← $i,j = \theta_i \phi_j$
4	2,1	2,16-10,1	10,16	3,1	3,16-11,1	11,16	4,1	4,32-13,1	13,32	
5	3,1	3,16-9,1	9,16	4,1	4,16-10,1	10,16	5,1	5,32-12,1	12,32	
6	4,1	4,16-8,1	8,16	5,1	5,16-9,1	9,16	6,1	6,32-11,1	11,32	
7	5,1	5,16-7,1	7,16	6,1	6,16-8,1	8,16	7,1	7,32-10,1	10,32	
8	6,1	6,16		7,1	7,16		8,1	8,32-9,1	9,32	

Fig.3.7 Il flusso delle informazioni in uscita dai gruppi di conversione

dati riguardanti il calorimetro elettromagnetico. Ad esempio sul terzo cavo di figura 3.10 la sequenza temporale delle informazioni è la seguente :

$$\begin{aligned}
 & \text{HAD A}(\theta_4\phi_1 \dots \theta_4\phi_{16} \rightarrow \theta_8\phi_1 \dots \theta_8\phi_{16}) \rightarrow \\
 & \rightarrow \text{HAD B}(\theta_5\phi_1 \dots \theta_5\phi_{16} \rightarrow \theta_9\phi_1 \dots \theta_9\phi_{16}) \rightarrow \\
 & \rightarrow \text{ELM}(\theta_6\phi_1 \dots \theta_6\phi_{32} \rightarrow \theta_{11}\phi_1 \rightarrow \theta_{11}\phi_{32})
 \end{aligned}$$

e quindi in un cavo vengono trasmesse in sequenza al massimo 128 parole delle 896 corrispondenti all'informazione generata nel calorimetro.

Dato che le strutture parallele sono molto più costose ma più veloci di quelle sequenziali, questa struttura ibrida permette un buon compromesso tra costo e prestazioni; infatti il tempo di trasferimento ottenuto con gli 8 cavi è $128 \times 60 \text{ ns} = 7.680 \mu\text{s}$ e siccome essa inizia alla fine delle conversioni

dei segnali provenienti dal calorimetro adronico, cioè dopo $5.6 \mu s$ dalla collisione, e si ferma per $0.56 \mu s$ per attendere la fine delle conversioni di quelli del calorimetro elettromagnetico, servono in totale $13.74 \mu s$ per trasferire e convertire tutte le informazioni ovvero poco più della metà del tempo a disposizione tra le due interazioni. La scelta di avere 8 cavi di uscita dai gruppi di conversione non è quindi casuale : infatti utilizzandone solo 4 occorrerebbero per la lettura $20.8 \mu s$, decisamente troppi in confronto ai $22 \mu s$ a disposizione; con 16 invece ne basterebbero $11.8 \mu s$ tempo che non comporta un grande miglioramento ma una successiva elaborazione con grado di parallelismo maggiore e quindi l'utilizzo di un maggior numero di moduli che farebbe quasi raddoppiare il costo del filtro.

Se LEP in futuro dovesse funzionare con $8 + 8$ pacchetti il tempo di calcolo disponibile per il filtro in energia risulterebbe essere di soli $11 \mu s$ e l'unica soluzione possibile sarebbe sostituire gli attuali ADC con dei convertitori molto più veloci.

La parte di elaborazione del filtro in energia accede ai dati che transitano lungo gli 8 cavi per effettuare le somme per riga e per colonna e quindi calcolare i 6 vettori menzionati precedentemente. Con un sommatore per cavo, che effettua l'operazione di somma ed accumula, si eseguono le somme in ϕ , mentre con dei sommatore in cascata quelle in θ . Le somme una volta eseguite vengono memorizzate sequenzialmente in memorie FIFO (moduli Data Stack della LeCroy), che permettono di memorizzarle per poter risin-

cronizzare le ulteriori elaborazioni che avvengono in tempi successivi. Per effettuare le somme e le somme ed accumula in 60 ns è stata progettata a Roma una ALU [1], unità logico-aritmetica. L'ALU utilizzata insieme alla Data Stack ed alla Memory Look up Unit (MLU), una RAM a sola lettura che permette se opportunamente programmata di eseguire operazioni logiche ed aritmetiche complesse, consente di realizzare tutti gli algoritmi della parte di elaborazione del filtro in energia.

3.3.3 Le temporizzazioni

Per poter controllare il flusso di dati attraverso tutta la cascata di moduli è necessario farli lavorare in maniera sincrona generando opportuni segnali di sincronismo con una fonte esterna. Poichè è desiderabile che le temporizzazioni siano flessibili ed adattabili a possibili cambiamenti futuri della struttura dello *hardware processor*, tale fonte deve essere programmabile; proprio per questo per la sua realizzazione sono stati costruiti due moduli: un generatore di impulsi con caratteristiche speciali presso il Laboratorio di Elettronica dell'INFN di Roma e, presso il laboratorio LAPP di Annecy, una particolare memoria programmabile a lettura sequenziale che può fornire o no impulsi in uscita.

Per la costruzione del filtro in energia si utilizzeranno i seguenti moduli:

1. FERA ADC modello 4300 della Lecroy Research Systems
2. FERA Driver LRS 4301
3. Memory Lookup Unit LRS 2372

4. Data Stack LRS 2375
5. Arithmetic and Logic Unit INFN ROMA
6. Bus Switch INFN ROMA
7. Bus Splitter INFN ROMA
8. Delay and Clock Generator INFN ROMA
9. ECLine Driver LAPP e ENERTEC

Al fine di poter comprendere il funzionamento del filtro in energia descritto in seguito si fornisce una breve descrizione del funzionamento dei singoli moduli (i dati tecnici dettagliati sono riportati nell'appendice A) e tra parentesi si indica la sigla che verrà usata in seguito per riferirsi a ciascun tipo di modulo:

FERA ADC (FERA): si tratta di un modulo che contiene 16 canali di conversione analogico digitale ad integrazione di carica la cui sensibilità è di 0.25 pC per bit. Il tempo di conversione è pari a 8.5 μ s per 11 bit e a 4.8 μ s per 10 bit; la lettura sequenziale dei 16 dati su connettore frontale può avvenire con un periodo minimo di 100 ns. Per poter collegare su un unico cavo le uscite differenziali di più FERA si utilizza il **Fera driver (FD)** che, oltre che adattare i segnali, si occupa anche di distribuire i sincronismi ai vari FERA.

Memory Lookup Unit (MLU): è una RAM di 64 kilobit con un tempo di accesso di 60 ns e può essere scritta solo via CAMAC "dataway". Su pannello frontale ha un ingresso ed un'uscita con protocollo ECL differen-

ziale: se all'ingresso si invia l'indirizzo, dopo un ritardo di 60 ns, si ottiene in uscita il contenuto della locazione indirizzata. Una volta programmata con un'opportuna tabella può effettuare una qualsiasi operazione logica o aritmetica: basterà caricare la memoria scrivendo nella locazione i il risultato dell'operazione $f(i)$.

Data Stack (DS): è una memoria FIFO (First In First Out) di 256 parole di 16 bit dotata di puntatori di scrittura e lettura indipendenti. Ha un tempo di lettura di 35 ns ed un tempo di scrittura di 50 ns.

Arithmetic and Logic Unit (ALU): è un modulo che può effettuare operazioni logiche ed aritmetiche su 2 parole di 16 bit. La selezione di una tra le 16 operazioni diverse disponibili, tra cui somme, sottrazioni e somme ed accumula, avviene mediante un codice operativo. I dati ed il codice operativo possono essere cambiati ogni 40 ns ed il tempo di ritardo è pari a 50 ns.

Bus Switch (BSW): è un'unità dotata di due ingressi ed una uscita dati. Con l'uso dei due registri si possono memorizzare le informazioni in ingresso e si può inoltre selezionare in uscita il contenuto di uno dei registri mediante un segnale di comando (A/B switch). Ambedue le uscite differenziali dello stesso segnale possono essere portate a livello logico 0 per cui è possibile connettere insieme più uscite di vari moduli.

Bus splitter (BSP): permette di duplicare il "bus" dati che si trova al suo ingresso in due uscite con un ritardo di 10 ns.

Delay and Clock Generator (DCG): è un generatore di impulsi con

4 uscite il cui periodo, ritardo e numero è programmabile via software.

Ecline Driver (ED): questo modulo consente di generare una serie impulsi a tempi prestabiliti solamente scrivendo prima delle tabelle nella sua memoria e poi rileggendola sequenzialmente. Viene perciò utilizzato nel filtro come generatore dei segnali di sincronismo da fornire ai vari moduli.

3.4 Lettura dei Fera

Assumendo che la collisione avvenga al tempo $t = 0$ bisogna attendere un certo tempo, diverso per i due calorimetri, prima di poter iniziare le conversioni: oltre a quello di propagazione dei segnali lungo i cavi pari a $0.4 \mu s$, bisogna tener conto del tempo necessario per eseguirne l'elaborazione analogica che risulta essere di $0.4 \mu s$ per il calorimetro adronico e di $1 \mu s$ per quello elettromagnetico. Quindi al tempo $t = 5.6 \mu s$ può iniziare la lettura delle matrici HAD-A, HAD-B dai gruppi di conversione e al tempo $t = 9.9 \mu s$ quella della matrice ELM.

È stato già visto (par. 3.3.2) che sono previsti 9 gruppi di conversione. Come si può notare dalla figura 3.8 il primo gruppo (gruppo 0) elabora i dati generati dal solo rivelatore di luminosità mentre il gruppo 1 quelli di una parte del calorimetro elettromagnetico. Gli altri sette gruppi elaborano i dati sia del calorimetro adronico che di quello elettromagnetico.

Si consideri il gruppo 6, mostrato in figura 3.9, che elabora i dati riportati come esempio nel paragrafo 3.3.2. Al tempo $t = 5.6 \mu s$ si deve iniziare la

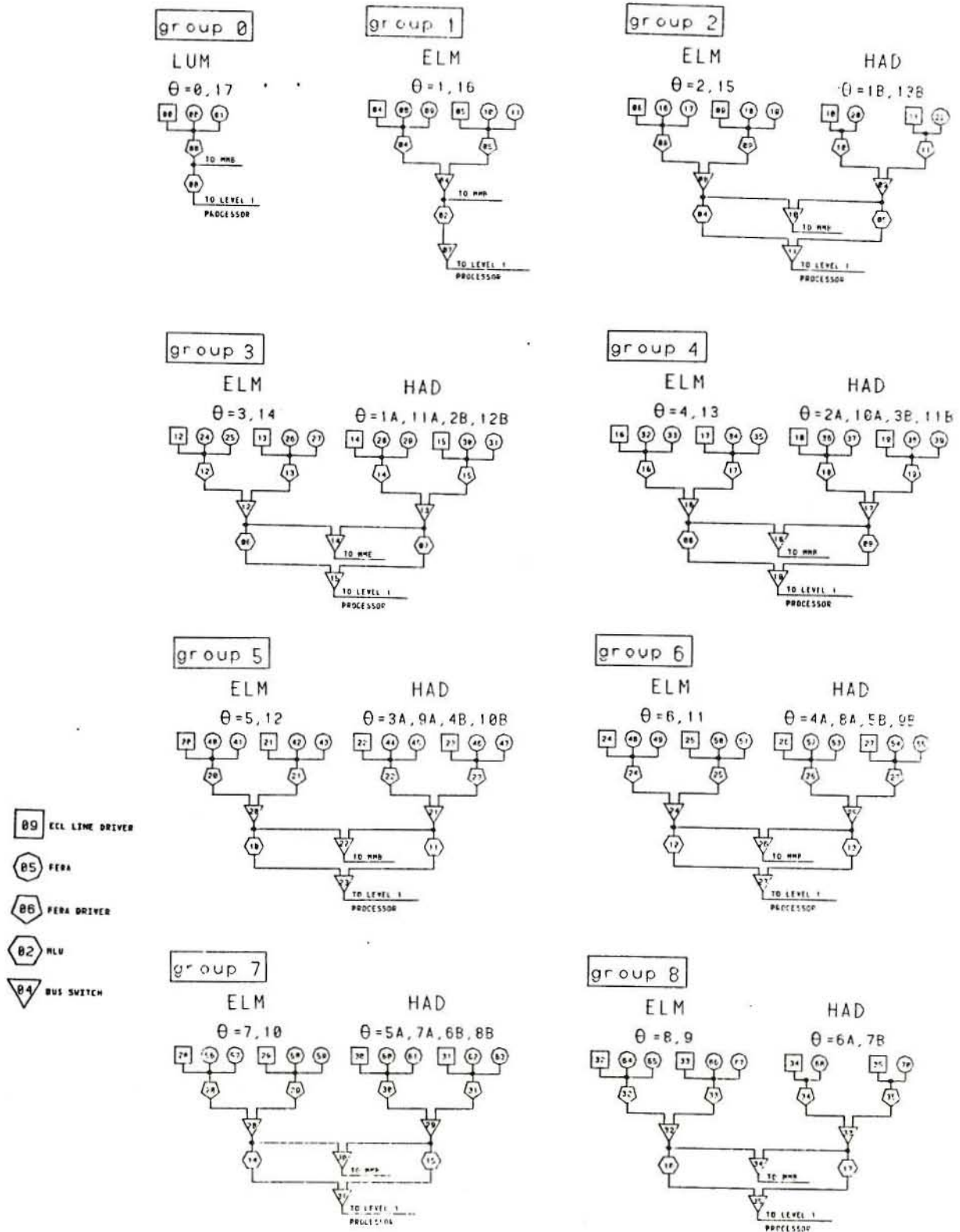


Fig. 3.8 I 9 gruppi di conversione

group 6

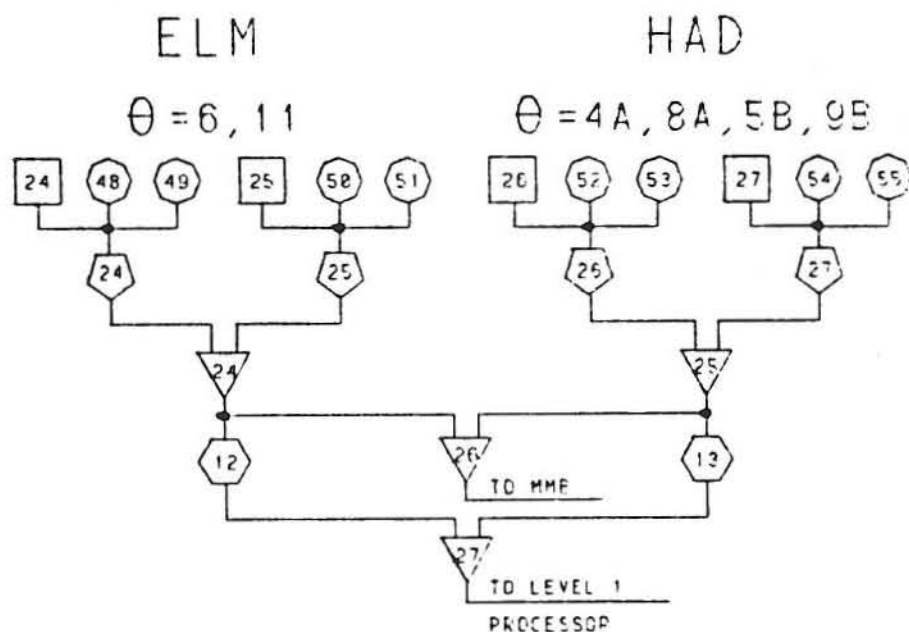


Fig. 3.9 il gruppo 6

lettura nei Fera 52, 53, 54, 55. Mandando 2 sincronismi con periodo di 120 ns, ritardati tra loro di 60 ns, ai Fera Driver 25 e 27, i 32×2 dati si presenteranno sequenzialmente agli ingressi del BSW 25.

Inviando due segnali di sincronismo Strobe A e Strobe B ogni 120 ns come mostrato in figura 3.10, si immagazzinano le informazioni sui registri di ingresso A e B del BSW 25 e, con il comando A/B Switch, si commutano alternativamente sull'uscita i contenuti dei due registri. In questo modo le informazioni ottenute cambieranno ogni 60 ns. L'uscita del BSW 25 è collegata agli ingressi di una MLU e di un nuovo BSW. Quest'ultimo permette di inviare in un primo momento le informazioni riguardanti il calorimetro adronico e poi quelle del calorimetro elettromagnetico alle MMB, memorie

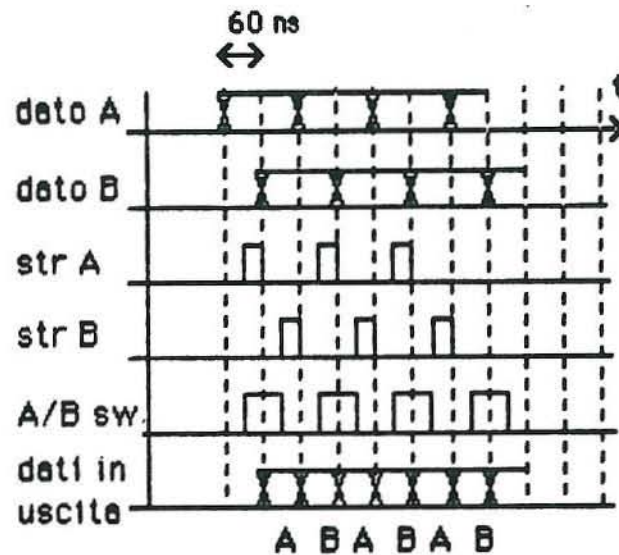


Fig. 3.10 Temporizzazioni del gruppo 6

tampone multievento del secondo livello. La MLU esegue varie operazioni di cui si parlerà in seguito, ed è seguita da un terzo BSW che si occupa di commutare sul medesimo cavo i dati provenienti dai due calorimetri.

La lettura dei dati del calorimetro adronico termina dopo un tempo pari a $t = 5.6 \mu s + 60 ns \times 64 = 9.44 \mu s$ e quindi la lettura di quelli del calorimetro elettromagnetico può iniziare, senza sovrapposizioni temporali, al tempo previsto $t = 9.9 \mu s$.

Il tempo necessario per leggere tutti i dati risulta essere di $13.74 \mu s$.

Come visto in precedenza i dati vengono elaborati da una MLU che esegue le seguenti operazioni :

1. Effettua l'equalizzazione dei guadagni dei Fera. Dalle caratteristiche

tecniche di questi moduli risulta che la pendenza delle rette di conversione (guadagno) può variare tra i vari canali del 3%. Per ottenere una precisione maggiore e pari all'1.5% si misurano i guadagni di tutti i convertitori, se ne calcola il valor medio e si moltiplicano per un fattore 1.015 le informazioni fornite da quei canali che hanno guadagni minori del valor medio. Per fare ciò si programma la MLU in maniera che se un bit di controllo è posto ad 1 o 0 aggiungerà o meno al dato in ingresso l'1.5% del suo valore.

2. Equalizza la scala in energia delle informazioni numeriche. Per i segnali del calorimetro adronico si usa un convertitore a 10 bit mentre per quelli dell'elettromagnetico uno ad 11 bit. È quindi necessario moltiplicare i primi per 2 affinché per tutti sia usata la stessa scala. All'uscita della MLU i dati saranno formati tutti da parole di 11 bit.
3. Sopprime il rumore coerente. Se tutti i canali dei segnali del filtro hanno un rumore comune e coerente, questo può arrivare a valori considerevoli quando vengono sommati insieme tutti i canali. È allora possibile introdurre una soglia per sopprimere questo rumore nella MLU, cioè l'uscita della MLU sarà zero se l'ingresso non supererà un certo valore minimo.
4. Segnala al filtro di conteggio delle tracce che l'energia depositata in un segmento è superiore ad un dato valore di soglia. Dato che ogni gruppo di Fera converte dati che corrispondono a determinati valori di θ simmetrici rispetto al centro del rivelatore, allora la soglia può essere

funzione dell'angolo programmando opportunamente ed in modo diverso le varie MLU dei singoli gruppi di conversione. Se l'energia è superiore a tale soglia la MLU porterà ad 1 il bit più significativo (bit 15) che sarà utilizzato poi per il conteggio delle tracce.

Inoltre c'è da tener presente che per quanto riguarda il calorimetro adronico un bit all'ingresso della MLU indicherà se i dati in ingresso provengono dallo strato A o B: si potrà così scegliere soglie diverse per i due diversi strati. In figura 3.8 si può notare che nei gruppi di conversione accanto ad ogni coppia di Fera è posto, con uscita sulla stessa linea dati, un modulo Ecline Driver. Questa unità si rende necessaria come sorgente di informazioni quando si vuole provare il funzionamento di tutto il processore: in questo caso non si generano gli impulsi necessari per la conversione dei Fera ma si manderanno dei segnali ai vari Ecline Driver per ottenere in uscita dei dati di simulazione e prova precedentemente scritti nelle memorie in essi contenute.

3.5 Schema a blocchi della parte di elaborazione numerica

Lo schema generale della logica del filtro in energia è mostrato nelle figure 3.11 e 3.12. Si possono notare in alto gli $8 + 1$ cavi in cui scorrono le informazioni riguardanti i calorimetri e il rivelatore di luminosità. Accanto ad ogni cavo viene riportata la sequenza in θ delle informazioni ; ad esempio sul secondo cavo dall'alto si legge : HAD 5,7A 6,8B ed ELM 7,10. Ciò significa che in quel cavo fluiscono i 16×2 dati relativi alle righe θ_5 e θ_7 della

matrice HAD-A, poi i 16×2 relativi a quelle θ_6 e θ_8 della matrice HAD-B ed infine le 32×2 informazioni θ_7 e θ_{10} del calorimetro elettromagnetico.

Calcolo dei vettori $R\theta$ (somme in ϕ): Per eseguire le somme in ϕ viene connessa una ALU che effettua la somma ed accumula (SA) ad ognuno dei primi otto cavi (fig. 3.13). Il quindicesimo bit non viene collegato in quanto è utilizzato per il conteggio delle tracce.

All'istante $t = 5.9\mu s$ dalla collisione dei pacchetti e^+e^- , i dati in ordine di ϕ crescente di 6 delle righe θ della matrice HAD-A cominciano ad arrivare sequenzialmente alle ALU le quali effettuano la somma ed accumula. Dopo 16×60 ns quindi a $t = 6.86\mu s$ viene terminata la prima parte delle somme in ϕ e i risultati, ovvero le prime 6 componenti del vettore ($HAD-A(R\theta_1) \dots HAD-A(R\theta_6)$), vengono memorizzate nei 4 BSW sottostanti. A questo punto le ALU, azzerate, ricominciano a sommare i dati $\phi_1 \dots \phi_{16}$ relativi alle righe $\theta_7 \dots \theta_{11}$. Mentre le ALU effettuano questo calcolo, le 6 somme vengono lette e trasferite nella DS HAD $R\theta$ -A (DS 11) in cui vengono memorizzate e nelle ALU che effettuano i calcoli dei bilanci energetici. Una volta terminate le somme relative alle righe $\theta_7 \dots \theta_{11}$ anche questi risultati vengono memorizzati nei BSW e poi inviati nelle stesse ALU e nella DS 11.

Con le stesse modalità vengono effettuati i calcoli sugli elementi delle matrici HAD-B ed ELM; gli altri due vettori risultanti vengono memorizzati rispettivamente nelle DS HAD- $R\theta$ -B (DS 12) e ELM- $R\theta$ (DS 1 e 2), per venir poi utilizzati dal filtro di *cluster* e dal filtro di fotone singolo.

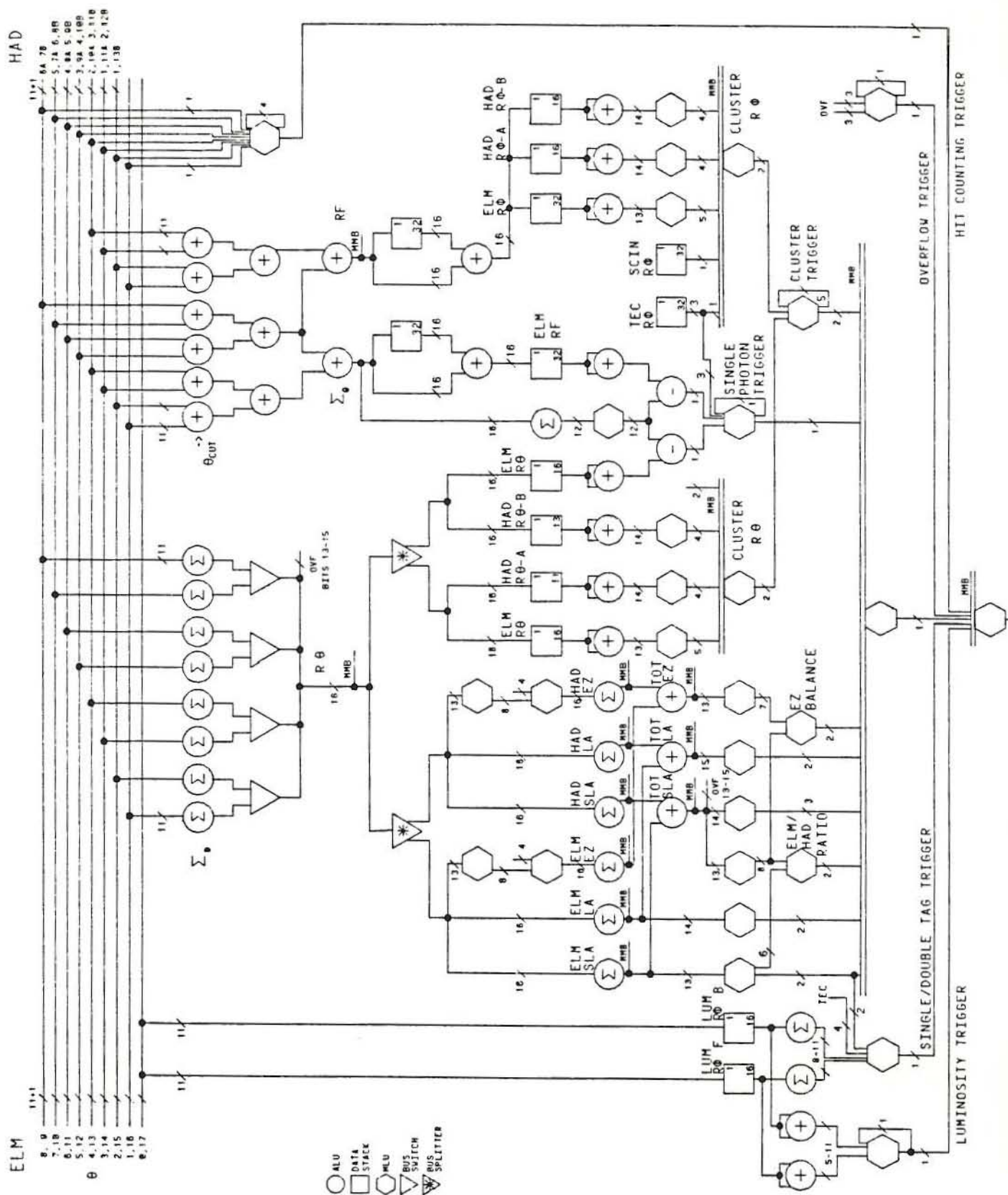


Fig. 3.11 Schema della logica del filtro

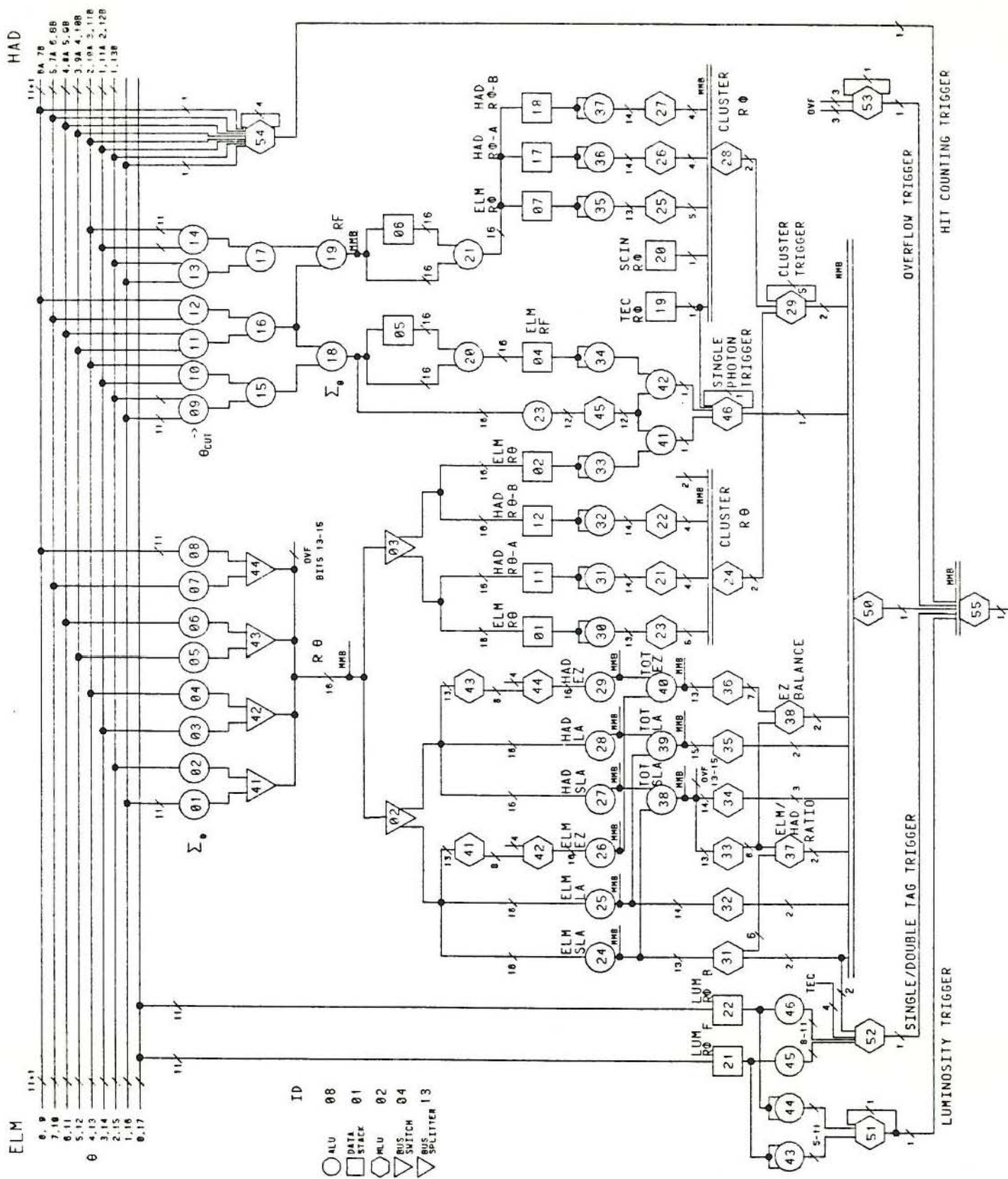


Fig. 3.12 Schema della logica del filtro con numerazione

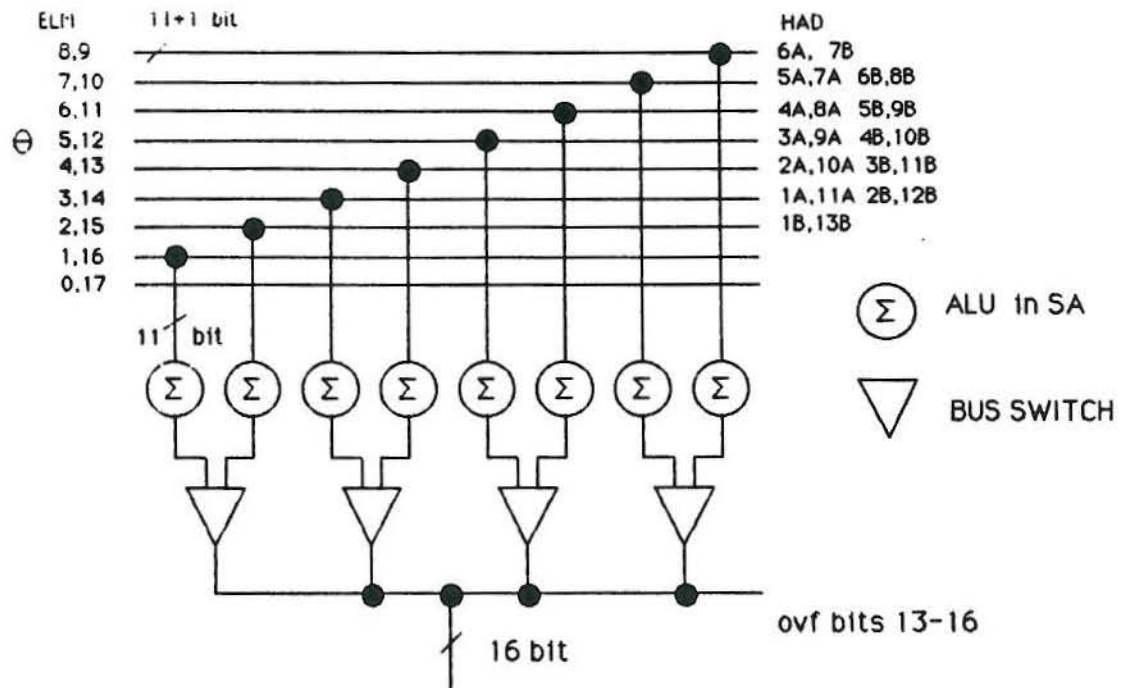


Fig. 3.13 Schema delle somme in ϕ

Calcolo dei vettori $R\phi$ (somme in θ): Poichè ogni 60 ns sono presenti contemporaneamente sugli 8 cavi le informazioni relative alla prima o la seconda metà di ogni colonna ϕ_i delle matrici, si può sommare in θ mentre si effettuano le somme in ϕ . Ad esempio per il calorimetro elettromagnetico scorrono sui cavi prima le informazioni delle righe $\theta_1 \dots \theta_8$ e poi quelle relative a $\theta_9 \dots \theta_{16}$; perciò, utilizzando delle ALU in cascata (ALU 9 ÷ 12 e 15, 16, 18) come in figura 3.14, si ottengono ogni 60 ns le somme degli elementi $ELM(\phi_i, \theta_1) + \dots + ELM(\phi_i, \theta_8)$ e poi $ELM(\phi_i, \theta_9) + \dots + ELM(\phi_i, \theta_{16})$. Per ottenere il vettore ELM $R\phi$ è necessario memorizzare le prime somme in una DS e poi sommarle con le seconde. Per far ciò si utilizzano due moduli: la DS 5 e la ALU 20. Nella DS 5 vengono memorizzate le prime 32 somme parziali

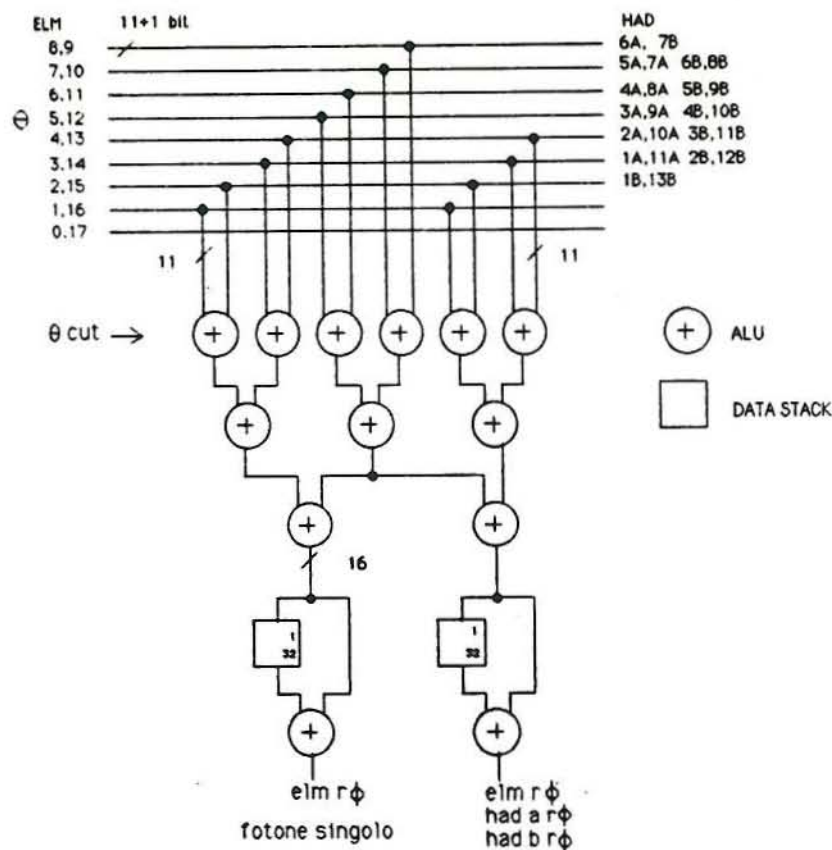


Fig. 3.14 schema delle somme su θ

che poi, rilette, vengono sommate con le corrispondenti somme parziali in ϕ tramite la ALU 20.

La ALU (23) (fig. 3.11), effettuando la somma ed accumula, calcola l'energia elettromagnetica totale utilizzata poi dal filtro di fotone singolo. Se si volessero escludere dei valori di θ dalle somme basterà non inviare i sincronismi alle corrispondenti ALU che perciò non elaboreranno il relativo dato.

Poichè per il filtro di *cluster* è utile poter escludere dalle somme valori di θ diversi da quelli del filtro di fotone singolo, i dati, provenienti lungo i 4 cavi corrispondenti ad 8 piccoli valori dell'angolo θ , sono mandati ad altre 3 ALU (13,14 e 17) che possono avere una diversa sequenza di sincronismi.

Le somme in θ , con questa diversa selezione degli angoli, vengono elaborate come quelle precedenti dalla DS 6 e dalla ALU 21 ed infine memorizzate nelle DS ELM- $R\phi$, HAD- $R\phi$ -A, HAD- $R\phi$ -B.

Quindi a $t = 14.64 \mu s$ dall'interazione, come si vedrà con più dettaglio in seguito, tutte le informazioni delle matrici sono state elaborate, si sono memorizzati i 6 vettori nelle DS per poi utilizzarli nei filtri di *cluster* e fotone singolo e si sono calcolati tutti i bilanci energetici.

3.5.1 Filtro in energia

Energia totale : Le 11 componenti del vettore HAD-A $R\theta$ e le 13 di HAD-B $R\theta$ provenienti dai BSW 41 ÷ 44, sono inviate all'ALU (27) che, effettuando la somma ed accumula, fornisce l'energia totale rilasciata nel calorimetro adronico. Le componenti di ELM $R\theta$ invece vengono mandate ad un'altra ALU (24) anch'essa in somma ed accumula che, una volta terminato il calcolo, invierà il risultato, ovvero l'energia elettromagnetica totale, alla MLU 31 tramite la quale viene confrontata con 3 diversi valori di soglia: per questo motivo il risultato fornito dalla MLU 31 è codificato in 2 bit. Nella ALU 38 si sommano i risultati delle ALU 27 e 24 ottenendo così l'energia totale rilasciata nel rivelatore che viene confrontata, tramite la MLU 34, a ben 7 valori di soglia, generando così un risultato formato da 3 cifre binarie.

Energia a grandi angoli : Nello stesso tempo le componenti dei vettori $R\theta$ sono inviate anche a due ALU (25 e 28) poste in somma ed accumula che,

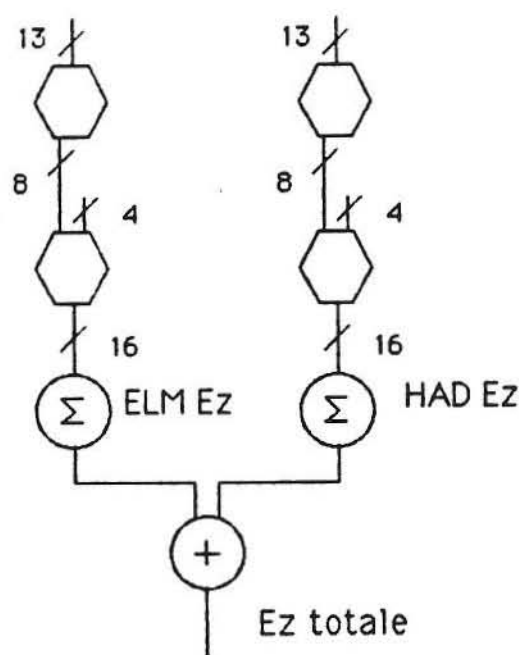


Fig. 3.15 Calcolo del bilancio energetico longitudinale

essendo sincronizzate solo per i θ grandi (ELM-LA, HAD-LA), forniscono l'energia a grandi angoli dei due calorimetri. Nella ALU 39 si sommano queste 2 energie ottenendo l'energia totale a grandi angoli. Anche per questa energia sono fissate 3 soglie programmando opportunamente la MLU 35.

Bilancio energetico longitudinale : Le componenti dei vettori $R\theta$ sono inoltre inviate a due cascate uguali di moduli (fig. 3.15). Ciascun insieme viene utilizzato rispettivamente per il calcolo del bilanciamento d'energia avanti-indietro per il calorimetro adronico e per quello elettromagnetico. Nella cascata la prima MLU serve per approssimare i dati da 13 a 8 bit, la seconda riceve questi dati insieme al valore di θ (codificato in 4 bit da un Ecline driver) e calcola, opportunamente programmata, $E(\theta_i) \cdot \cos(\theta_i)$,

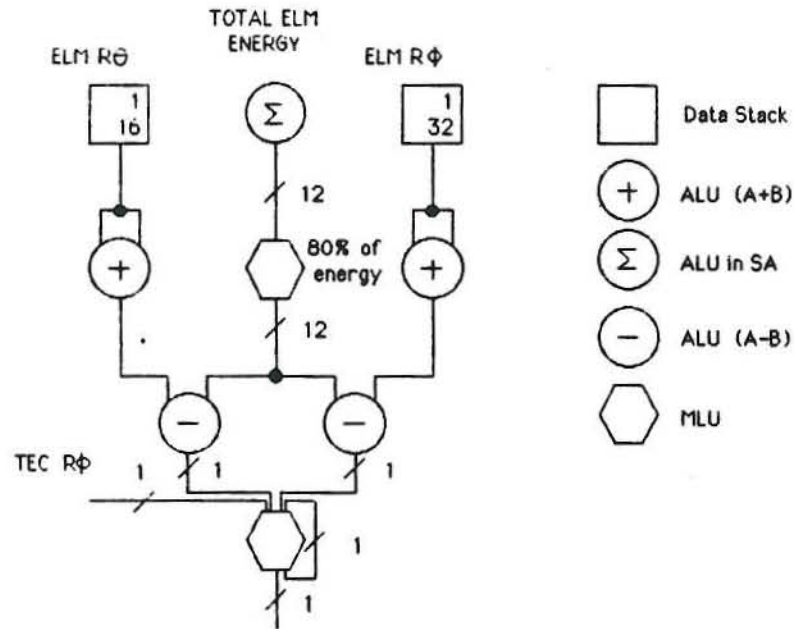


Fig. 3.16 Il filtro di fotone singolo

infine la ALU, che effettua la somma ed accumula, fornisce il bilancio energetico longitudinale. Quest'ultima ALU per effettuare correttamente la somma deve ricevere le informazioni codificate in complemento a 2 per valori di $\theta > \frac{\pi}{2}$. Nel modulo ALU 40 viene effettuato il bilancio energetico longitudinale totale che viene poi confrontato, insieme all'energia totale, con 3 soglie differenti tramite la MLU 38.

3.5.2 filtro di fotone singolo

Una volta memorizzato il vettore ELM $R\theta$ nella DS 2, ELM $R\phi$ nella DS 4 e calcolata l'energia elettromagnetica totale nell' ALU 23, può iniziare l'elaborazione del filtro di fotone singolo (fig. 3.16).

Prima di tutto si invia il risultato della ALU 23 nella MLU 45 che ne

calcola una certa percentuale (l'80%). Se il valore dell'energia è particolarmente piccolo verrà generato per l'energia un valore talmente alto tale da disabilitare il filtro di fotone singolo in modo da evitare di accettare eventi di fondo. Le componenti di ELM $R\theta$ contenute nella DS 2 vengono lette e inviate sequenzialmente alla ALU 33 che, avendo i due ingressi connessi in comune, può eseguire le somme di due componenti contigue, ovvero di due righe adiacenti. Infatti i dati vengono caricati nella ALU con la seguente sequenza:

- i. Il primo dato nel registro A,
 - ii. il secondo dato nel registro B ottenendo in uscita, dopo 50 ns, la somme delle due righe adiacenti ELM $R\theta_1$ e ELM $R\theta_2$
 - iii. il terzo dato in A ottenendo la somma di ELM $R\theta_2$ con ELM $R\theta_3$
 - iv. il quarto dato in B ottenendo la somma di ELM $R\theta_3$ con ELM $R\theta_4$
- e così via.

L' ALU 41 sottrae dalla frazione dell'energia totale calcolata con la MLU 45 le somme delle righe adiacenti. Se si ottiene un risultato negativo significa che l'energia è in massima parte concentrata in quelle due righe. Poichè un valore negativo all'uscita di una ALU è espresso in complemento a due, basterà controllare, tramite la MLU (46) se, tra i risultati della sottrazione uno dei bit non significativi (ad esempio il bit 15) è pari ad 1. Una operazione identica viene effettuata per le colonne ϕ (somme in θ). Per escludere il caso in cui il deposito di energia è stato generato da un elettrone, bisogna verificare

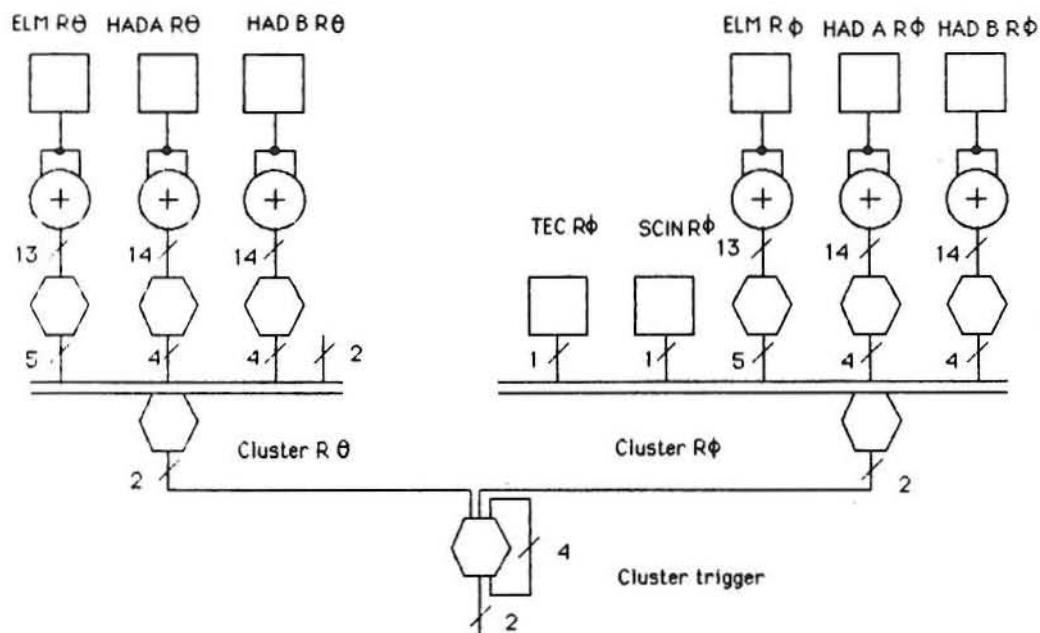


Fig. 3.17 Il filtro di cluster

che nello stesso tempo non vi siano tracce nella TEC; ciò viene fatto inviando alla MLU 46 una cifra binaria prelevata dalla DS 19 in cui sono memorizzati in maniera opportuna i dati provenienti della TEC. Si avrà un evento di fotone singolo quando in ambedue le ALU (41 e 42) si sarà registrato almeno un risultato negativo e non ci saranno tracce nella TEC.

3.5.3 filtro di clusters

Per decidere la presenza di *clusters* nel piano $R\theta$, vengono utilizzati i vettori $ELM-R\theta$, $HAD-A-R\theta$ e $HAD-B-R\theta$ contenuti nelle DS 1, 11 e 12 (fig. 3.17). Verranno sommate, nella stessa maniera del filtro di fotone singolo, tramite le ALU 30, 31 e 32 righe adiacenti della matrice del calorimetro elettromagnetico e se sarà necessario anche di quelle del calorimetro adronico.

Le somme effettuate dalle ALU vengono approssimate a 5 bit dalla MLU 23 per il vettore ELM $R\theta$ e a 4 bit per i vettori HAD-A $R\theta$ e HAD-B $R\theta$ (MLU 21 e 22). I risultanti (13 bit) sono analizzati tramite la MLU (24) che decide sulla presenza o meno e sul tipo di *clusters* controllando il valore dell'energia ottenuto nelle 3 somme. Altri due bit d'ingresso, sempre in quest'ultima MLU, possono venir utilizzati per selezionare 4 differenti regioni in θ : in questo modo il criterio per la ricerca dei *clusters* potrà essere funzione dell'angolo. Dividendo in θ il calorimetro in 4 spicchi le quattro configurazioni (0, 1, 2, 3) delle due cifre binarie in ingresso corrispondono rispettivamente all'intero cilindro, primo spicchio, secondo e terzo spicchio, quarto spicchio. Per poter definire più di un tipo di *cluster* la decisione viene codificata in 2 bit.

Una logica simile è utilizzata nel piano $R\phi$ ma in questo caso si utilizzano anche le informazioni provenienti dagli scintillatori e dalla TEC. Infatti vengono inviati alla MLU 28 : 1 bit dalla DS (19) TEC $R\phi$, 1 bit dalla DS (20) SCIN $R\phi$, 5 bit delle somme di componenti contigue di ELM- $R\phi$, 4 bit da quelle di HAD- $R\phi$ -A e di HAD- $R\phi$ -B. I 2 + 2 bit contenenti l'informazione riguardo ai *clusters* nei due piani sono infine elaborate dalla MLU 29 che, avendo 4 bit dell'uscita connessi all'ingresso, può contare il numero di *clusters* trovati e dal loro numero totale prendere la decisione finale del filtro di *cluster* che viene codificata in due bit.

3.5.4 Conteggio delle tracce ad alta energia

Le MLU dei gruppi di lettura dei Fera sono programmate in maniera che il bit 15 viene portato ad uno se l'energia del segmento è al di sopra di un certo valore. Il conteggio delle tracce viene effettuato con una sola MLU (54) alla quale è connesso il quindicesimo bit di ogni "bus": contando quante volte questo bit è a uno si contano le tracce ad alta energia nei calorimetri. Un bit di controllo indicherà se la traccia è stata trovata nel calorimetro adronico (0) o elettromagnetico (1), mentre 4 bit, 2 per il calorimetro elettromagnetico e 2 per quello adronico serviranno per contare le tracce. La decisione finale consiste in un solo bit.

3.5.5 Filtro di luminosità

Per il filtro di luminosità, i 16×2 dati vengono memorizzati in due Data stack: i 16 riguardanti il rivelatore anteriore nella DS 21, e i 16 per quello posteriore nella DS 22. Le ALU 43 e 44 sommano con la solita modalità due segmenti in ϕ adiacenti; i sette bit più significativi del risultato sono inviati ad una MLU (51) che per identificare gli eventi di Bhabha controlla se c'è stato un deposito di energia in due segmenti corrispondenti sia in avanti che indietro. Le ALU 45 e 46 invece calcolano l'energia totale rilasciata nei due rivelatori di luminosità. Queste due informazioni insieme all'energia totale, all'energia a grandi angoli e al bilancio E_z permettono tramite la MLU (52) di riconoscere eventi di *single/double tag*.

3.5.6 Filtro di "overflow"

Il filtro in energia è stato progettato in maniera da poter elaborare parole

di 13 bit, cui corrispondono 200 GeV, in quanto questo è il numero minimo di bit significativi all'ingresso delle MLU utilizzate. Quindi se l'energia totale di un evento supera i 200 GeV si è in presenza di un "overflow" che comporta degli errori nelle elaborazioni. Perciò si prelevano i bit 13,14 e 15 in due punti del filtro:

- i) all'uscita dei BSW 41÷44 per controllare un eventuale "overflow" durante il calcolo dei vettori $R\theta$,
- ii) all'uscita dell'ALU 38, dove l'"overflow" viene raggiunto se l'energia totale è maggiore di 200 GeV.

Analizzando queste informazioni tramite la MLU 53 si può riconoscere se è avvenuto un "overflow".

3.5.7 Dati inviati al filtro di secondo livello

I dati digitalizzati dai Fera e alcuni risultati intermedi e finali del filtro di primo livello vengono mandati, tramite alcuni Bus Switch, alle MMB (Cap.2) del filtro di secondo livello per effettuare calcoli più sofisticati e per controllare anche alcuni risultati preliminari del filtro al primo livello.

3.5.8 La decisione finale

Tutti i risultati delle varie operazioni dei filtri in energia, di *cluster* e fotone singolo sono analizzati nella MLU 50 che produce un risultato di un bit. I 16 bit dell'ingresso sono suddivisi in questo modo :

3 bit dall' energia totale ($ELM + HAD$) (MLU 34)

2 bit dall'energia totale a grandi angoli ($ELM + HAD$) (MLU 35)

- 2 bit dall'energia elettromagnetica totale (ELM) (MLU 31)
- 2 bit dall'energia elettromagnetica a grandi angoli (MLU 32)
- 2 bit dal confronto tra il bilancio E_z e l'energia totale (MLU 38)
- 2 bit dal rapporto delle energie $ELM/(ELM + HAD)$ (MLU 37)
- 1 bit dal filtro di fotone singolo (MLU 46)
- 2 bit dal filtro di *cluster* (MLU 29)

L'aver utilizzato soglie multiple per i vari bilanci energetici permetterà in futuro una notevole flessibilità nel gestire le varie condizioni di "trigger" programmando opportunamente la MLU 50.

La decisione finale (si/no) viene presa nella MLU 55 che produce l'OR delle seguenti 5 informazioni :

- 1 bit dal filtro di luminosità
- 1 bit dal filtro di single e double tag
- 1 bit dalle somme in energia e dal filtro di *cluster* e fotone singolo (risultato della MLU 50)
- 1 bit dal filtro di overflow
- 1 bit dal filtro di conteggio delle tracce.

3.6 Temporizzazione del filtro

In LEP la collisione dei 4 + 4 pacchetti di elettroni e positroni avviene ogni 22.231 μs e quindi il tempo di elaborazione del filtro di primo livello non deve superare tale valore. Ad ogni collisione una "scatola di controllo"

genera, a tempi diversi, 2 segnali: il CLR (*clear*) e il BG (*beam gate*). Questi segnali vengono utilizzati da parte dei convertitori e dei filtri di primo e secondo livello, sia per le operazioni di inizializzazione (CLR) che per il campionamento dei dati dell'evento (BG). Se al tempo $t = 0$ avviene la collisione a $t \simeq -1 \mu s$ si manda il CLR e a $t \simeq 0.5 \mu s$ il BG. Il BG è ritardato di circa $0.5 \mu s$ rispetto al momento in cui avviene la collisione in quanto il ritardo dei cavi tra il rivelatore e l'elettronica del primo livello è stimato essere $0.4 \mu s$ (80 mt. di cavi). Questi segnali vengono soppressi nel caso di evento accettato per tutto il di tempo in cui avvengono le conversioni e le memorizzazioni dei dati principali del sistema di acquisizione.

In particolare nel filtro di energia, il segnale CLR inizializza tutti i Fera, i generatori di sincronismi, e tutti i moduli che ne hanno bisogno (ALU in SA, Data Stacks, MLU), mentre dal BG con opportuni ritardi vengono generati i segnali di conversione ("gate") e di inizio per la lettura dei gruppi e l'elaborazione dei dati. Tutto l'*hardware processor* è governato da un insieme di segnali che possono essere suddivisi nelle seguenti 3 categorie [6]:

- Segnali S (Sincronismi) di durata pari a 30 ns
- Segnali C (Comandi : ad es. A/B switch nei Bus Switch) con durata 60 ns con il fronte di salita contemporaneo a quello dei sincronismi.
- Segnali D (Dati : ad es. θ_i nel calcolo di $E(\theta_i)\cos\theta_i$) con durata di 60 ns con il fronte di discesa contemporaneo a quello delle sincronismi.

Negli Ecline Driver possono venir memorizzate 256 parole di 16 bit cias-

cuna. Se, ad esempio, si memorizza sul bit 1 di varie parole poste in successive locazioni della memoria la seguente configurazione 1, 0, 0, 1, 0, 1, etc. rileggendole sequenzialmente con una temporizzazione esterna (CK) si può avere nel tempo una successione di cambiamenti di livello all'uscita propria del bit 1 come mostrato in figura 3.18a. Il modulo può anche funzionare in "gated mode" in cui i segnali in uscita sono tutti messi in AND con l'inverso del segnale di temporizzazione esterna generando così dei segnali come quelli riportati in figura 3.18b. È chiaro quindi che per generare i segnali S, C, e D si utilizza un unico modulo DCG (Delay and Clock Generator) come generatore principale di impulsi (*master clock*) e tre gruppi di Ecline Driver connessi a tre uscite indipendenti del DCG. Gli Ecline Driver preposti alla formazione dei segnali S dovranno essere posti in funzionamento "gated mode" mentre quelli che forniscono i segnali C riceveranno un treno di impulsi dal DCG ritardato di 30 ns rispetto a quello ricevuto dagli altri.

Si riassumono qui di seguito i vari tempi di elaborazione considerando $t = 0$ l'istante dell'interazione.

1. Lettura dei Fera.

La lettura dei dati convertiti che riguardano il calorimetro adronico inizia al tempo $t = 5600$ ns mentre quella del calorimetro elettromagnetico a $t = 9900$ ns. In ambedue i casi si leggono 64 dati e si deve considerare il ritardo di 5 moduli prima che i dati si presentino in uscita ai gruppi di conversione. Perciò la lettura termina al tempo $t = 9900 + 64 \times 60 +$

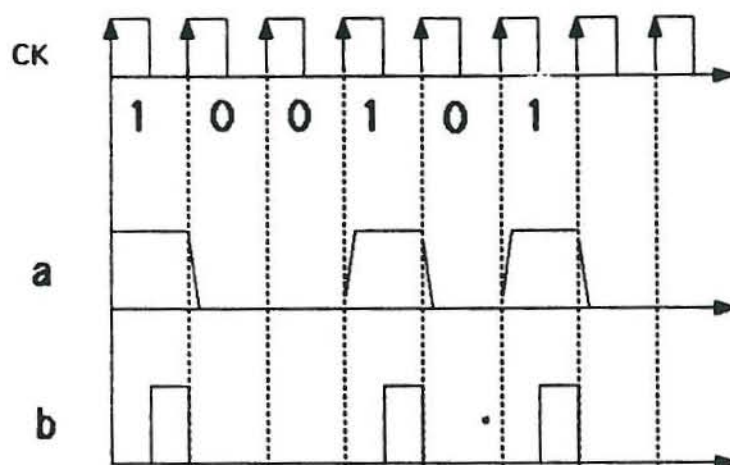


Fig. 3.18 I segnali in "gated mode" e "non gated mode"

$$5 \times 60 \text{ (ritardo moduli)} = 14040 \text{ ns.}$$

2. Somme in ϕ .

Al tempo $t = 14040 + 60 \text{ ns}$ sono terminate le somme in ϕ che vengono memorizzate nei BSW per 60 ns e poi lette in $8 \times 60 \text{ ns}$. Quindi l'ultimo dato dai BSW 41÷44 verrà prelevato al tempo $t = 14580 \text{ ns}$.

3. Somme in θ .

Dato che l'ultimo dato è presente nei "bus" al tempo $t = 14040 \text{ ns}$ e che la cascata di ALU che effettua le somme in θ è profonda 4 moduli, considerando anche il ritardo della DS, si ottiene che l'ultima somma in θ sarà memorizzata nella DS ELM- $R\phi$ al tempo $t = 14040 + 5 \times 60 = 14340 \text{ ns}$.

4. Bilanci energetici.

Tenendo conto dei ritardi dei moduli usati in cascata per ogni bilancio si possono calcolare i tempi per :

a) l'energia elettromagnetica totale (SLA) ed a grandi angoli (LA)

$$t = 14580 + 60 \times 2 = 14700 \text{ ns}$$

b) l'energia totale SLA e LA $t = 14580 + 3 \times 60 = 14760 \text{ ns}$

c) il rapporto $\frac{ELM}{ELM + HAD}$ $t = 14580 + 60 \times 4 = 14820 \text{ ns}$

d) il bilancio E_z $t = 14580 + 60 \times 6 = 14940 \text{ ns}$.

5. Filtro di *clusters*.

Sono necessari al massimo 32 impulsi per leggere le informazioni contenute nelle 6 DS del filtro di *cluster*. Siccome l'ultimo dato è memorizzato al tempo $t = 14580 + 60 = 14640 \text{ ns}$ solo da questo istante può iniziare la ricerca dei *clusters* che quindi termina al tempo $t = 14640 + 32 \times 60 + 4 \times 60 = 16800 \text{ ns}$ dove $32 \times 60 \text{ ns}$ è quello occorrente per la lettura della DS e $4 \times 60 \text{ ns}$ è il ritardo dei moduli.

6. Filtro di fotone singolo.

Anche in questo caso l'elaborazione inizia al tempo $t = 14640 \text{ ns}$. Si devono leggere 32 informazioni nella DS ELM $R\phi$ e ci sono poi 3 moduli in cascata perciò si avrà il risultato del filtro di fotone singolo al tempo $t = 14640 + 32 \times 60 + 3 \times 60 = 16740 \text{ ns}$.

7. Filtro di *single double tag* e luminosità. Inizia a $t = 9900 + 5 \times 60 = 10200 \text{ ns}$ ed occorrono 16×2 impulsi per scrivere e poi leggere ogni DS

del filtro. Il risultato è quindi disponibile al tempo $t = 10200 + 32 \times 60 + 2 \times 60 = 12240$ ns.

8. Overflow.

L'ultimo dato relativo al calcolo dell'energia totale che può generare un overflow è fornito al tempo $t = 14700$ ns a cui si deve aggiungere i 60 ns di ritardo della MLU 53.

9. Conteggio delle tracce.

Poichè dai Fera l'ultimo dato viene trasferito al tempo $t = 14040$ ns e tenendo conto del ritardo della MLU 54 l'elaborazione finisce al tempo $t = 14040 + 60 = 14100$ ns.

10. Decisione finale.

Da quanto detto l'elaborazione più lunga è effettuata dal filtro di *clusters* che termina al tempo $t = 16800$ ns, quindi la decisione finale, tenendo conto dei ritardi dovuti alle MLU 50 e 55, è presa dall'intero filtro in un tempo di $16800 + 2 \times 60 = 16920$ ns.

Al momento dell'installazione finale dell'intero filtro può essere utile per controllarne la temporizzazione avere una tabella con i tempi di inizio e fine di ogni processo come riportato in tabella 4.1.

Operazione	Inizio(μ s)	Fine(μ s)	Durata(μ s)
Collisione	0.0	-	-
Ritardo cavi HAD, ELM	-	-	0.4
sample/hold HAD	0.4	0.8	0.4
conversione FERA HAD	0.8	5.6	4.8
sample/hold ELM	0.4	1.4	1.0
conversione FERA ELM	1.4	9.9	8.5
lettura FERA HAD,ELM,LUM	5.6	14.04	8.44
calcolo energia totale	5.9	14.76	8.86
conteggio tracce	5.9	14.1	8.2
filtro di overflow	6.98	14.7	7.72
filtro di luminosità	9.9	12.24	2.34
filtro di single/double tag	9.9	12.24	2.34
filtro fotone singolo	14.64	16.74	2.1
filtro di cluster	14.64	16.8	2.16
decisione finale	-	16.92	-
tempo inutilizzato	16.92	21.6	4.68
clear,reset	21.6	22.6	1.0
prossima collisione	22.231	-	-

Tabella 4.1 : Stato del filtro in funzione del tempo

4 Moduli e problemi connessi ai sincronismi

4.1 Premessa

Il filtro di energia di primo livello é costituito da un *hardware processor* composto da piú di 250 moduli CAMAC utilizzati come elementi per realizzarne tutte le funzioni richieste. Ben 20 *crate* Camac verranno usati per contenere il filtro di energia e si stima che un migliaio di cavi saranno necessari per le connessioni tra i moduli. É importante, ai fini del perfetto funzionamento e della affidabilit  di un sistema cos  complesso, che ogni singolo modulo sia controllato nel suo funzionamento e caratterizzato mediante opportuni parametri da misurare in laboratorio.

4.2 Il modulo

Un modulo CAMAC che funziona utilizzando il protocollo ECL differenziale pu  essere schematizzato nella seguente maniera (fig. 4.1) :

- Un registro o due di ingresso (*latch*) in cui vengono memorizzati i dati tramite il fronte di salita di un segnale di sincronismo (*strobe*)
- Una rete combinatoria che esegue l'operazione
- Un'eventuale registro di uscita (*latch*) del dato

Il registro d'ingresso serve per "catturare" il dato presente alla porta frontale d'ingresso del modulo; in questo modo é sufficiente che il dato sia

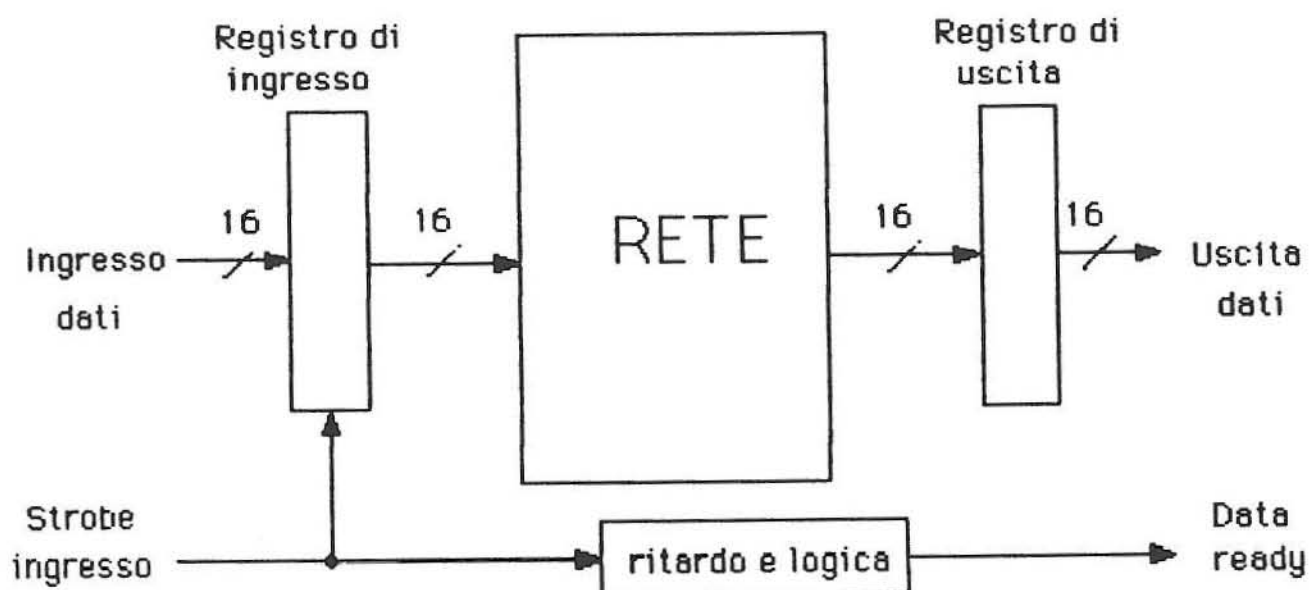


Fig. 4.1 Schema di un modulo Camac con bus ECL differenziale

stabile quei pochi ns (tempo di memorizzazione del *latch*) in cui il fronte di salita del sincronismo abilita il registro.

La rete esegue l'operazione in un certo tempo che può essere più o meno lungo a seconda della complessità dell'operazione stessa e della configurazione delle cifre binarie nel dato in ingresso. Quando l'operazione ha termine sull'uscita si presenterà il risultato, ma il passaggio dal risultato

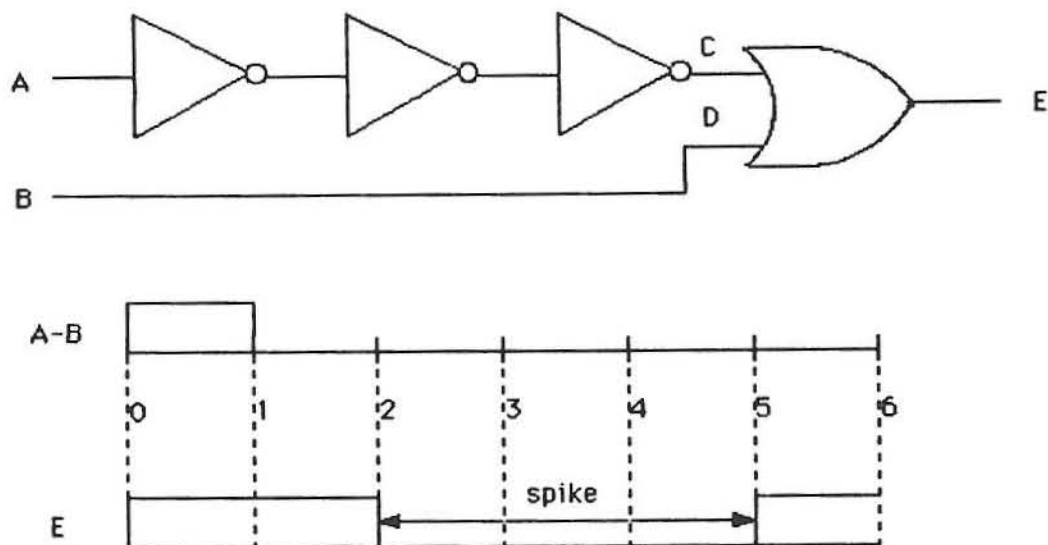


Fig. 4.2 Una semplice rete combinatoria e il suo diagramma temporale

preesistente a quello nuovo non sarà istantaneo ma ci sarà un periodo di tempo in cui il dato non sarà stabile in quanto saranno presenti degli impulsi dovuti ai diversi tempi di transito dei segnali lungo la rete. Per chiarire questo fatto facciamo un esempio. Si consideri la semplice rete combinatoria di Fig. 4.2: se si commutano a $t = 1$ ambedue gli ingressi da "1" a "0" logico si ha il seguente andamento temporale dell'uscita (consideriamo unitario il ritardo di ogni singola porta logica) :

tempo	A (input)	B (input)	C	D	E (output)
0	1	1	0	1	1
1	0	0	0	0	1
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	1	0	0
5	0	0	1	0	1

si vede chiaramente che c'è un periodo di tempo ($2 \leq t \leq 5$) in cui il risultato non è valido, fatto dovuto al ritardo intrinseco delle 3 porte NOT. Quello che si vedrebbe su un oscilloscopio sarebbe un'impulso (*spike*) della durata di 3 unità di tempo. Risulta chiaro anche che la presenza di questo cambiamento di livello spurio dipende dalla configurazione dei bit all'ingresso: infatti se facessimo commutare sempre per la stessa rete al tempo 1 l'ingresso A da 1 a 0 e quello B da 0 ad 1, l'andamento temporale sarebbe il seguente :

tempo	A (input)	B (input)	C	D	E (output)
0	1	0	0	0	0
1	0	1	0	1	0
2	0	1	0	1	1
3	0	1	0	1	1
4	0	0	1	1	1
5	0	0	1	1	1

In questo caso il dato è valido già a $t = 2$.

Per un modulo CAMAC la complessità delle reti logiche è di gran lunga maggiore e quindi il fatto di avere un periodo di tempo in cui il dato non è stabile ma oscilla tra i due stati logici è spiacevole ma perfettamente normale.

Inoltre per il fatto di far commutare molto velocemente le informazioni si possono creare degli accoppiamenti tra le linee dei singoli bit (*cross talk*); questo accade specialmente quando, in presenza di “bus”, si fanno cambiare tutte le cifre binarie tranne una da uno stato logico all’altro. Questo fatto mette ulteriormente in evidenza come il tempo di ritardo del modulo dipenda dalla configurazione dei bit in ingresso.

Per ovviare ai problemi esposti, il modulo stesso genera un segnale di “dato pronto” nel momento in cui il dato in uscita é ormai ben stabile. Questo segnale viene generato formando e ritardando il sincronismo di ingresso.

Per eliminare del tutto gli impulsi spuri in uscita, si é soliti mettere un “registro di uscita” che di nuovo memorizza il dato solo quando é stabile. Poiché l’uso di questo registro va a scapito della velocità di elaborazione, il suo utilizzo é limitato solo ai moduli particolarmente veloci in cui l’incremento di $5 \div 10$ ns sul tempo di elaborazione non comporta problemi. Anche in questo caso comunque viene generato il “dato pronto” per poter essere utilizzato come sincronismo d’ingresso per un modulo successivo.

4.3 Definizione di τ_1 e di τ_2

Abbiamo visto nel capitolo 3 che nel filtro di energia tutte le temporizzazioni vengono effettuate in maniera sincrona, cioè ci sarà una fonte che genera i sincronismi per tutti i moduli. Questo perché solo così si può avere la sincronizzazione di dati provenienti da diversi percorsi in determinati punti

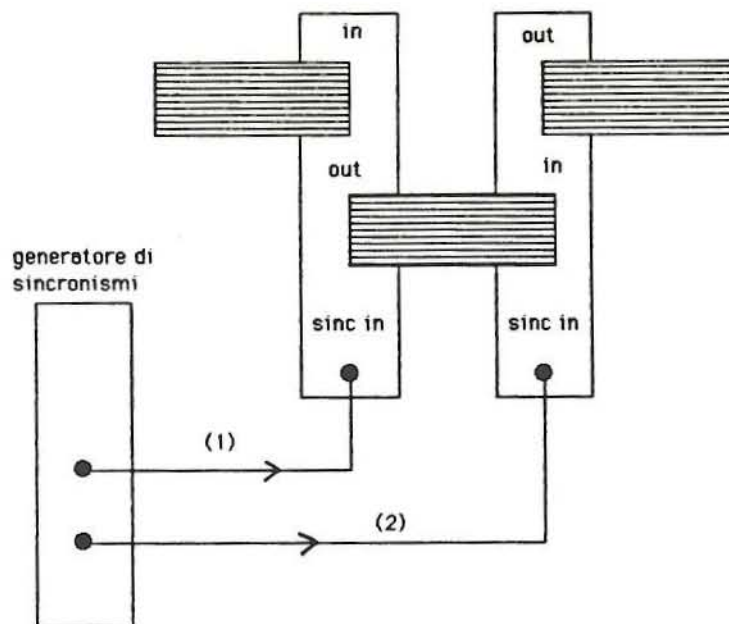


Fig. 4.3

del filtro. Quindi non verrà utilizzato il segnale di “dato pronto” dei moduli come sincronismo per il dato perché esso varia da modulo a modulo. Questa scelta, “necessaria” per il funzionamento di un sistema così complesso comporta un inconveniente : ci costringe a sapere in quale periodo di tempo il dato all’uscita di un qualsiasi modulo è stabile e quindi può essere utilizzato (ovvero in quale intervallo di tempo non si deve mandare il sincronismo al modulo successivo).

Dato che il tempo di ritardo del modulo più lento è di 60 ns, i segnali di sincronismo possono essere generati con un periodo minimo proprio di 60 ns, cioè quando ormai il dato è pronto all’ingresso di qualsiasi modulo. Però non bisogna dimenticare il ritardo dei cavi che potrebbe fare in modo che i dati in certi punti del filtro arrivino in anticipo o in ritardo con i sincronismi.

Infatti si consideri il caso di figura 4.3 in cui due moduli inviano i dati l’uno all’altro opportunamente sincronizzati da un generatore di sincronismi.

Per semplicitá si consideri il cavo dei dati di lunghezza trascurabile. Si vede che il ritardo tra i due sincronismi, generati ogni 60 ns, dipende dalla lunghezza dei cavi L_1 ed L_2 . Infatti se $L_1 > L_2$ si avrá che il sincronismo (2) non arriva dopo 60 ns ma in anticipo rispetto al sincronismo (1) di :

$$\frac{L_1 - L_2}{T_p} ns$$

dove T_p è il tempo di propagazione del segnale nel cavo ($\simeq 5$ ns/mt), ed L_1 , L_2 sono espresse in metri. Al contrario se $L_2 > L_1$ il sincronismo (2) arriva in ritardo di :

$$\frac{L_2 - L_1}{T_p} ns$$

Puó perciò accadere che questi tempi siano abbastanza grandi da fare in modo che il sincronismo capiti in momenti in cui il dato non è stabile pregiudicando il funzionamento dei filtro.

Una soluzione triviale del problema è quella di fare tutti i cavi di lunghezza uguale al cavo piú lungo. Ciò però comporta dei problemi pratici: si pensi ad un cavo di qualche metro per collegare 2 moduli a pochi centimetri di distanza! Mentre una soluzione più funzionale è la seguente : si cerca di “recuperare” i ritardi dei cavi mediante una intelligente collocazione fisica del modulo non solo rispetto a quelli da cui riceve e invia le informazioni ma anche rispetto al modulo generatore di sincronismi, e facendo passare i cavi lungo percorsi obbligati.

Per capire come ciò sia possibile si considerino i 20 *crate* utilizzati per contenere il filtro in energia che sono disposti in cinque “rack” contenenti

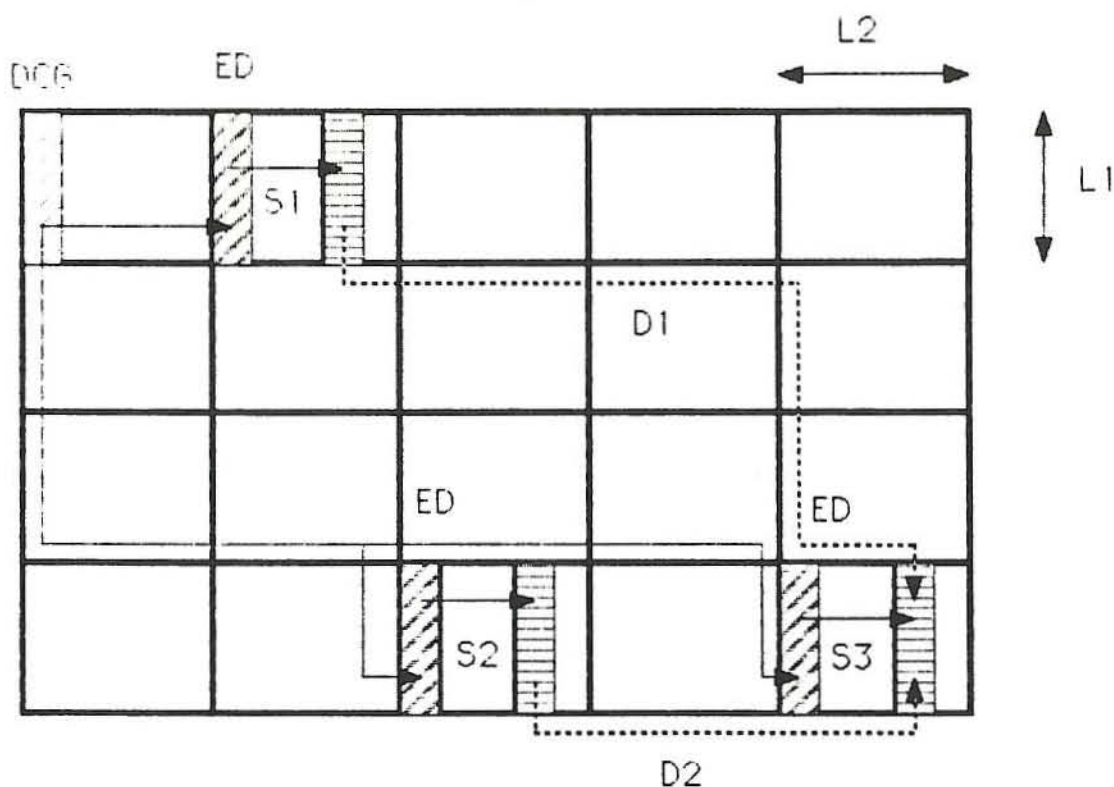


Fig. 4.4 Disposizione dei crates del filtro in energia

ognuno quattro *crates* come mostrato in figura 4.4. Si pensi ora di dover connettere 3 moduli abbastanza distanti. Ebbene il principio che verrà utilizzato per posizionare i moduli ed i cavi è dato dalle seguenti regole:

- i. Le informazioni ed i sincronismi viaggiano dall'alto verso il basso e da sinistra verso destra.
- ii. Il *Master clock* cioè il generatore principale delle temporizzazioni viene posto in alto a sinistra e i moduli generatori di sincronismi (Ecline Drivers) uno per *crate*.
- iii. I cavi passano esclusivamente lungo gli spazi tra *crate* e *crate*.

Sfruttando queste regole si sono effettuati i collegamenti come mostrato in figura 4.4 dove le linee piene sono i cavi dei sincronismi e quelle tratteggiate

individuano i cavi dei dati.

Si può calcolare il momento in cui i dati D1 e D2 e il sincronismo S3 si presentano nel modulo ricevente, considerando per semplicità nulli i ritardi dei moduli. Le due lunghezze L_1 ed L_2 possono essere considerate uguali perché nella configurazione finale tra due *crates* contigui nello stesso "rack" verranno interposti dei gruppi di ventilazione e di refrigerazione. Se a $t = 0$ è generato un impulso dal modulo DCG di figura 4.4, il dato D1 si presenterà sul modulo 3 in basso a destra dopo $8L$ (se si definisce L il tempo di ritardo di un cavo lungo L_1) in quanto viene generato al tempo $t = 2L$ e il cavo è lungo $6L_1$. Anche D2 ed S3 arriveranno al tempo $t = 8L$: infatti sono generati rispettivamente agli istanti $t = 6L$ e $t = 7L$ e si propagano su dei cavi lunghi $2L_1$ e L_1 .

Quindi D1, D2 e S3 giungono al modulo destinazione esattamente in fase pur utilizzando dei cavi di lunghezza diversa. Comunque in alcuni casi non si potranno rispettare le suddette regole: infatti per utilizzare al massimo le stazioni disponibili nei *crate* si potranno avere dei piccoli ritardi tra dati e sincronismi. Per sapere se questi ritardi sono accettabili o meno bisogna conoscere esattamente, per ogni modulo, l'intervallo di tempo in cui il dato è stabile in uscita.

Per caratterizzare in maniera precisa il comportamento temporale di un modulo sono stati definiti 2 tempi τ_1 e τ_2 .

Il tempo τ_1 è definito come il minimo tempo tra il fronte di salita del

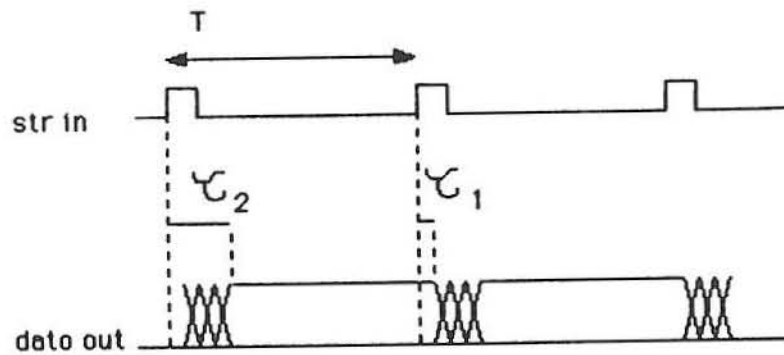


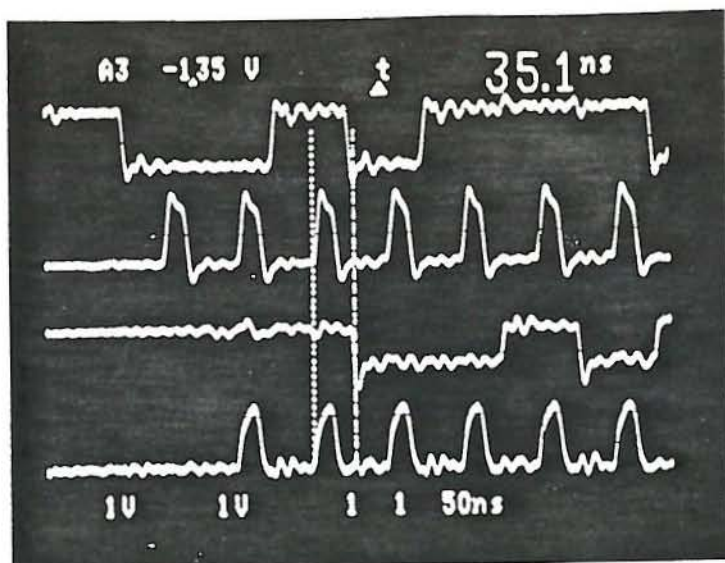
Fig. 4.5 definizione di τ_1 e τ_2

sincronismo che abilita la memorizzazione del dato in ingresso e il momento in cui comincia a cambiare il dato all'uscita. Invece τ_2 é il tempo minimo tra il fronte di salita del sincronismo e il momento in cui il dato é ormai stabile (fig. 4.5). Quindi se il periodo dei sincronismi in ingresso é T , l'intervallo di tempo in cui il dato é stabile (t_b) é :

$$\tau_2 < t_b < \tau_1 + T$$

Perció la conoscenza di τ_1 e τ_2 permette anche di stabilire la massima differenza di lunghezza dei cavi dei sincronismi e dei dati nei diversi percorsi.

Eeguire la misura di questi tempi non é semplice. Il primo approccio (sbagliato) potrebbe essere quello di mandare il sincronismo e un dato all'ingresso del modulo e misurare con l'oscilloscopio il tempo tra fronte di salita del sincronismo e cambiamento del dato. Ma in questa maniera (vedi fig. 4.6) si può non vedere il tempo di instabilità del dato, poiché questo può



DATO INGRESSO
STROBE INGRESSO
DATO USCITA
STROBE USCITA

Fig. 4.6 Foto all'oscilloscopio del cambiamento di un dato della ALU

esserci o meno a seconda della configurazione dei bit all'ingresso (Par. 4.2).

Per poter effettuare una misura significativa si dovrebbe effettuare la misura dei tempi all'oscilloscopio per tutti i cambiamenti tra le configurazioni possibili dei 16 bit di ingresso.

4.4 Misura di τ_1 e τ_2

Un approccio più realistico che fornisce un metodo efficiente di misura è il seguente: prendiamo, oltre al modulo di cui dobbiamo misurare τ_1 e τ_2 , due moduli che servono da sorgente e destinazione per i dati (ad. es. 2 Data stacks) ed un generatore di sincronismi, collegati al modulo in esame come mostrato in figura 4.7.

Il modulo "sorgente" deve essere tale da poter essere caricato, via cal-

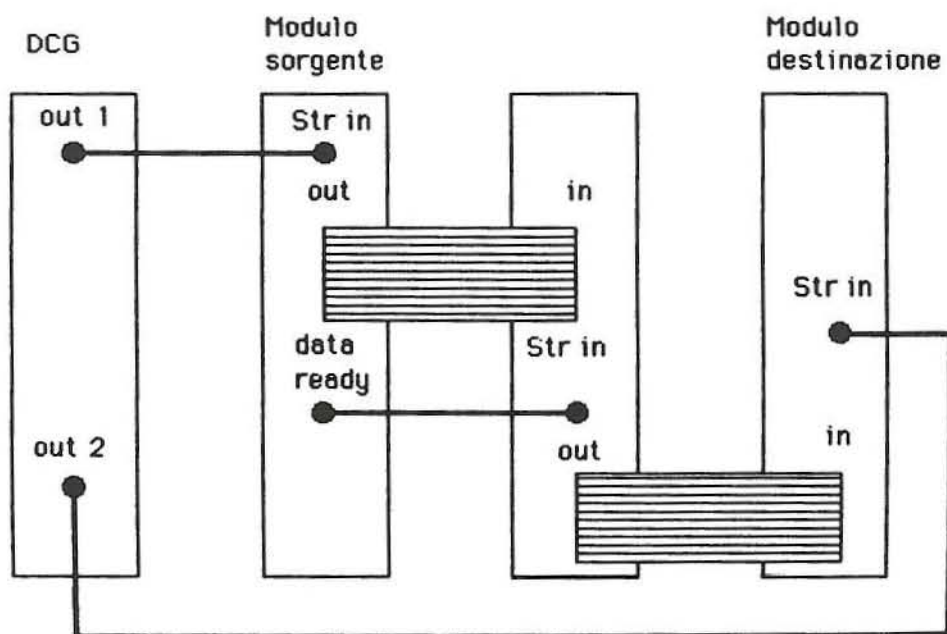


Fig. 4.7 connessioni per la misura di τ_1 e τ_2

colatore con sequenze abbastanza lunghe di parole binarie che poi vengono inviate ad alta velocità al modulo in prova. La "destinazione" al contrario deve poter ricevere sequenze di parole ed essere letta poi via calcolatore.

L'idea che si sfrutta in questo caso è quella di spostare nel tempo il sincronismo del dato in uscita del modulo di cui si deve misurare τ_1, τ_2 mediante il modulo DCG (Delay and Clock generator) e controllare, mediante

calcolatore, quando si ottengono risultati corretti od errati nel modulo di destinazione.

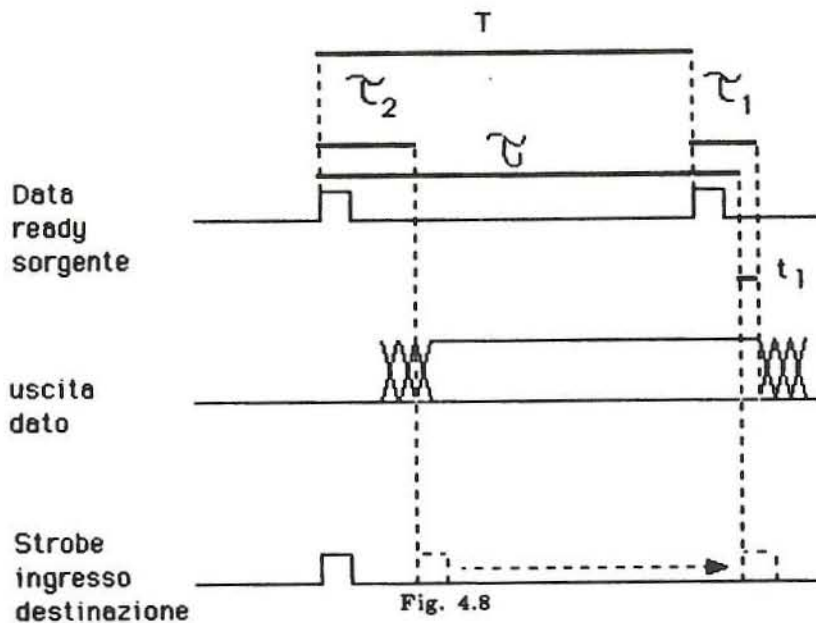
Utilizzando tali moduli si effettua la misura in questa maniera :

- Si carica la "sorgente" con delle sequenze particolari di parole che verranno descritte diffusamente nel capitolo 5.
- Si fa in modo che il sincronismo del modulo destinazione arrivi troppo presto rispetto ai dati.
- Si effettua il trasferimento dati.
- Si controlla leggendo la destinazione via calcolatore se ci sono stati errori nel trasferimento, in caso affermativo si ritarda il sincronismo (in passi di 5 ns col DCG). Si prosegue finché non avviene nessun errore.

A questo punto si effettua la misura con l'oscilloscopio del tempo tra il sincronismo d'ingresso del modulo e il sincronismo d'ingresso della "destinazione" ricordandoci di sottrarre il tempo di propagazione del dato lungo il cavo (in realtà tale tempo é stato trascurato essendo tale cavo di pochi cm.): questo tempo é τ_2 .

Per poter incrementare il ritardo in passi di un nanosecondo ed ottenere misure piú precise si sono utilizzati dei cavi di lunghezze diverse per connettere l'ingresso di sincronismo del modulo "destinazione".

Nonostante gli oscilloscopi a disposizione permettessero misure molto piú precise, é stata sufficiente una precisione di ± 1 ns in quanto sperimentalmente



si é notato che τ_2 peggiora o migliora a seconda del modulo utilizzato come “destinazione” e dipende dal tipo di terminazione usata.

Fatta la misura, si porta il sincronismo d’ingresso verso la fine del dato, non appena si verificano errori si torna al passo precedente (l’ultimo in cui tutto funzionava perfettamente) e si misura il tempo (τ) tra i due segnali di sincronismo (fig. 4.8); tale tempo risulta pari a

$$\tau = T + \tau_1 - t_l$$

dove T é il periodo dei sincronismi e t_l é il tempo di *latch* del modulo destinazione cioè il tempo in cui il dato deve essere stabile in ingresso per poter essere memorizzato. Quindi si ha :

$$\tau_1 = \tau - T + t_l$$

t_l é noto per i moduli utilizzati ed é pari a 10 ns. Con questa procedura sono state effettuate le misure di τ_1 e di τ_2 sui moduli utilizzati nel filtro di

energia. I risultati sono riassunti nella tabella seguente.

modulo	τ_1 (ns)	τ_2 (ns)
ALU	36	51
Bus SWitch	11	30
Data Stack	20	35
Ecline Driver	24	37
MLU	30	60

Purtroppo avendo a disposizione solo poche unità (a volte una) di alcuni moduli i dati sono solo preliminari. Quando verranno prodotti più moduli si potranno misurare definitivamente i valori minimi e massimi per τ_1 e τ_2 .

5 Le prove dei moduli

5.1 Introduzione

Per assicurare un perfetto funzionamento del filtro é necessario controllare la funzionalità dei moduli e misurare le loro caratteristiche. Così é nata l'esigenza di scrivere un'insieme di programmi che permettessero di provare in maniera automatica e veloce ogni singolo modulo. Ciò é stato reso possibile dal fatto che é stato utilizzato un calcolatore Digital VAX 750 interfacciato allo standard (fig. 5.1) CAMAC tramite un *crate* di sistema. Tramite questo sistema si possono far eseguire ai moduli tutte le funzioni CAMAC per cui sono stati progettati.

Come si é visto nel capitolo 4 (figura 4.5), per poter controllare il funzionamento via pannello frontale dei moduli sono necessari dei moduli addizionali che permettono di trasferire ad alta velocità sequenze di parole binarie utilizzando il protocollo ECL differenziale. Il programma, per effettuare una prova, dovrà caricare tali "sorgenti" (v. fig. 4.5) con delle parole binarie particolari, far eseguire al modulo le operazioni via pannello frontale abilitando un generatore di sincronismi ed infine leggere i risultati nel modulo "destinazione" per poter controllare il corretto funzionamento del modulo stesso.

Per fare ciò é stato sviluppato un pacchetto di programmi che permette, utilizzando opportune configurazioni di "hardware", di provare i moduli e

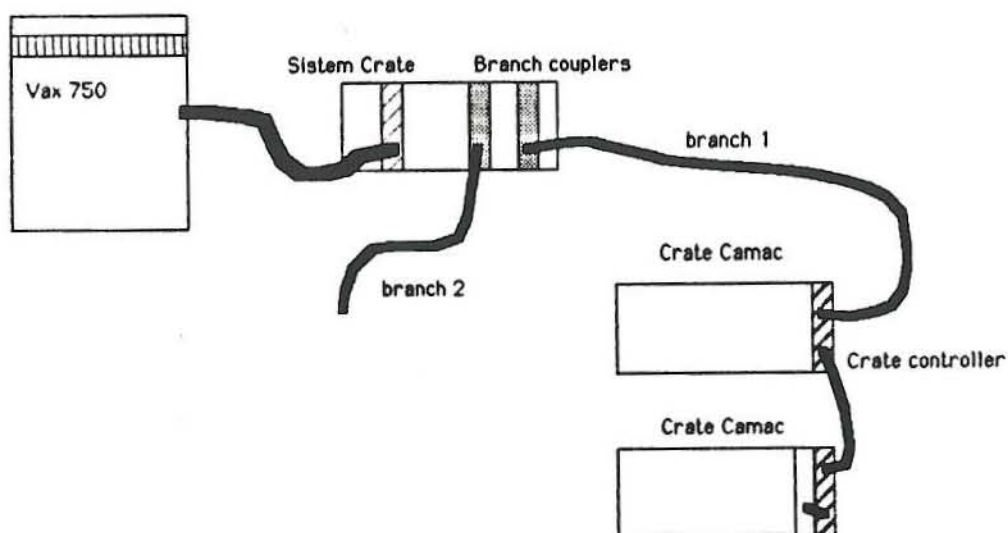


Fig. 5.1 Schema dell'interfaccia Camac - Vax

di misurare τ_1 e τ_2 . In appendice B sono riportati i listati dei programmi nonché la relativa documentazione per poterli utilizzare.

5.2 I programmi

L'insieme di questi programmi è stato sviluppato utilizzando la filosofia dei menù per renderne più facile l'uso ad utenti poco esperti; l'utilizzo della libreria MNPACK fornita dal CERN ha reso molto facile la gestione di questi menù. Tramite varie scelte guidate dai menù l'utente può decidere in quale stazione del *crate* collocare i moduli, quale tipo di prova eseguire, modificare vari parametri etc. Il diagramma di flusso di un algoritmo che esegue la prova di un modulo è schematizzato in figura 5.2.

Il diagramma vale, a parte lievi modifiche, per tutti i moduli che eseguo-

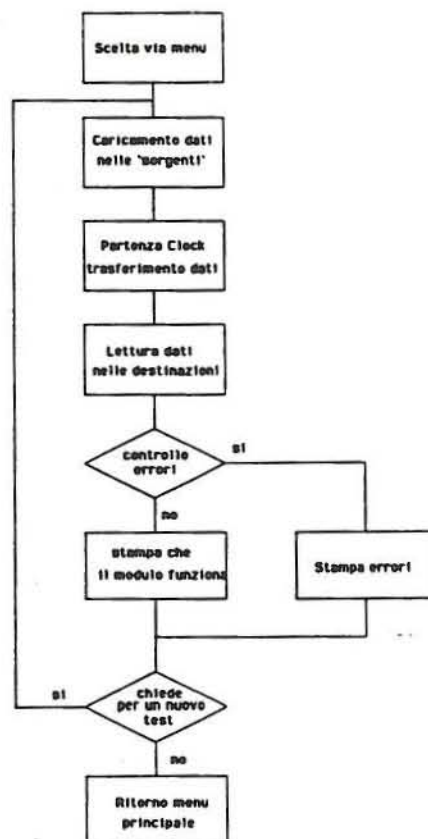


Fig. 5.2 Diagramma di flusso delle prove

no operazioni logiche (ALU, BSW, MLU, DS, ED) ma non per i FERA che essendo dei convertitori analogico-digitali richiedono controlli sulla linearità e sui piedistalli e quindi dei controlli statistici a cui si dedica un paragrafo a parte.

Si possono eseguire, considerando lo schema di figura 4.1, le seguenti prove :

- Prova dei registri di ingresso :

Qualora esistano nel modulo delle funzioni CAMAC che permettano la lettura e scrittura dei registri di ingresso é possibile controllare il funzionamento dei soli registri in ingresso.

- Prova delle connessioni frontali :

Scrivendo da pannello frontale e leggendo via Camac si possono controllare le connessioni in ingresso. Un analogo controllo può essere eseguito per le connessioni di uscita.

- Controllo dei risultati :

Il programma simula l'operazione che deve effettuare il modulo e ne confronta i risultati con quelli ottenuti.

- Prove temporali :

consentono di stabilire e caratterizzare il comportamento temporale del modulo.

Una fase molto importante nella definizione di questi programmi é stata la scelta delle configurazioni di bit da caricare nei moduli sorgente poiché, come si é già visto, la significatività delle prove (cfr. cap. 4) dipende da esse. Un ovvia scelta é quella di utilizzare tutte le configurazioni possibili. Ma le configurazioni possibili di 16 bit sono 65536 e i modi in cui possiamo far cambiare il contenuto di un registro sempre a 16 bit sono :

$$(65536 \times 65536) \sim 36 \cdot 10^{10}$$

Ora il Vax 750 per eseguire una funzione Camac impiega qualche millisecondo (circa 4 ms in media) quindi per il test completo di 1 solo registro occorrono:

$$36 \cdot 10^{10} \times 4 \cdot 10^{-3} \sim 12 \cdot 10^8 \text{sec.} \sim 12 \cdot 10^3 \text{giorni}$$

che risultano eccessivi per un solo controllo.

È stato quindi necessario utilizzare delle configurazioni in un certo senso “approssimate” che permettessero però di eseguire le prove in un tempo ragionevole e che tenessero conto delle condizioni critiche di funzionamento dei singoli moduli.

1. Prova del *Cross talk*.

Il *cross talk* avviene quando c'è un cifra binaria che, cambiando di livello, può perturbare lo stato logico di un'altra che non cambia il proprio stato. Per poter effettuare questa prova si utilizza la seguente sequenza di parole binarie :

```
1111 1111 1111 1111
0000 0000 0000 0001
1111 1111 1111 1111
0000 0000 0000 0010
```

·
·
·

```
1111 1111 1111 1111
1000 0000 0000 0000
```

e

```
0000 0000 0000 0000
1111 1111 1111 1110
0000 0000 0000 0000
1111 1111 1111 1101
```

·
·
·

```
0000 0000 0000 0000
0111 1111 1111 1111
```

cioè la filosofia è quella di far cambiare tutti i bit di una parola tranne uno in maniera che i circuiti logici siano sottoposti a dei cambiamenti bruschi di configurazione.

2. Prova a campionamento (*sampling test*).

È una approssimazione del caso ideale (prova di tutte le configurazioni), in cui si considera la parola a 16 bit come formata da 4 parole indipendenti di 4 bit ciascuna; quindi per coprire tutte le configurazioni bastano solo 16 parole infatti in esadecimale :

0000	→	0000	0000	0000	0000
1111	→	0001	0001	0001	0001
2222	→	0010	0010	0010	0010
		.			
		.			
		.			
FFFF	→	1111	1111	1111	1111

3. Prova dei bit pari e dispari.

Consiste nello scrivere sequenze alternate in cui sono ad uno i bit pari o i bit dispari mentre gli altri sono a zero; è una prova molto rapida che di solito è utilizzata per controllare il funzionamento delle memorie poste all'interno dei moduli.

4. Prova con numeri casuali.

Si generano sequenze casuali di 16 bit utilizzando il generatore di numeri casuali del Fortran. Più sequenze vengono generate più la prova è significativa: è stato usato questo procedimento per le prove di lunga durata (alcuni giorni di funzionamento) dei moduli.

Una volta caricati i moduli sorgente (Data stack A, B di fig. 5.3) si abilita il generatore di impulsi (Clock Generator di fig. 5.3) tramite una funzione Camac da calcolatore. Gli impulsi generati, fungendo da sincronismi per la lettura dei moduli sorgente, permetteranno il trasferimento delle parole al modulo da provare il quale, dopo averle elaborate, le invierà insieme ad un

sincronismo al modulo "destinazione" (Data stack C di Fig. 5.3). A questo punto il programma leggerá i risultati, tramite Camac, nella "destinazione" e segnerà l'eventuale mancato funzionamento del modulo, mostrando la parola in ingresso, cosa si é letto in uscita ed il risultato che ci si aspettava. Ogni modulo ha il proprio programma di prova che richiede una particolare configurazione di "hardware" per poter essere eseguito; il tutto é stato poi raccolto in un unico programma (Bigtest) a cui é allegato un "file" di documentazione in cui é descritto dettagliatamente sia come collegare i vari moduli per la prova che gli algoritmi usati. Il listato del programma (Bigtest) é riportato in appendice B.

5.3 L'hardware delle prove

Per effettuare le prove é necessaria una particolare configurazione di moduli dipendente da quello in esame. Si é cercato di utilizzare il minor numero possibile di moduli utilizzando, ove possibile, i segnali di "dato pronto" come sincronismo dei dati.

Uno dei problemi é stato come memorizzare su calcolatore le connessioni tra i moduli necessarie per le singole prove; ciò é stato risolto utilizzando i caratteri speciali del codice ASCII e componendo cosí dei disegni come quello riportato in figura 5.3 che, molto semplicemente, mostrano le connessioni necessarie e sono facilmente visualizzabili su qualsiasi tipo di terminale o stampante.

Ovviamente tutti i moduli addizionali utilizzati devono essere perfetta-

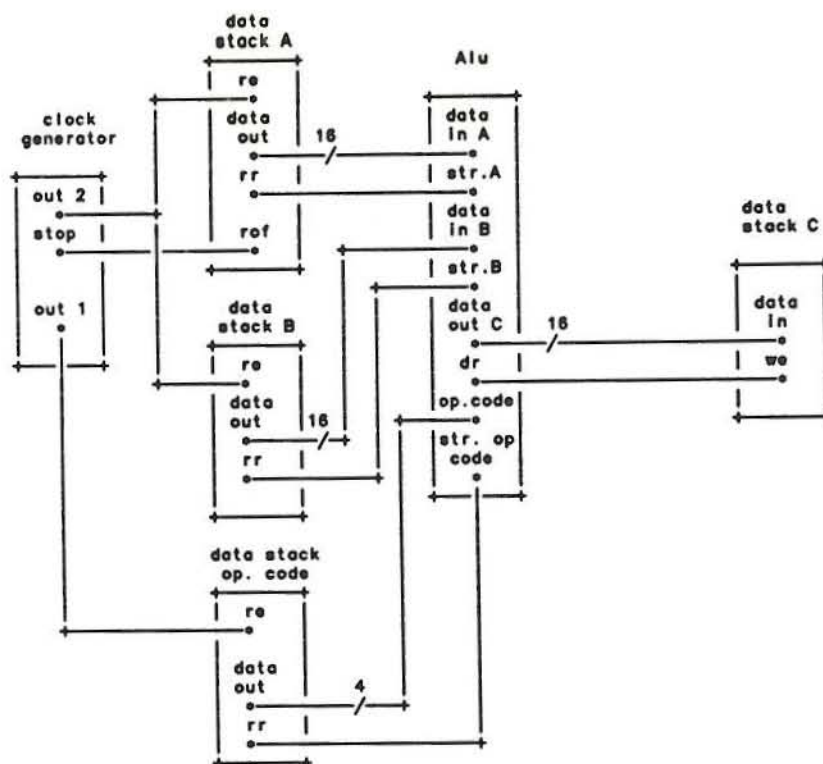


Fig. 5.3 Connessioni per il test dell'ALU

mente funzionanti. Nel caso di fig. 5.3, uno dei piú complessi che riguarda la prova di una ALU, sono necessarie 4 Data Stack e un Delay and Clock Generator. Il DCG serve ad abilitare la lettura via pannello frontale delle 3 data stack (2 per i dati di ingresso, una per il codice operativo dell'ALU) fornendo i relativi sincronismi di lettura a quest'ultime. La quarta DS serve per memorizzare i risultati che verranno poi letti via il "dataway" CAMAC.

5.4 Le prove sui Fera

Le prove eseguite sui Fera sono molto diverse dalle altre, poiché questi sono dei convertitori analogico digitali ed hanno bisogno di particolari controlli; tali prove sono particolarmente importanti dato che i Fera forniscono i dati per l'elaborazione del primo e secondo livello di filtro.

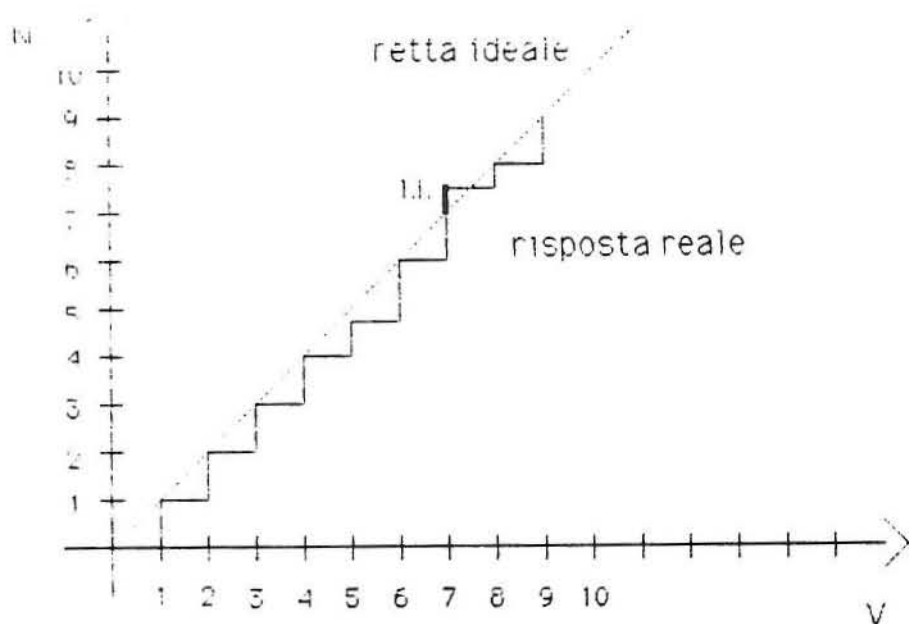


Fig. 5.4 Grafico della risposta di un ADC

Per un convertitore analogico digitale si può costruire il grafico [5] $N = f(V_{input})$ (in cui N è il valore numerico corrispondente al V_{input}) variando V_{input} con un generatore programmabile (DAC). Per un ADC ideale questo grafico (Fig. 5.4) è una retta. Le deviazioni da quest'ultima permettono di valutare la non linearità dell'ADC. Un'altra caratteristica fondamentale degli ADC è la linearità integrale, cioè lo scarto tra l'uscita digitale e l'uscita aspettata (i.l. in fig. 5.4). Infine è necessario controllare la stabilità del piedistallo (fuori zero), cioè verificare che sia sempre lo stesso nei limiti di precisione dell'ADC.

5.4.1 Prove sulla linearità.

Per eseguire le prove è stato utilizzato il DAC a 12 bit programmabile

interno al Fera Driver che permette di generare una tensione di prova per i Fera. Variando questa tensione da zero al fondo scala del Fera in passi determinati é stato eseguito un "fit" con la retta $y = a \cdot x + b$, in cui x é il valore binario caricato nel DAC del Fera Driver e y é il valore all'uscita della conversione del Fera meno il suo piedistallo. Si possono allora calcolare le 3 grandezze a , b e χ^2 ;

Per la pendenza (a) o guadagno dell'ADC il valore aspettato é dato dalle specifiche tecniche LeCroy ed é (v. app. A):

$$0.97 \leq a \leq 1.03$$

Quello di b sempre dalle specifiche, risulta essere:

$$-2 \leq b \leq 2$$

Invece per il χ^2 , che misura la bontá del "fit", non era fornito dal costruttore nessun valore di confronto. Quanto valesse il χ^2 di un Fera funzionante non era noto e quindi si é pensato di effettuare uno studio statistico per capire quale fosse il suo valore.

Per fare ciò nel novembre 1986 si sono provati tutti i Fera allora disponibili al CERN presso la collaborazione L3 (38 moduli $\rightarrow$ 608 canali di ADC) ed é stato fatto un istogramma dei valori χ^2/n , dove n é il numero di misure effettuate, dei singoli canali. In questa maniera sullo stesso istogramma si possono riportare i valori χ^2/n sia per i Fera a 10 bit che per quelli a 11 poiché il fondo scala dei primi é esattamente la metà di quello degli altri.

Dall'istogramma di fig. 5.5 si nota che la maggior parte dei Fera é funzionante e χ^2/n risulta intorno al valore 1, mentre il picco per valori di χ^2/n tra 4 e 5 denuncia l'esistenza di moduli malfunzionanti. Il risultato ottenuto non dipende dal DAC del Fera Driver poiché utilizzandone un altro si ha la stessa distribuzione come mostrato in figura 5.6.

Anche per quanto riguarda la distribuzione dei guadagni dei singoli canali é stata riscontrata l'indipendenza dal Fera Driver usato (Fig. 5.7 e 5.8). Sarebbe stato estremamente sgradevole se i guadagni dei Fera fossero stati dipendenti dal Fera Driver usato anche se perfettamente funzionante: infatti se ciò fosse stato vero non si sarebbe mai potuto misurare il vero guadagno dei canali, ed effettuarne poi l'equalizzazione. (V. cap. 3 par. 5 equalizzazione del guadagno).

5.4.2 Prove sulla linearità integrale.

La linearità integrale é definita nel nostro caso come :

$$l.i. = (ADC_{output} - pedestal) - DAC_{input} \cdot a$$

dove a é la pendenza. Dalle specifiche tecniche tale linearità deve risultare:

$$l.i. \leq 2 + 0.25\% \cdot ADC_{output}$$

É stata perciò misurata la linearità integrale di ciascun canale variando la tensione in ingresso al Fera e controllando che rientrasse nei limiti previsti.

5.4.3 Prove dei piedistalli.

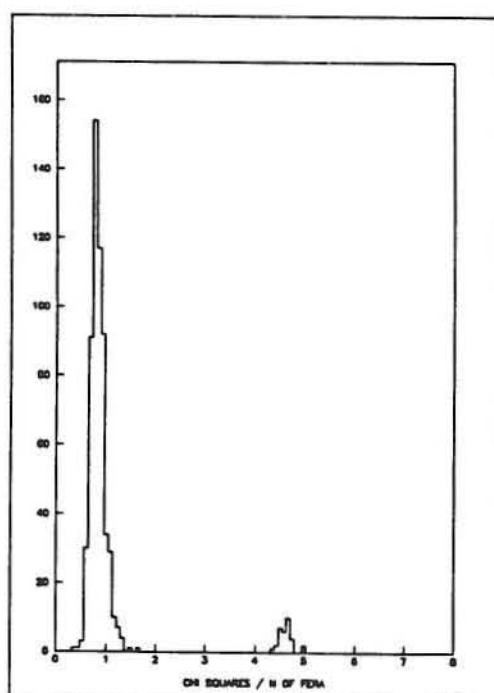


Fig. 5.5 Istogramma del χ^2 calcolato col FD 1

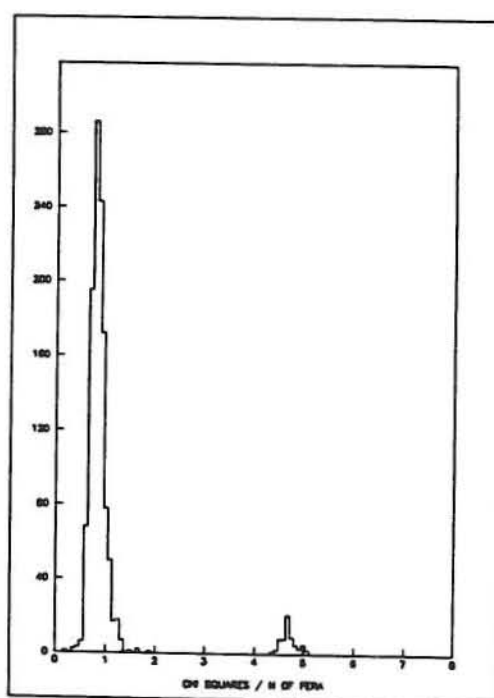


Fig. 5.6 Istogramma del χ^2 calcolato col FD 2

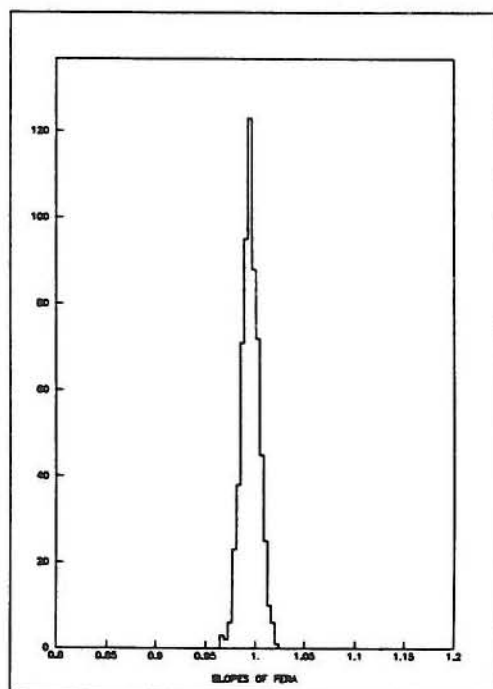


Fig. 5.7 Guadagni dei Fera calcolati col FD 1

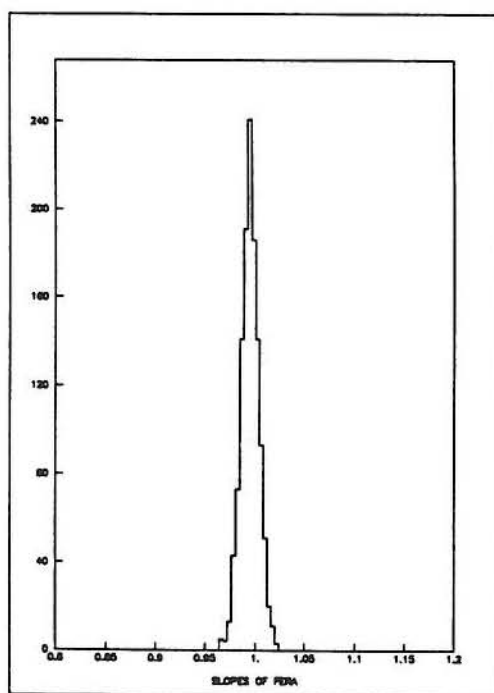


Fig. 5.8 Guadagni dei Fera calcolati col FD 2

Il piedistallo é quel valore che assume l'uscita di un ADC quando il suo ingresso analogico é a tensione nulla. É molto importante che esso sia stabile nel tempo nei limiti degli errori dell'ADC, poiché una sua instabilità causerebbe un'inevitabile variazione dell'informazione digitale in uscita da cui può venir sottratto automaticamente nei Fera stessi.

Per controllarne la stabilità, il piedistallo é stato misurato N volte ed é stata calcolata la media e la varianza dei risultati. Tale varianza é un buon parametro di controllo: infatti é possibile calcolare quale deve essere il suo valore per una grandezza analogica assolutamente stabile posta in ingresso all'ADC. Supponiamo che tale valore sia uguale a V : in uscita all'ADC, poiché la conversione ha una precisione di $\pm \frac{1}{2}$ bit, per n misure si potrà avere un risultato pari a x e per k il valore $x + 1$, dove il numero totale delle misure eseguite é $N = n + k$. La varianza é definita come:

$$\sigma^2 = \overline{x^2} - \bar{x}^2$$

ma nel nostro caso

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i = \frac{1}{n+k} \sum_{i=1}^{n+k} x_i = \frac{1}{n+k} (nx + k(x+1)) = x + \frac{k}{n+k}$$

elevando al quadrato si ottiene:

$$\bar{x}^2 = x^2 + \frac{2kx}{n+k} + \frac{k^2}{(n+k)^2}$$

Per $\overline{x^2}$ analogamente si ha:

$$\overline{x^2} = \frac{1}{n+k} (nx^2 + k(x+1)^2) = x^2 + \frac{2kx+k}{n+k}$$

Quindi la varianza e':

$$\sigma^2 = \overline{x^2} - \bar{x}^2 = \frac{nk}{(n+k)^2}$$

Ma essendo $N = n + k$ e quindi $k = N - n$ si ha:

$$\sigma^2 = \frac{nN - n^2}{N^2}$$

Il cui massimo, per $n = \frac{N}{2}$, vale 0.25 per cui:

$$\sigma \leq 0.5$$

Cosí nei nostri controlli é stato necessario verificare che per segnali nulli in ingresso, ogni canale avesse un valore di $\sigma \leq 0.5$ su tempi lunghi (circa 5 minuti: si deve tener conto che durante la futura presa dati i piedistalli dovrebbero essere controllati circa ogni minuto). Inoltre si deve considerare il fatto che il piedistallo varia al variare del segnale di "gate" dei Fera. Con queste misure quindi é stata controllata anche la stabilitá di tale segnale. Il listato del programma per eseguire tutte le prove é riportato in appendice B.

6 Le prove dei blocchi funzionali

6.1 I blocchi funzionali

In un apparato elettronico come il filtro di energia è molto importante prima del montaggio completo, che avverrà alla fine del 1987, aver effettuato delle prove di alcune sue parti per poterne verificare non solo il corretto funzionamento ma anche per comprendere quali problemi possono sorgere al momento dell'installazione completa. Sono state perciò isolate delle parti che effettuano una particolare funzione logica nel filtro e che vengono dette "blocchi funzionali". Ad esempio un blocco funzionale può essere costituito da un tipico gruppo di conversione, dall'insieme di moduli che costituiscono il filtro di fotone singolo (fig. 3.16) etc. L'estensione di un blocco funzionale può essere più o meno grande a seconda di quale funzione del filtro viene isolata. Nel nostro caso siamo stati costretti ad isolare blocchi relativamente piccoli a causa del ristretto numero di moduli a nostra disposizione (3 ALU, 1 BSW, 4 Data Stack, 4 Fera, 2 Fera Driver, 1 Ecline Driver, 4 MLU). Nonostante queste limitazioni le prove in laboratorio hanno dato dei risultati molto interessanti: si sono infatti incontrati dei problemi sia nella temporizzazione dei blocchi che nel funzionamento di qualche modulo. In particolare sono stati provati i seguenti blocchi funzionali :

1. Gruppo di conversione
2. Calcolo del vettore ELM $R\phi$
3. Filtro di *cluster* nel piano $R\phi$

Il primo è stato scelto perchè il 60% del numero totale dei moduli è utilizzato nei gruppi di conversione, gli altri due in quanto avrebbero potuto presentare problemi di temporizzazione: infatti non era ben chiaro dalle specifiche il modo di operare della DS per operazioni di scrittura e lettura ad alta velocità, ne se il modo proposto per sommare con le ALU le informazioni contigue potesse funzionare.

6.2 Le prove in laboratorio

6.2.1 Prova del Gruppo 1 di conversione

Il gruppo 1, mostrato in figura 6.1, è formato da 4 Fera, 2 Fera Driver, 1 Bus Switch, 1 Ecline driver ed 1 MLU. La funzione di questo gruppo è di convertire i dati del calorimetro elettromagnetico corrispondenti alle righe $\theta = 1$ e $\theta = 16$ (32×2 parole) della matrice ELM; i dati possono essere quelli provenienti dal calorimetro e convertiti dai Fera oppure quelli di “simulazione” generati dall’ Ecline driver (Par 3.4). Si è scelto di provare questo gruppo in quanto era il più grande che si poteva realizzare con i moduli a disposizione.

I due Fera Driver si occupano di distribuire i segnali di sincronismo ai 2 Fera a loro connessi in modo da leggerne i dati contenuti. Si è visto (Cap. 3)

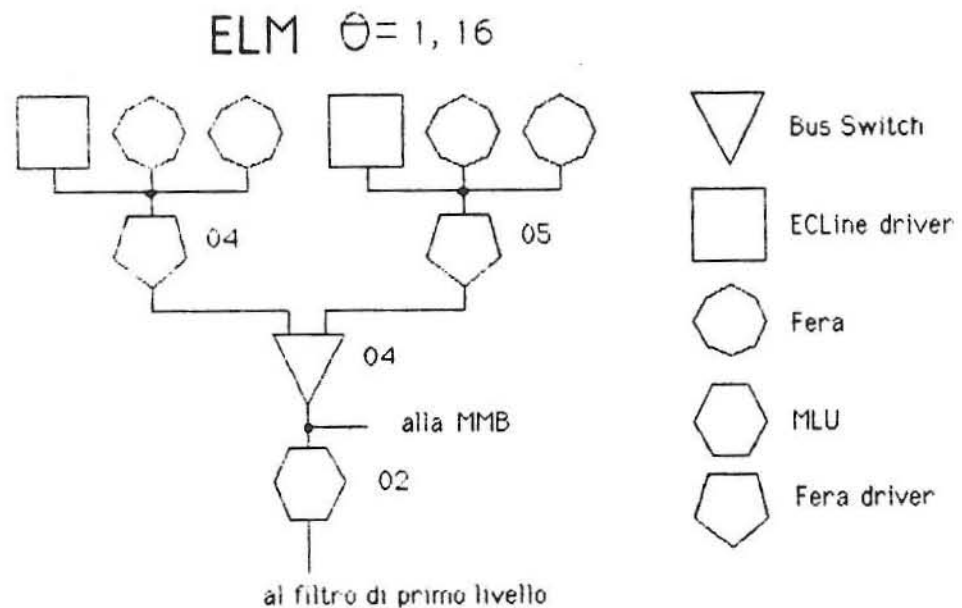


Fig. 6.1 Il gruppo 1 di lettura dei Fera

che con un BSW si ottiene la frequenza di lettura di un dato ogni 60 ns da 2 coppie di Fera in parallelo e con una MLU le operazioni previste sul dato tra cui la correzione del guadagno dei canali dei Fera.

Per temporizzare tutti i moduli del blocco occorrono i seguenti segnali :

- Fera e Fera Driver (da mandare solo al Fera driver).

- 1) CLI (CLear Input): azzeramento del contenuto dei registri dei Fera.
- 2) GAI (GAt e Input): "gate" (intervallo di campionamento del segnale) per i Fera ADC.
- 3) WAI (Write Acknowledge Input): segnale di "handshake" che informa il Fera che la parola in uscita sul Fera driver è stata memorizzata e si può passare alla successiva.

- Bus Switch.

- 1) STR A: sincronismo per i dati in ingresso alla porta A.
- 2) STR B: sincronismo per i dati in ingresso alla porta B.
- 3) A/B switch: segnale di comando che permette di commutare in uscita il contenuto del registro A ($A/B = 0$) o B ($A/B = 1$).

- MLU.

- 1) ENABLE: sincronismo dati in ingresso.
- 2) Bit 12 dei dati in ingresso: questo bit viene utilizzato per effettuare la correzione del guadagno dei Fera.

Per poter leggere i risultati del blocco si è utilizzata una Data Stack posta all'uscita della MLU. Non avendo un Ecline Driver al momento del test, si sono generati i segnali di tipo S, C e D con una Data Stack ed un Bus Splitter. Mediante la Data Stack (che è molto simile ad un Ecline driver ma molto più costosa) vengono generate le sequenze di segnali di tipo C e D, mentre quelli di tipo S si ottengono "formando" un'uscita del Bus splitter con un impulso ritardato del DCG.

A questo punto viene disegnato lo schema delle temporizzazioni (fig. 6.2) considerando i tempi caratteristici dei moduli τ_1 e τ_2 . Ai due Fera Driver vengono mandati il CLI (per inizializzare i Fera) e poi il GAI che fa iniziare la conversione, dopo $8.5 \mu s$ sono presenti alle porte di uscita dei Fera Driver i dati relativi ELM $\theta_1\phi_1$ ed ELM $\theta_1\phi_2$ che vengono quindi memorizzati nel Bus Switch con i segnali STR A e STR B ritardati tra loro di 60 ns e con

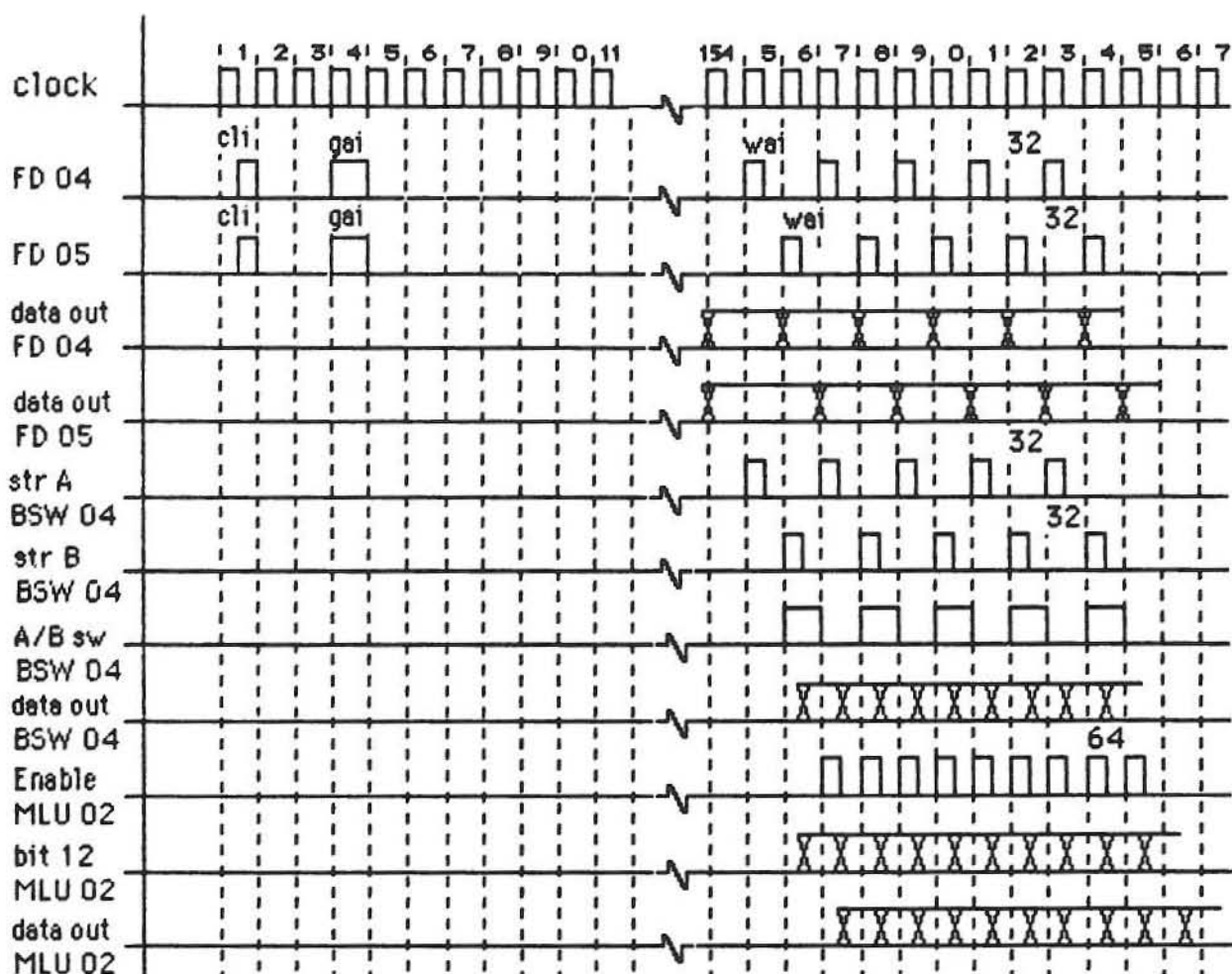


Fig. 6.2 Temporizzazioni del gruppo 1

periodo di 120 ns. In fase a STR A e B si mandano i WAI rispettivamente ai Fera Driver 04 e 05. Insieme alla STR B si pone A/B allo 0 logico, cosicchè dopo 30 ns all'uscita del Bus Switch si ha il dato del registro A ($ELM \theta_1 \phi_1$) e dopo altri 60 ns, ponendo A/B all'1 logico, quello contenuto in B. I dati, ora a 60 ns di periodo, vengono inviati alla MLU insieme al bit 12 (correzione del guadagno) ed il segnale di sincronismo ENABLE. La MLU effettua o meno

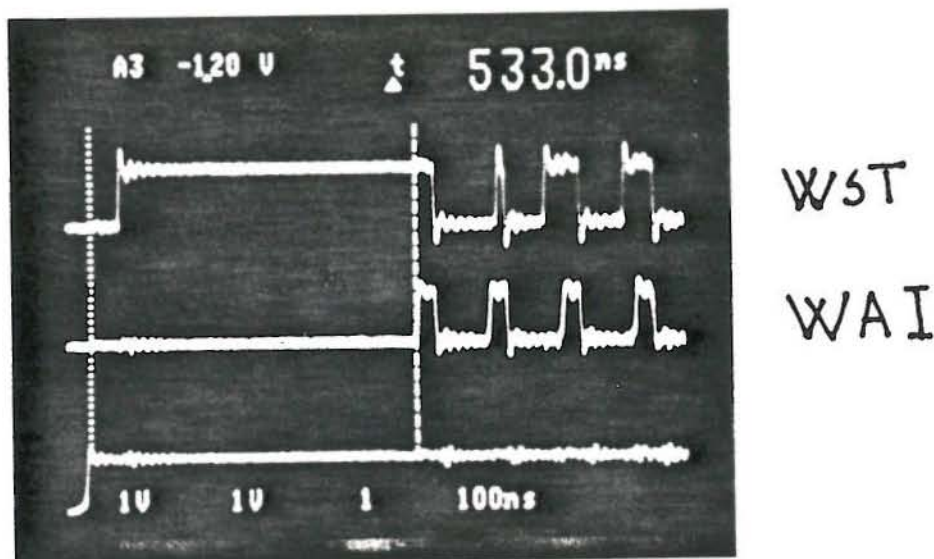


Fig. 6.4

per controllarne il funzionamento molto simile a quelli di prova dei moduli. In una prima fase si è letto come valore d'uscita il solo piedistallo dei Fera mentre in una seconda sono stati inviati dei livelli di tensione decrescente ad ogni singolo ingresso di conversione. Il controllo è stato fatto confrontando i dati contenuti nei registri interni degli ADC con quelli trasferiti via pannello frontale e memorizzati nella Data Stack.

Il blocco non funzionava nonostante lo schema di temporizzazione fosse esatto: in particolare il primo WAI mandato al Fera Driver non abilitava l'uscita del secondo dato, se tale segnale giungeva al Fera molto tempo dopo la fine della conversione (più di 300 ns). Il problema è mostrato nella foto di figura 6.4, dove si vede che mandando il WAI dopo 533 ns dalla fine

della conversione, viene generato in modo anomalo il secondo WST (Write STrobe: il sincronismo dei dati). Per risolvere il problema è stato analizzato lo schema circuitale del Fera ed è stato notato che la “logica” di lettura dei dati è abilitata dal WST che viene creato formando e ritardando il fronte di salita del WAI. Per questo motivo in caso di WST anomalo non possono uscire i dati. Poichè il ritardo tra WAI e WST viene generato tramite una rete RC, la durata del WST dipende dalla durata e dal periodo di ripetizione del treno di impulsi WAI. Fortunatamente il condensatore era variabile per cui, solamente modificandone il valore, si è riusciti ad ovviare al problema. Quindi è stato necessario un lavoro di regolazione in tutti i Fera, in quanto quella eseguita dalla LeCroy causa sempre questo problema.

6.2.2 Calcolo del vettore ELM $R\phi$

Lo schema proposto per questo calcolo è quello mostrato in figura 6.5. I dati provenienti dalle ALU in cascata ovvero le prime 32 semisomme in $\theta \left(\sum_{i=1}^8 ELM\theta_i\phi_j (j = 1, \dots, 32) \right)$ passano attraverso la ALU 20 inalterati (codice operativo uscita $C = A$) e vengono memorizzati nella Data Stack 05. All'arrivo delle seconde 32 semisomme in θ , cioè $\sum_{i=9}^{16} ELM\theta_i\phi_j$, si leggono dalla Data Stack 05 le prime semisomme e si addizionano (codice operativo $C = A + B$) alle seconde nell'ALU 20, calcolando così il vettore ELM $R\phi$ le cui componenti vengono infine memorizzate nella Data stack 04.

Per la prova del blocco sono stati simulati i dati provenienti dalle ALU utilizzando una Data stack. I segnali di sincronismo necessari sono:

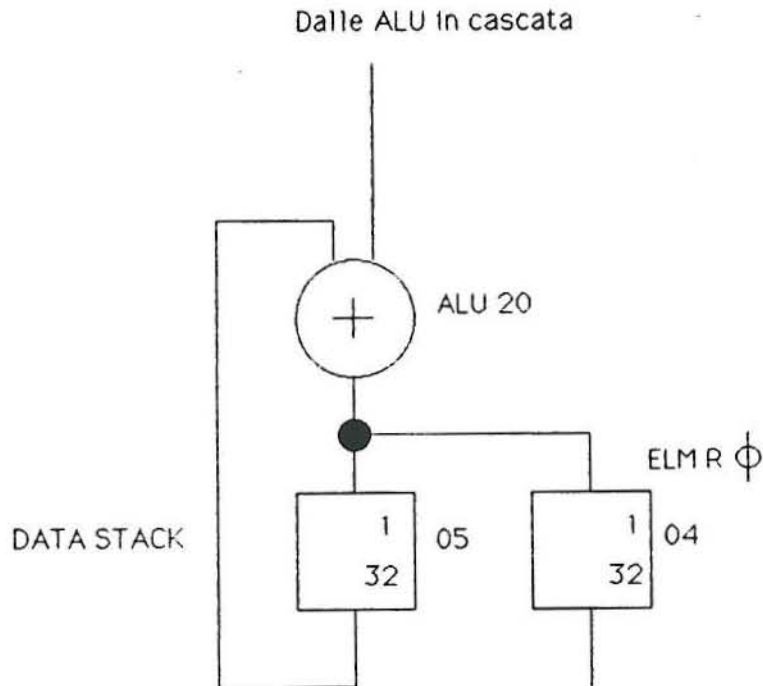


Fig. 6.5 Schema per il calcolo $ELM R\phi$

- DATA STACK 50 (simulazione dati):
 - 1) RE (Read Enable: sincronismo di lettura)
- ALU 20:
 - 1) STR A (Sincronismo dati in ingresso porte A)
 - 2) STR B (Sincronismo dati in ingresso porte B)
 - 3) 2 bit (1 e 4) per il codice operativo
- DATA STACK 05:
 - 1) RE (Read Enable: sincronismo di lettura)
 - 2) WE (Write Enable: sincronismo di scrittura)
 - 3) MRST (Master ReSeT: reset dei puntatori di lettura e scrittura)
- DATA STACK 04:

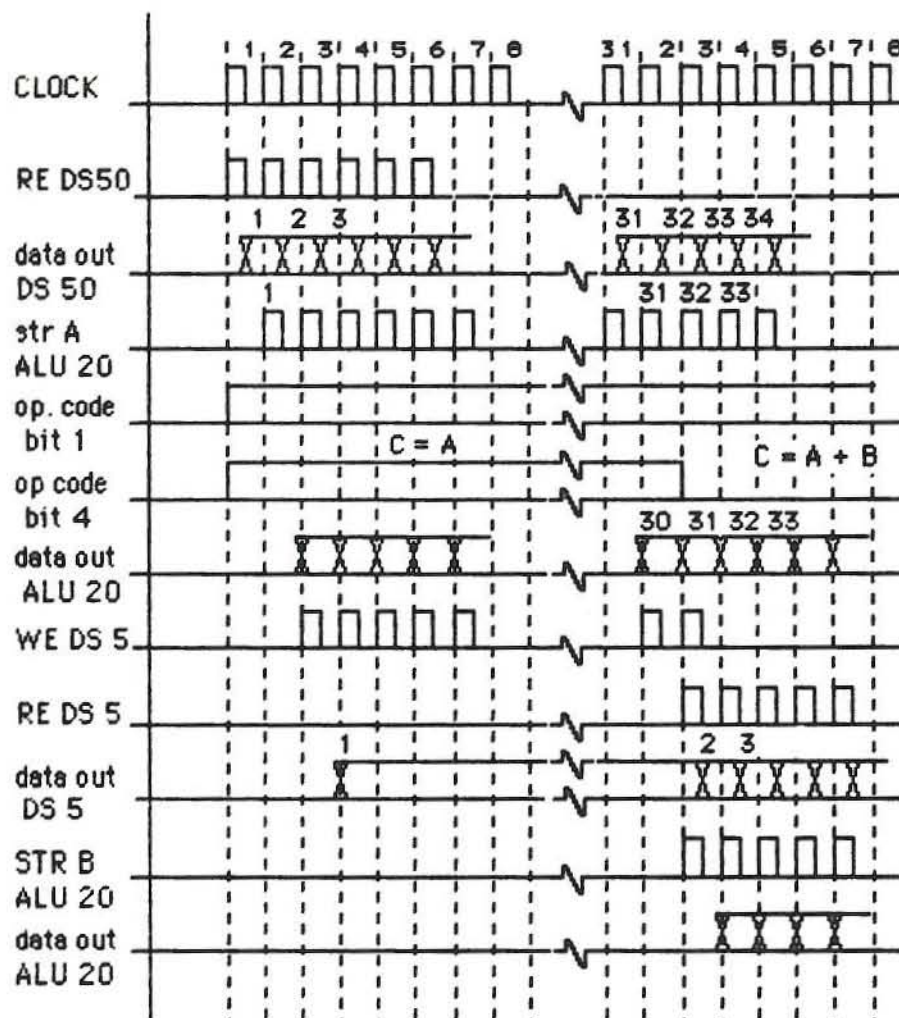


Fig. 6.6 Temporizzazioni per il calcolo di ELM $R\phi$

- 1) WE (Write Enable: sincronismo di scrittura)
- 2) MRST (Master ReSeT: reset dei puntatori di lettura e scrittura)

Lo schema delle temporizzazioni ottenuto è mostrato in figura 6.6 e si può vedere che il blocco non può funzionare in quanto per poter sommare in tempo le due prime semisomme di ciascun gruppo si è costretti a mandare nello stesso momento alla Data Stack 05 un sincronismo di lettura (RE) per

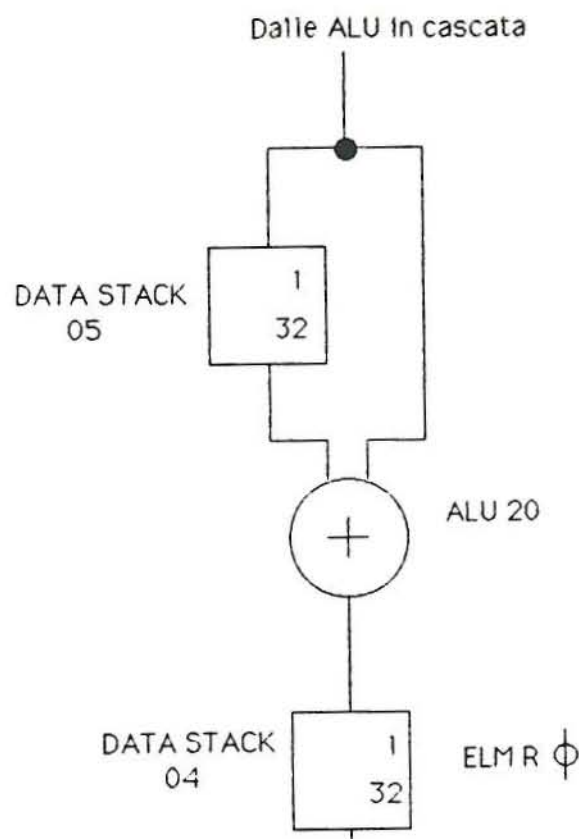


Fig. 6.7 nuovo schema per il calcolo di $ELM R\phi$

leggere la semisomma $\sum_{i=1}^8 ELM\theta_i\phi_1$ ed uno di scrittura (WE) per scrivere $\sum_{i=1}^8 ELM\theta_i\phi_{32}$, DATA STACK 05.

Perciò è stato riprogettato il blocco così come viene mostrato in figura 6.7. Ora i dati vengono direttamente memorizzati nella DATA STACK 05 senza passare inutilmente per l'ALU. In questa maniera non si deve neanche cambiare il codice operativo dell'ALU risparmiando due segnali di comando e, utilizzando il codice operativo $C = A + B$ che permette di sincronizzare ambedue i registri di ingresso con la sola STR B nella ALU 20, si può risparmiare un'ulteriore segnale. È stato ottenuto così il nuovo schema delle temporizzazioni di figura 6.8 e si può vedere che in questo caso non c'è sovrapposizione tra WE ed RE della Data stack 05.

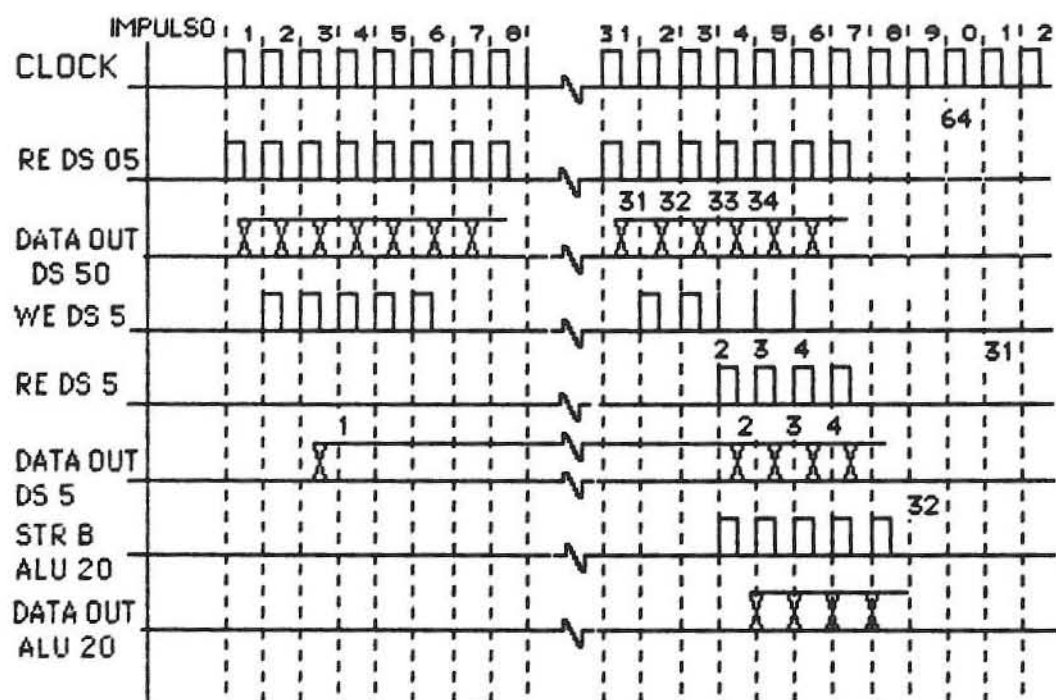


Fig. 6.8 Nuove temporizzazioni per il calcolo di ELM $R\phi$

Procedendo in analogo modo a quanto fatto per le prove del gruppo di conversione, dallo schema è stata ricavata l'informazione (fig. 6.9) da memorizzare nell' Ecline Driver, finalmente a disposizione, per la generazione dei sincronismi. Tramite un programma Fortran (App. B) è stato controllato, con successo, che le operazioni effettuate dal blocco fossero corrette.

La modifica apportata allo schema generale è stata accettata dalla col-

A	B	C	D	E	Impulso no.	t	
1	0	0	0	0	1	0	
1	1	0	0	0	2	60	
1	1	0	0	0	3	120	Sequenza di bit per
1	1	0	0	0	4	180	generare :
1	1	0	0	0	5	240	
1	1	0	0	0	6	300	A: RE DS 50
1	1	0	0	0	7	360	B: WE DS 05
1	1	0	0	0	8	420	C: RE DS 05
1	1	0	0	0	9	480	D: STR B ALU 20
1	1	0	0	0	10	540	E: WE DS 04
1	1	0	0	0	11	600	
1	1	0	0	0	12	660	
1	1	0	0	0	13	720	
1	1	0	0	0	14	780	
1	1	0	0	0	15	840	
1	1	0	0	0	16	900	
1	1	0	0	0	17	960	
1	1	0	0	0	18	1020	
1	1	0	0	0	19	1080	
1	1	0	0	0	20	1140	
1	1	0	0	0	21	1200	
1	1	0	0	0	22	1260	
1	1	0	0	0	23	1320	
1	1	0	0	0	24	1380	
1	1	0	0	0	25	1440	
1	1	0	0	0	26	1500	
1	1	0	0	0	27	1560	
1	1	0	0	0	28	1620	
1	1	0	0	0	29	1680	
1	1	0	0	0	30	1740	
1	1	0	0	0	31	1800	
1	1	0	0	0	32	1860	
1	1	0	0	0	33	1920	
1	0	1	1	0	34	1980	
1	0	1	1	0	35	2040	
1	0	1	1	0	36	2100	
1	0	1	1	0	37	2160	
1	0	1	1	0	38	2220	
1	0	1	1	0	39	2280	
1	0	1	1	0	40	2340	
1	0	1	1	0	41	2400	
1	0	1	1	0	42	2460	
1	0	1	1	0	43	2520	
1	0	1	1	0	44	2580	
1	0	1	1	0	45	2640	
1	0	1	1	0	46	2700	
1	0	1	1	0	47	2760	
1	0	1	1	0	48	2820	
1	0	1	1	0	49	2880	
1	0	1	1	0	50	2940	
1	0	1	1	0	51	3000	
1	0	1	1	0	52	3060	
1	0	1	1	0	53	3120	
1	0	1	1	0	54	3180	
1	0	1	1	0	55	3240	
1	0	1	1	0	56	3300	
1	0	1	1	0	57	3360	
1	0	1	1	0	58	3420	
1	0	1	1	0	59	3480	
1	0	1	1	0	60	3540	
1	0	1	1	0	61	3600	
1	0	1	1	0	62	3660	
1	0	1	1	0	63	3720	
1	0	1	1	0	64	3780	
0	0	0	1	1	65	3840	
0	0	0	1	1	66	3900	
A	B	C	D	E			

Fig. 6.9 tabella di temporizzazione per il calcolo di $ELM R\phi$

laborazione L3.

6.2.4. Filtro di *cluster* nel piano $R\phi$

Compito di questo blocco è controllare se ci sono dei *cluster* di particelle nei due calorimetri ed in particolare si deve controllare se la distribuzione in energia rivelata nei calorimetri è consistente con quella di un fascio di particelle provenienti dal vertice di interazione. Con la segmentazione in $R\phi$

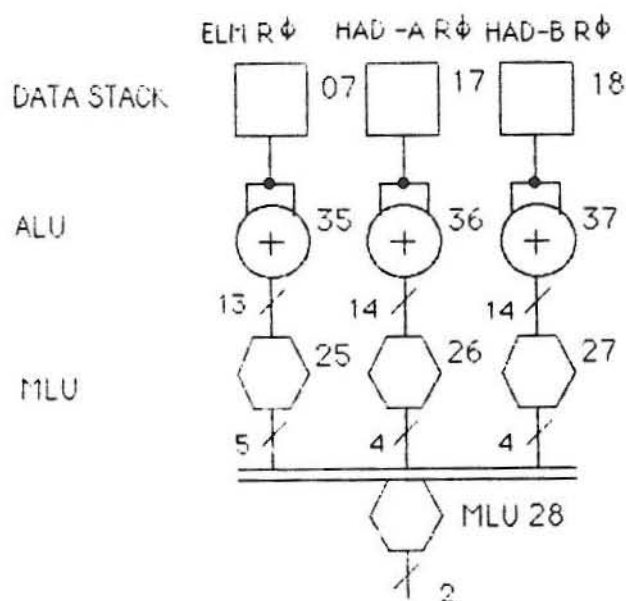


Fig. 6.10 Filtro di cluster

riportata al capitolo 3 (fig. 3.1) bisogna sommare e confrontare i segmenti nella seguente maniera:

ELM		HAD A-B
$\phi_2 + \phi_3$	} →	$\phi_1 + \phi_2$
$\phi_3 + \phi_4$		
$\phi_3 + \phi_4$	} →	$\phi_2 + \phi_3$
$\phi_4 + \phi_5$		
.		.
.		.
.		.
$\phi_{32} + \phi_1$	} →	$\phi_{16} + \phi_1$
$\phi_1 + \phi_2$		

Se l'energia rilasciata nei segmenti adiacenti dei due calorimetri è superiore ad un certo valore, tramite un'informazione di due bit si segnala il *cluster* ed il suo tipo.

Lo schema del filtro di *cluster* è mostrato in figura 6.10. Nelle tre Data stack sono memorizzate rispettivamente $32 + 16 + 16$ parole che rappresentano la distribuzione energetica nel piano $R\phi$ rispettivamente nel calorimetro elettromagnetico e nelle due parti del calorimetro adronico. Per eseguire la somma dei segmenti adiacenti vengono usate tre ALU i cui due ingressi sono connessi in comune. Con una opportuna sequenza di sincronismi (STR A, STR B ALU 35-36-37 di figura 6.11) si memorizzano i dati provenienti dalla Data stack alternativamente nel registro di ingresso A (primo, terzo, etc.) e nel registro B (secondo, quarto, etc.). In questa maniera si ottengono ad ogni sincronismo (tranne che per il primo) le somme delle componenti $R\phi_1 + R\phi_2$, $R\phi_2 + R\phi_3$, $R\phi_3 + R\phi_4$, etc. Poichè bisogna confrontare insieme 2 somme del calorimetro elettromagnetico con una di quello adronico, affinchè i dati arrivino sincronizzati alla MLU 28 la frequenza dei sincronismi delle ALU 36 - 37 deve essere la metà dell'ALU 35. Dato che la segmentazione fatta per il calorimetro elettromagnetico costringe ad iniziare a sommare dalla seconda componente del vettore ELM $R\phi$, è necessario leggere il primo dato nella Data stack 7 senza poi memorizzarlo nei registri dell'ALU. Poichè sequenzialmente i tre risultati delle somme debbono essere inviati ad un'unica MLU (MLU 28), è necessario approssimare, con tre MLU 25, 26 e 27 le tre parole di tredici cifre binarie in due parole da quattro ed una da cinque, in quanto il massimo numero di bit in ingresso ad una MLU di cui si utilizzano due bit in uscita è quindici. Si preferisce approssimare le parole, invece di non

tener conto delle cifre meno significative, per diminuire l'errore commesso nell'operazione.

Per temporizzare tutti i moduli del blocco servono i seguenti segnali:

- DATA STACK 7 - 17 - 18

1) RE

- ALU 35 - 36 - 37

1) STR A

2) STR B

- MLU 25 - 26 - 27 - 28

1) Enable.

Come al solito per registrare i risultati è stata utilizzata una Data Stack e dallo schema delle temporizzazioni (fig. 6.11) è stata ricavata la tabella (fig 6.12) da memorizzare nell'Ecline Driver utilizzato per la generazione dei sincronismi. Tenendo conto di quanto detto nel capitolo 4 riguardo alla dislocazione dei moduli, non sono stati trovati particolari problemi nelle temporizzazioni.

Prima dell'inizio delle prove le MLU dovevano essere programmate, cioè dovevano essere riempite le loro memorie con le tabelle che rappresentavano le operazioni da svolgere.

Per effettuare l'approssimazione binaria da 13 a 5 o 4 bit, la tabella di valori memorizzata nella MLU 25 è stata calcolata dividendo il numero per 2^{13-n} , dove n è il numero di bit significativi dopo l'approssimazione, e

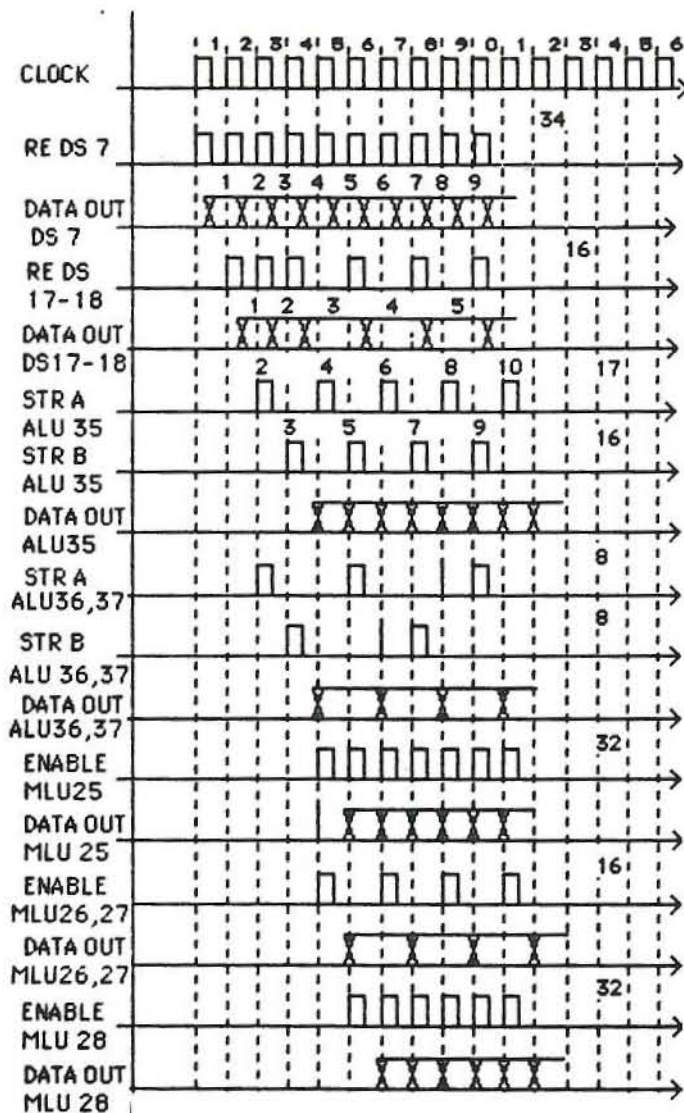


Fig. 6.11 Temporizzazioni per il filtro di cluster

prenderne poi l'intero più vicino. Questo in Fortran è espresso dalle seguenti

istruzioni:

N = 4 ! numero di bit significativi

DO ADDRESS = 0, 8095

MLU_OUTPUT = JNINT(ADDRESS / 2**(13-N)) ! calcola il valore da mettere nella locazione ADDRESS

CALL LOAD_MLU(ADDRESS,MLU_OUTPUT) ! subroutine con funzioni CAMAC che caricano la MLU

END DO

A B C D E F G H I L	Impulso no.	t	
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	Sequenza di bit per generare:
1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	2	60	
1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0	3	120	
1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0	4	180	A: RE DS 7
1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0	5	240	B: STR A ALU 35
1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0	6	300	C: STR B ALU 35
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	7	360	D: ENABLE MLU 25
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	8	420	E: ENABLE MLU 28
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	9	480	F: WE DS di prova
1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	10	540	G: RE DS 17-18
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	11	600	H: STR A ALU 36-37
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	12	660	I: STR B ALU 36-37
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	13	720	L: ENABLE MLU 26-27
1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	14	780	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	15	840	
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	16	900	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	17	960	
1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	18	1020	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	19	1080	
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	20	1140	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	21	1200	
1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	22	1260	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	23	1320	
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	24	1380	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	25	1440	
1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	26	1500	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	27	1560	
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	28	1620	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	29	1680	
1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	30	1740	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	31	1800	
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	32	1860	
1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	33	1920	
1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	34	1980	
0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0	35	2040	
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	36	2100	
0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	37	2160	
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	38	2220	
A B C D E F G H I L			

Fig. 6.12 tabella di temporizzazione per il filtro di cluster

La tabella immagazzinata nella MLU 28 è stata realizzata tenendo conto se uno dei valori delle tre informazioni in ingresso superavano una certa soglia e codificando le due cifre binarie in uscita nel seguente modo:

0 : Nessun Cluster

1 : ELM Cluster

2 : HAD Cluster

3 : ELM+HAD Cluster

In Fortran l'algoritmo perciò risulta essere:

```
DO ADDRESS=0, LAST_ADDRESS
  WORD = 0
  ELM.T = JIBITS(ADDRESS,0,5) ! estrae i bit 0 ÷ 4 da ADDRESS
  valore dell'energia elettromagnetica
  HAD.A.T = JIBITS(ADDRESS,5,4) ! estrae i 4 bit dell'energia
  adronica nella parte A
  HAD.B.T = JIBITS(ADDRESS,9,4) ! estrae i 4 bit dell'energia
  adronica nella parte B
  IF (ELM.T.GE.4) THEN ! controlla la soglia
    ELM.CLUST = 1
  ELSE
    ELM.CLUST = 0
  ENDIF
  IF (HAD.A.T.GE.4.AND.HAD.B.T.GE.4) THEN ! controlla la soglia
  per HAD
    HAD.CLUST = 1
  ELSE
    HAD.CLUST = 0
  ENDIF
  WORD = ELM.CLUST + HAD.CLUST * 2 ! parola da caricare nella
  MLU
  CALL LOAD_MLU(ADDRESS,WORD)
ENDDO
```

Una volta inizializzate le MLU, il programma (App. B) scrive nelle tre Data stack dei dati generati casualmente che rappresentano i 3 vettori $R\phi$, con cui si controlla se il blocco, con le temporizzazioni scelte, esegue correttamente le operazioni. Il blocco ha funzionato subito bene, e facendolo lavorare per alcune ore consecutive non ha mai mancato di identificare un *cluster*.

7 Conclusioni

Come abbiamo visto il lavoro di preparazione al montaggio definitivo e messa a punto del filtro in energia ha dato dei risultati molto interessanti.

Per quanto riguarda il punto di vista dello *hardware* lo studio e la caratterizzazione temporale del comportamento dei moduli ha permesso di trovare un "algoritmo" intelligente per la loro disclocazione e tale da permettere un disegno dettagliato dello *hardware processor*, mentre le prove dei blocchi funzionali hanno permesso la correzione di imperfezioni del progetto primario specialmente per quanto riguarda i problemi di sincronizzazione dei moduli e la conoscenza più approfondita del comportamento di alcuni moduli utilizzati in configurazioni complesse.

Dal punto di vista del *software* l'aver scritto ed utilizzato un insieme di programmi per il controllo della funzionalità dei blocchi e dei moduli ci ha resi consapevoli di quali dovranno essere le caratteristiche del software finale. Infatti ora é chiaro che i dati da utilizzare per il controllo del funzionamento del filtro saranno quelle "pattern" utilizzate nelle prove dei singoli moduli, opportunamente adattate, che consentono prove veloci e significative, e che i programmi dovranno avere un alto grado di interattività per poter svolgere bene le loro funzioni. Ora é noto anche quante e che tipo di informazioni sarà necessario memorizzare in un Data Base che fornirá tutte le notizie sul filtro modulo per modulo. Da questo Data Base i programmi di controllo

ed inizializzazione potranno prelevare tutte le informazioni necessarie quali il numero della stazione di un determinato *crate* occupata da un modulo, le sue caratteristiche e funzioni espletate nel filtro, le sue connessioni con altri moduli etc.

Quindi ora é possibile iniziare la stesura dei programmi che verranno usati durante il montaggio e messa a punto finale nonché quelli che verranno utilizzati per il controllo delle operazioni durante la presa dati, mentre il lavoro di studio preliminare della disposizione (layout) dei moduli e dei cavi nei 20 "crates" é ormai praticamente finito.

BIBLIOGRAFIA

- [1] *F. Cesaroni, S. Di Marco, E. Gennari, S. Gentile* - A general purpose arithmetic and logic unit - Accettato da Nuclear Instruments Methods (NIM)
- [2] *M. Vincelli, L. Levit* - A modular implementation of sophisticated triggers - NIM A235 (1985) 396-406
- [3] Camac, un sistema modulare di strumentazione per l'elaborazione dati - ESONE
- [4] *G. Pallottino, S. Cantarano* - Elettronica dei convertitori - Etas Libri
- [5] *A. Serra* - Elettronica numerica - Veschi editrice
- [6] *R. Bizzarri, F. Cesaroni, S. Gentile, P. Pacchiarotti* - Energy trigger basic plan : first update - 24 giugno 86
- [7] *A. Carter et Alii* - Fast track trigger for the OPAL experiment - NIM A250 (1986) 503-513
- [8] Fastbus, a modular high speed data acquisition system for high energy physics and other applications - ESONE/FB/01
- [9] L3 data acquisition software - 20 sett. 83 - nota no. L3J-6
- [10] L3 data acquisition system - nota no. L3J-2
- [11] L3 letter of intent - 2 gen. 82
- [12] L3 level 1 energy trigger - 6 dic. 84 - nota no. L3J-1
- [13] L3 level 1 energy trigger version 2 - 6 agosto 86 - nota no. L3J-7
- [14] L3 level 1 muon trigger - 23 ott. 85 - nota no. L3J-5

- [15] L3 technical proposal - maggio 83
- [16] *J. Millman, C. Halkias* - Microelettronica - Edizioni Boringhieri
- [17] *J. Ellis, R. Peccei* - Physics at LEP - CERN 86-02 vol. 1 (1986)
- [18] *B. Borgia, F. Cesaroni, E. Petrollo* - Preliminary proposal for fast energy trigger
- [19] *A. O. Allen* - Probability, statistics and queuing theory - Academic press
- London 1978
- [20] *A. Fucci, B. Martin et alii* - The 3081/E processor - CERN DD DD/84/4
- [21] *R. Bizzarri, F. Cesaroni* - The level 1 energy trigger for L3 experiment
- nota no. Rome L3J-3
- [22] The second level trigger for L3 - 3 mar. 85 - nota L3J-3
- [23] *Ph. Gavillet* - Trends in new collider experiment data acquisition systems - NIM A235 (1985) 363-379
- [24] Trigger and data acquisition system of L3 - CERN/LEPC/84/5
- [25] *P. Bahler, N. Bosco, T. Lingjaerde* - XOP : a very high speed processor for trigger applications - CERN DD DD/84/12
- [26] *J. D. Jackson* - Classical electrodynamics - J. Wiley New York 1975
- [27] *R. Bizzarri, F. Cesaroni, G. Ciapetti et alii* - On-line fiducial volume trigger for a small, rapid cycling bubble chamber - NIM 222 (1984) 441-447
- [28] *W. von Ruden* - Trigger and data acquisition plans for LEP experiments
- CERN DD/85/16

- [29] The UA1 calorimeter trigger - CERN DD/84-17
- [30] Proceedings CERN school of computing. 1984 Aiguablava - (CERN 85-09)
- [31] Proceedings, VMEbus in phisic conference - (CERN 86-01)
- [32] *Niklaus Wirth* - Principi di programmazione strutturata - ISEDI Arnoldo Mondadori Editore
- [33] *C. Ausiello, A. Marchetti* - Teoria e progetto di algoritmi fondamentali - Franco Angeli editore
- [34] *A. Celentano* - Il Fortran 77 - Edizioni CLUP Milano
- [35] *S. Cantarano G.V. Pallottino* - Elettronica integrata - ETAS libri Fabbri Editore
- [36] *Manfredi, Rimini* - Teoria delle reti lineari - casa editrice Ambrosiana Milano
- [37] *Manfredi, Maranesi, Tacchi* - L'amplificatore operativo - Boringhieri
- [38] Motorola Semiconductors - MECL device data
- [39] Texas Instruments - The TTL data book
- [40] Monolithic Memories - PAL Programmable Array Logic

Appendice A:
Specifiche tecniche dei moduli

LeCroy



CAMAC Model 4300

16 Channel, Fast Encoding and Readout ADC (FERA)

- **High density:** 16 channels in a single width standard CAMAC module.
- **Fast conversion:** 11 bits in 8.5 μ sec or 10 bits in 4.8 μ sec, 9 bits in 2.8 μ sec.
- **High sensitivity:** 11 bits, 480 pC full scale; 9 or 10 bits, 256 pC full scale.
- **Floating common signal ground:** to eliminate sensitivity to common mode noise (hum) and DC offset.
- **50 or 100 Ω input impedance:** compatible with flat twisted pair cables or 50 Ω multi-coaxial flat cables.
- **Pedestal compensation:** minimizes the gate width effect on the pedestal.
- **Internal pedestal memory:** programmable pedestal subtraction during readout.
- **Programmable zero suppression:** compresses data and accelerates readout.
- **Programmable high speed readout:** 100 nsec/word via front panel ECL port.
- **Sequential or random access CAMAC readout.**
- **Common test feature.**
- **System compatibility:** designed for operation with fast buffer memories, bit slice processors, look up memories, data stacks and other ECL trigger processing units.

The LeCroy Model 4300 Fast Encoding and Readout ADC (FERA) is designed to cover applications where charge measurements and readout must be performed rapidly and with high resolution. The FERA may be used as a single standard CAMAC module, or as part of a large system.

The FERA is a high speed charge integrating analog-to-digital converter. Each module contains 16 independent ADCs that are enabled by a common gate of 50 nsec to 500 nsec duration. Larger gate widths are possible after pedestal adjustment. The inputs, designed to accommodate 17-pair flat cables, offer a common mode rejection of ± 200 mV.

After conversion, the digitized data may be automatically corrected for pedestal with values contained in the internal programmable pedestal memory. Digitized data is available first on the front panel ECL port and, subsequently on the CAMAC dataway. The ECL port readout is optional. All zero data words may be suppressed separately for the ECL port or CAMAC readout to provide data compression. The compression procedure requires 1.6 μ sec (both after the conversion and ECL port readout).

Copyright © March, 1984 by LeCroy Research Systems Corporation



Innovators in Instrumentation

X

The front panel bus system includes the protocol necessary to permit high speed sequential readout ECL compatible modules including the LeCroy Models 2375 Data Stack Module, 2372 Memory Look Unit, and 4302 Triple Port Fast Memory. The Model 4302 is the memory extension of the LeCroy 48 series of Fast Bit Slice CAMAC Processors which permits the distribution of programmable intelligence within the CAMAC system.

The readout modes, pedestals and remote testing of the Model 4300 FERA are controlled via CAMAC. Gate, fast clear, test signals and ECL port timing for up to a crate of FERAs, may be controlled from Model 4301 FERA Driver.

SPECIFICATIONS

CAMAC Model 4300

16 Channel, Fast Encoding and Readout ADC (FERA)

ANALOG INPUTS (16)

Connector:	17 x 2-pin front panel connector (BERG 75789-101-34). The upper pins of the left row are negative signal inputs. The upper 16 pins of right row are connected to the common virtual ground (AC coupled ground). The lowest 2 pins are connected to ground.
Input Sensing:	Charge (current integrating).
Full Scale:	256 pC (9 or 10 bits), 480 pC (11 bits).
ADC Resolution:	Two factory options: 9 bits or 10/11 bits; the 10/11-bit selection is made via 4 internal jumpers and adjustment by an internal potentiometer.
Conversion Time:	Typically 9 bits in 2.8 μ sec, 10 bits in 4.8 μ sec, 11 bits in 8.5 μ sec.
Range:	9 or 10 bits typically 256 pC minus ADC pedestal. 11 bits typically 480 pC minus ADC pedestal.
Sensitivity:	10 or 11 bits 0.25 pC/count $\pm$ 3%, 9 bits 0.5 pC/count $\pm$ 3%.
Input Impedance:	Two factory options: 100 Ω $\pm$ 3%, within the range of 0 to -30 mA or 50 Ω $\pm$ 3%, within the range of 0 to -60 mA DC. Above these limits diode protection clamping will affect input impedance.
Input Protection:	$\pm$ 25 V for 1 μ sec transients (clamping diodes to ground and -3 V).
Input Limitations:	Maximum current for linear response -30 mA. With 50 Ω input impedance, the linearity is degraded to typically $\pm$ (1% of reading + 0.25 for -60 mA.
Common Mode Properties:	Common mode rejection ratio > 50 dB for $\pm$ 200 mV (DC to 1 kHz).
ADC Isolation:	> 50 dB.
Integral Linearity:	Typically $\pm$ 0.5 pC. Worst case $\pm$ (0.25% of reading + 0.5 pC) for signals of slew rate mA/nsec. For signals of slew rate 4 mA/nsec, linearity is degraded typically $\pm$ (1% of reading + 0.25 pC).
Differential Linearity:	Typically $\pm$ 10%, worst case $\pm$ 20%.
Residual Pedestal:	Minimum 1 pC and maximum 13 pC for a gate width from 50 to 500. and all inputs open. Adjustable with an internal potentiometer for width > 500 nsec. Subtracted from data by CAMAC command.

Pedestal/Gate Width Coefficient:	$< \pm 8 \text{ pC}/\mu\text{sec}$, typically $\pm 3 \text{ pC}/\mu\text{sec}$.
Operating Temperature:	0° to 40° C.
Temperature Coefficient:	Typically $(-0.05\% \text{ of reading} \pm 0.1 \text{ count})/^{\circ}\text{C}$ for a gate width of 500 nsec. Coefficient may vary slightly for other gate widths.
Long Term Stability:	$\pm (0.25\% \text{ of reading} + 0.5 \text{ pC})/\text{week}$ at constant temperature and voltage.

COMMAND BUS

Connector:	8 x 2-pin front panel connector. The input matching resistors and output pull down resistors may be removed for high impedance inputs and outputs. When these resistors are in place, a front panel LED (PD ON) is lit.
Input Levels:	Differential ECL levels, 100 Ω impedance differential.
Output Levels:	Differential ECL levels (into 100 Ω differential).
Gate Input (GATE):	One, common for all ADCs. Non-retriggerable. Gate width 50 to 500 nsec ($> 500 \text{ nsec}$ is possible after pedestal adjustment). Gate must precede input signals by at least 20 nsec.
Clear Input (CLR):	One, common for all ADC front ends and digital logic. May be executed at any time. Minimum clear width: 5 nsec. CLEAR settles to within 1 count in less than 2 μsec during conversion; the action is immediate after conversion.
Request Output (REQ):	One, ECL port readout request indicates that this module is ready to take control of ECL port data transfer. After conversion time, REQ is set only if the ECL port readout is programmed and the module contains valid data.
Write Strobe Output (WST):	One, indicates when the data is valid on the ECL port output. WST is set in a minimum of 10 nsec after the data is presented to the ECL port (settling time) and released when the write acknowledge is received. The ECL port data is stable during entire WST pulse. Minimum write strobe width 40 nsec.
Write Acknowledge Input (WAK):	One, acknowledge signal from the ECL port receiver indicating data has been loaded and the next data word may be sent. The next WST is set 50 nsec after the release of WAK. Minimum write acknowledge width 30 nsec.
Test Reference Voltage Input (TRV):	Two paralleled front panel 2-pin connectors, high input impedance (200 k Ω), accept the Test Reference Voltage; second connector usable for monitoring or daisy chaining. The gate and charge pulses are generated at the input of all ADCs by the CAMAC function $F(25) \cdot A(0)$. The test charge pulse is proportional to the DC level at the TRV input. Channel-to-channel matching of the proportionality constant is $\pm 1\%$. TRV input range: 0 to + 10.24 V, equivalent to 0 to 512 pC $\pm 0.1\%$ or each ADC. The common virtual ground offset is automatically compensated.

ECL PORT ENABLE / PASS

Readout Enable
Input (REN):

1 × 2-pin front panel connector. Accepts differential ECL levels. Input impedance 100 Ω differential.

Indicates to this module that it can take control of the ECL Port Bus. The REN signal must be maintained during entire readout time. REN enables the ECL port outputs, WST output and WAK input if the module is ready for data transfer (REQ output ON).

Pass Output (PASS):

1 × 2-pin front panel connector, generates differential ECL levels (into 100 Ω differential). Indicates that this module has finished using the ECL port or is not ready. PASS output is activated by the REN input signal and closed by the REQ output internal command.

Transit time between REN input and PASS output typically 3 nsec if module empty.

ECL PORT OUTPUT

Connector:

17 × 2-pin front panel connector; (BERG 75789-101-34). The last 2 pins are not connected.

Output Levels:

Differential ECL levels (into 100 Ω differential).

The pull down resistors may be removed for high impedance outputs. When these resistors are in place, a front panel LED (PD ON) is lit.

Specifications:

Data word size: 16 bits. Sequential data readout with maximum output frequency 10 MHz (see Readout Block Format).

CAMAC COMMANDS

Note: after a GATE, the module is BUSY (=1), and only the data readout functions are enabled. The BUSY status is released (=0) only after a clear CAMAC function or front panel CLR signal.

Z:

Initializes module; clears the module and sets the seven command bits of the status register to 1.

C:

Clears the module.

I:

Inhibits the front panel GATE during CAMAC inhibit command.

X:

X response is generated for all valid functions.

Q:

Q response is generated only if a function can be executed.

L:

Look-At-Me (LAM) set, if enabled, after the end of conversion (or ECL port readout) and if there is valid data to be read. LAM is cleared by Z, C, F(9), F(10), front panel CLR or, if sequential read is selected, by F(2) after the last data has been read.

F(0)•A(0)

Read status word register. R1 to R15; R16 = 0; Q = 1 if BUSY = 0.

R1 to R8:

VSN; Virtual Station Number; index source for sequential readout with zero suppression.

R9: EPS; ECL port Pedestal Subtraction: when EPS = 1, pedestals are subtracted for ECL port readout.

R10: ECE; ECL port data Compression Enable; when ECE = 1, data zeros are suppressed for ECL port readout.

R11: EEN; ECL port ENable: when EEN = 1, ECL port readout is permitted.

R12: CPS; CAMAC Pedestal Subtraction: when CPS = 1, pedestals are subtracted for CAMAC readout.

R13: CCE; CAMAC data Compression Enable: when CCE = 1, data zeros are suppressed for CAMAC sequential readout.

R14: CSR; CAMAC Sequential Readout: when CSR = 0, CAMAC random access readout is enabled; when CSR = 1, CAMAC sequential readout is enabled.

R15: CLE; CAMAC LAM Enable: when CLE = 1, LAM is enabled.

Note: The seven bits, EPS to CLE, are set to 1 by the CAMAC Z function.

F(1)•A(0-15):

Reads pedestal memory on R1 to R8; R9 to R16 = 0.
Random access to the 16 pedestal words.
Channels 0 to 15 are addressed by subaddresses 0-15.
Q = 1 if BUSY = 0.

F(2)•A(0-15):

CSR = 0; random access to the 16 ADC values. Reads ADC value on R1 to R11; R12 to R16 = 0. Channels 0 to 15 are addressed by subaddresses 0-15. Q = 1 if BUSY = 1 and data are ready.

CSR = 1; sequential readout of ADC values. Reads ADC value on R1 to R16 (see Readout Block Format). The increment to the next word is triggered by S2. Q = 1 if BUSY = 1 and as long as valid data to be read are present. The subaddress A is not decoded.

F(8)•A(0):

Tests Look-At-Me. Q = 1 if LAM is present.

F(9)•A(0):

Clears the module. Q = 1.

F(10)•A(0):

Tests and clears LAM. Q = 1 if LAM is present; LAM reset at S2 if set.

F(16)•A(0):

Writes status word register. W1 to W15; W16 not used; see F(0) for assignment. Q = 1 if BUSY = 0. The status word register is loaded at S1.

F(17)•A(0-15):

Random access for writing of the 16 pedestals. Writes pedestal memory on W1 to W8; W9 to W16 not used. Channels 0 to 15 are addressed by subaddress 0-15. Q = 1 if BUSY = 0. The 8-bit pedestal word is loaded at S1.

F(25)•A(0):

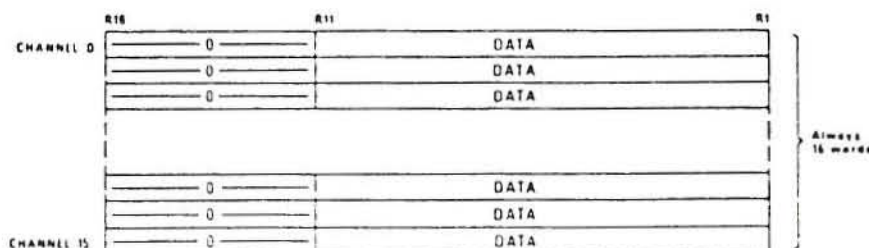
Enables test. Q = 1 if BUSY = 0.
The gate is opened for 550 nsec at S2. The TRV input generates a charge proportional to the TRV for each ADC. Maximum rate 10 kHz; recovery time 100 μ sec.

READOUT BLOCK FORMAT

Without Zero
Suppression:

ECL port readout;
CAMAC readout;

EEN = 1, ECE = 0.
CSR = 1, CCE = 0.

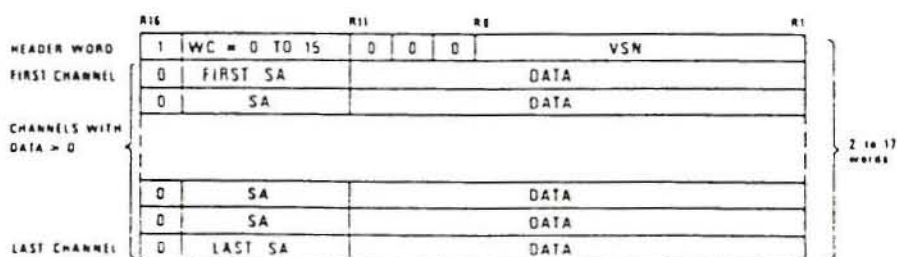


DATA: 9-bit resolution; 0 to 511;
overflow = 2047.
10-bit resolution; 0 to 1023;
overflow = 2047.
11-bit resolution; 0 to 1919;
overflow = 2047.

With Zero
Suppression:

ECL port readout;
CAMAC readout;

EEN = 1, ECE = 1.
CSR = 1, CCE = 1.



VSN: Virtual Station Number loaded in the Status Register.
WC: Number of data words following the header word, 0 indicates 16 data words.
DATA: 9-bit resolution; 1 to 511;
overflow = 2047.
10-bit resolution; 1 to 1023;
overflow = 2047;
11-bit resolution; 1 to 1919;
overflow = 2047;
SA: Channel SubAddress.

GENERAL

Packaging:

RF-shielded, #1 width CAMAC module.

Power Requirements:

2.1 A at +6 V; 2.7 A at -6 V

0.1 A at +24 V (65 mA plus 1.5 times average input current).

Note: when all output pull down and input matching resistors are removed, the current at -6 V is reduced to 2.4 A.

SPECIFICATIONS SUBJECT TO CHANGE

The LeCroy logo is displayed in a stylized, bold, sans-serif font. It is centered within a white rectangular box that is part of a larger graphic element consisting of a thick black crosshair.

CAMAC Model 4301

Fast Encoding and Readout ADC (FERA) System Driver

- Provides all utility functions for a full crate of LeCroy Model 4300 FERAs including:
 - Gate and Clear signals distributed to all FERAs.
 - Write Acknowledge used in ECL Bus transfers distributed to all FERAs.
 - Bus Request for the ECL Bus received from all FERAs, available on the front panel.
 - Write Strobe for the ECL Bus transfers received from all FERAs, available on the front panel.
 - Variable Test Reference Voltage distributed to all FERAs for testing and calibration. Generated from CAMAC controlled 12-bit DAC.
- ECL and NIM Inputs and Outputs.
- 16-Channel Single-Ended ECL to Differential ECL Converter.

The Model 4301 Fast Encoding and Readout ADC System Driver provides a simple mechanism for the distribution and collection of common signals for up to an entire crate of Model 4300 FERAs. The functions include all ADC control functions such as Gate and Clear as well as all timing and control functions for the ECL Port.

An internal CAMAC controlled Digital-to-Analog Converter (DAC) permits precise generation of reference voltage levels for test and calibration of the Model 4300 FERAs.

The Model 4300 FERAs use an ECL Port for 10 MHz readout of the 16-bit data words. When bussed together, these standard differential ECL signals become single-ended ECL. The signals must be converted back to differential ECL before being sent to a standard ECLine logic module or memory unit. This translation is conveniently handled by the Model 4301. To permit simple interconnections, connectors are located at the same position as those of the Model 4300 ECL Port and the Model 4302 Triple Port Fast Memory Module.

The Model 4301 FERA Driver uses two separate ECL Busses. The Command ECL Bus is used to receive and distribute signals used by a set of FERAs via an 8 x 2-pin connector. The Data ECL Bus is connected to the ECL Ports of the Model 4300 FERAs and carries the data during fast readout. The Data ECL Bus uses a standard ECLine 17 x 2-pin connector for both input and output interconnections. LeCroy recommends one Model 4301 be used with up to 22 Model 4300 FERAs, a full CAMAC crate.

Copyright © January, 1984 by LeCroy Research Systems Corporation

Innovators in Instrumentation

SPECIFICATIONS

CAMAC Model 4301

FERA DRIVER

INPUT/OUTPUT CHARACTERISTICS

Gate (GAI & GATE):	Two inputs (GAI); Lemo-type connector accepts NIM signal levels (50 Ω input impedance), and 1 $\times$ 2-pin connector accepts differential ECL signals (100 Ω input impedance). Inputs are OR'd and available on the Command ECL Bus (GATE) for distribution to the Model 4300 FERAs.
Clear (CLI & CLR):	Two inputs (CLI); Lemo-type connector accepts NIM signal levels (50 Ω input impedance), and 1 $\times$ 2-pin connector accepts differential ECL signals (100 Ω input impedance). Inputs are OR'd and available on the Command ECL Bus (CLR) for distribution to the Model 4300 FERAs.
Write Acknowledge (WAI & WAK):	Two inputs (WAI); Lemo-type connector accepts NIM signal levels (50 Ω input impedance), and 1 $\times$ 2-pin connector accepts differential ECL signals (100 Ω input impedance). Inputs are OR'd and available on the Command ECL Bus (WAK) for distribution to the Model 4300 FERAs. The WAK may also echo the WST signal after a fixed time delay as described below.
Write Strobe (WST & WSO):	Input via Command ECL Bus (WST) from the Model 4300 FERAs. Two Outputs (WSO); Lemo-type connector generates NIM signal levels into 50 Ω impedance, and 1 $\times$ 2-pin connector generates differential ECL signals into 100 Ω . For synchronous operation, the WSO signal may be connected directly to the WAI via a fixed delay cable or active delay.
Inhibit Readout (IRI):	Two inputs (IRI); Lemo-type connector accepts NIM signal levels (50 Ω input impedance), and 1 $\times$ 2-pin connector accepts differential ECL signals (100 Ω input impedance). Inputs are OR'd and act as a veto for the REO signal described below. Since the REO is usually connected to the first Model 4300 FERA to be read out, the IRI inhibits the readout and may be used to block the readout process until the receiver module is ready.
Readout Request (REQ, RQO & REO):	Input via Command ECL Bus (REQ) from the Model 4300 FERAs. Following a fixed delay, three outputs are generated (factory setting for delay, 200 nsec. may be adjusted by internal potentiometer RQ DEL). The first two outputs (RQO) may be used to signal a request to readout; Lemo-type connector generates NIM signal levels into 50 Ω impedance, and 1 $\times$ 2-pin connector generates differential ECL signals into 100 Ω . The third output (REO) can be vetoed by the IRI signal and is normally connected to the REN or Readout Enabled input of the first Model 4300 FERA to be read out. 1 $\times$ 2-pin connector generates differential ECL signals into 100 Ω .
Test Reference Voltage (TRV):	Internal CAMAC controlled 12-bit DAC generates a precision reference voltage on the Command ECL Bus (TRV), for testing and calibrating the Model 4300 FERAs (see complete description below).
ECL Data Bus (IN & OUT):	16 inputs via 17 $\times$ 2-pin connector (IN); Accepts single ended ECL signals from the ECL Ports of the Model 4300 FERAs. Generates differential ECL outputs on the 17 $\times$ 2-pin connector (OUT) for all 16 signals.

TEST REFERENCE VOLTAGE SPECIFICATIONS

The TRV output on the Command ECL Bus is driven by an internal CAMAC controlled Digital-to-Analog Converter (DAC). The DAC uses a control register which may be set and read via CAMAC commands (see below).

WARNING- the register content is randomly set at power on.

Range:	0 to 10 2375 V corresponding to inputs for the Model 4300 FERA of 0 to 511 875 pC.
LSB:	2.5 mV corresponding to inputs for the Model 4300 FERA of 0 125 pC.
Integral Linearity:	± 0.5 LSB.
Full Scale Error:	± 1 LSB.
Settling Time:	5 μ sec.
Current Limit:	3 mA.

CAMAC COMMANDS

X, Q:	X and Q responses are generated for each valid function.
F(0)*A(0):	Reads the contents of the 12-bit DAC control register on CAMAC lines R1 to R12. X and Q responses are generated.
F(9)*A(0):	Generates a CLR signal on the Command ECL Bus at time S1. Same action as front panel CLI input. X and Q responses are generated.
F(16)*A(0):	Sets the contents of the 12-bit DAC control register from CAMAC lines W1 to W12. X and Q responses are generated.

GENERAL

Packaging:	RF-shielded ± 1 width CAMAC module.
Current Requirements:	100 mA at +6 V, 1.2 A at -6 V. 20 mA at +24 V, 30 mA at -24 V.

SPECIFICATIONS SUBJECT TO CHANGE

CAMAC Model 2372 Memory Lookup Unit

- For trigger processors and complex triggers
- Arbitrary user defined transfer function
- High capacity/speed
- Transparent or strobed mode
- 64 K-bit dual-ported RAM
- Access time 60 nsec
- Battery backup—2 year integrity
- 16-bit front panel Data-in, Data-out connectors

The Model 2372 is a 64 K-bit dual-ported random access memory based upon the Fermilab module ECL-2. Its access time is ≤ 60 nsec. Packaged in a single-width CAMAC module, the unit offers an exceptionally high capacity/speed product while dissipating < 12 W. Battery backup maintains the integrity of the data within the 2372 (for 2 years) when the CAMAC crate is shut off. This continuous memory feature is particularly useful during the startup phases of an experiment. The MLU serves as a fundamental element of a trigger processor. It also may be used to provide complex functions required of trigger logic.

In normal use, the MLU is first downloaded with the required trigger data and then verified via CAMAC. This information, which defines the function of the MLU, is accessed via the front panel for real time applications. The Model 2372 offers 16-bit front panel Data-In and Data-Out connectors, although not all 32 bits may be used at once. The dimensionality of the Model 2372 is selected via CAMAC to serve the application. The dimensionality options are:

DIMENSIONALITY	INPUT WORD SIZE	MEMORY SIZE (WORDS)	OUTPUT WORD SIZE
0	16	64 K	1
1	15	32 K	2
2	14	16 K	4
3	13	8 K	8
4	12	4 K	16

The function of the MLU may be user defined to meet the application. The required function is downloaded via CAMAC. Virtually any function may be defined: angle logic or clusterized track multiplicity are examples of complex trigger functions. When used as a trigger processor element, the MLU may be used for any operation which is a one-to-one mapping. In conjunction with an ADC, the MLU can be loaded with the calibration to energy. The MLU can also be loaded with a digital comparator function or any arithmetic calculation, e.g. $E_1 + E_2$, $E/c \sin\theta$, $(X^2 + Y^2)$.

The strobe facilities of the Model 2372 are used in the software selectable Strobed Mode. In this mode, the MLU is strobed by the ECL Input Enable signals, coming from previous logic unit(s). After 60 nsec, the output "word" of the MLU becomes valid along with four ECL Output Ready signals which may be used to strobe the output word into other modules like the 2372. The output word remains static until another Input Enable is received. This automatically accounts for the propagation delay of the unit. Four Output Ready's are supplied to allow the 2372 to drive multiple logic units. Four Input Enables are provided with an Enable condition being defined as their coincidence (unused inputs are set to logical 1).

The Model 2372 may also be used without regard to the strobe logic. In the Transparent Mode, the output of the Model 2372 may change without regard for the Input Enables. This mode bypasses the latch circuitry and thus provides a slight speed advantage, offering a throughput time of only 55 nsec. This difference is reflected in the Input Enable, Output Ready timing when the Transparent Mode is selected.

October 1982

SPECIFICATIONS

CAMAC Model 2372

MEMORY LOOKUP UNIT

INPUTS

- Input Word:** Up to 16 bits, ECL levels via 17-pair header. Pin-outs match ECLine Standard. Impedance $100\ \Omega \pm 5\%$, high impedance by simple user modification. Minimum width 10 nsec in the Strobed mode or 65 nsec in the Transparent mode. Number of active bits depends upon Dimensionality.
- Input Enable:** Four differential ECL inputs via 2-pin headers. Required only in Strobed mode. Must arrive ≥ 2 nsec after Input word is settled. Minimum width—Strobed mode: 10 nsec, Transparent mode: 65 nsec (if used).

OUTPUTS

- Output Word:** Up to 16 bits, ECL levels via 17-pair header. Pin-outs match the ECLine Standard. Output width depends upon Operating mode. Number of active bits depends upon Dimensionality.
- Output Ready:** Four differential ECL outputs via 2-pin headers. Occurs ≥ 5 nsec after the outputs are settled.
- Propagation Delay:** 60 nsec. Input Enable to Output Ready—Strobed mode. Transparent mode: 20 nsec after a new Input word becomes valid, the Output word becomes indeterminate. The correct Output word appears within 55 nsec of the new Input word.

CONTROL

- Dimensionality Control:** CAMAC selected via CAMAC Control Register (CCR). Defines the number of inputs and outputs.
- Mode Control:** CAMAC selected via CCR.
Strobed Mode—The leading Edge of Enable condition (and of four Input Enables, unused inputs set to logical 1 state) latches Input word. Output Ready's become false within 30 nsec. Output word becomes invalid within 30 nsec. New Output word settled and Output Ready's true within ≤ 60 nsec. Ready and Output word remain until the next Enable condition.
Transparent Mode—operates without regard to Input Enables. Output word present for duration of input word after approximately ≤ 55 nsec delay.
Inhibit Mode—disables front panel inputs to allow CAMAC programming of memory.

MEMORY PROTECTION

- Continuous Memory:** Memory battery back-up. This feature preserves contents of memory and CCR during CAMAC power down. Life of the battery is two years of operation.

CAMAC COMMANDS

- F(0)•A(0):** Read 1 to 16-bit Output word addressed by CAMAC Address Register (CAR). Increment CAR. Requires operation in Inhibit mode. Inactive bits are not masked and are arbitrary.
- F(0)•A(1):** Read 16-bit CAR address. Requires operation in Inhibit mode. Inactive bits are set to 1 if F(16)•A(1) was executed after power up.
- F(0)•A(2):** Read 5-bit CCR word.
- F(0)•A(3):** Read 1 to 16-bit Output word addressed by front panel Input word. Inactive bits are not masked and are arbitrary.
- F(0)•A(4):** Read 12 to 16-bit front panel Input Word.
- F(16)•A(0):** Write 16-bit word into memory location addressed by CAR. Requires operation in Inhibit mode. Write commands are performed with dimensionality of four. See Owner's Manual.
- F(16)•A(1):** Write 12 to 16-bit CAR address. Dataword at ECL output will change accordingly. Requires operation in Inhibit mode.
- F(16)•A(2):** Write 5-bit CCR word.
- Q:** A Q = 1 response is generated for F(16)•A(0) and F(0)•A(0). A Q = 0 response is generated for these commands when CAR reaches terminal count. This Q Response is valid only when CAR has been loaded via F(16)•A(1) after power up.

PRC RAMMING WORDS
CAMAC Control Register (CCR)

D = Dimensionality
M = Mode Control

W5	W4	W3	W2	W1
M		D		

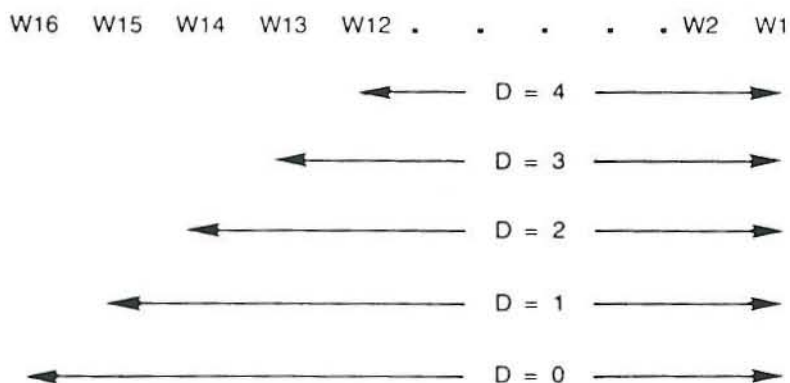
DIMENSIONALITY OPTIONS

D	Dimensionality
0 0 0	0 ; 16 in/1 out
0 0 1	1 ; 15 in/2 out
0 1 0	2 ; 14 in/4 out
0 1 1	3 ; 13 in/8 out
1 X X	4 ; 12 in/16 out

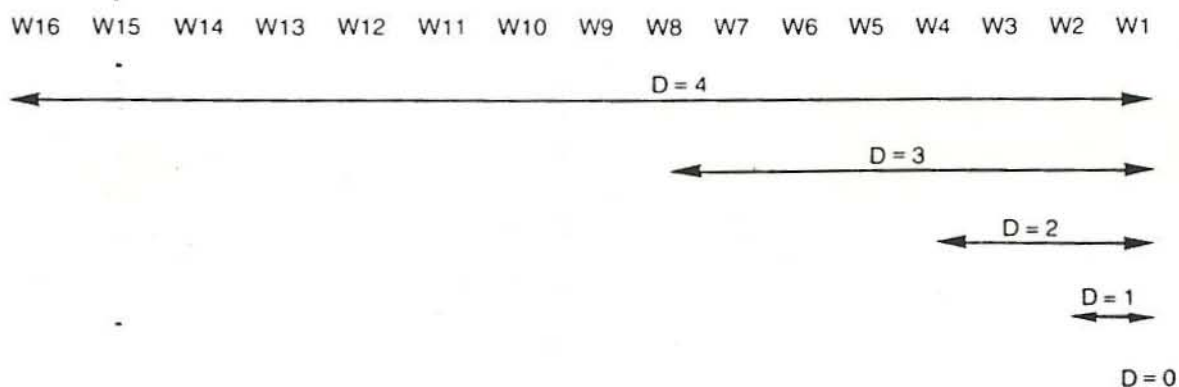
MODE OPTIONS

M	Mode
0 0	Strobed
0 1	Transparent
1 X	Inhibit

CAMAC Address Register (CAR)



Output Word (OW)



GENERAL
Packaging:

In conformance with CAMAC Standard for Nuclear Modules (ESONE Committee Report EUR4100). RF shielded CAMAC 1 Module.

Power:

< 1.25 A at +6 V
< 850 mA at -6 V

Specifications SUBJECT TO CHANGE



ECLine Model 2375 Data Stack Module

- Trigger Processing
- Multiple Event Buffer
- High Speed: 50 nsec cycle time
- High Density: 256 sixteen-bit words
- Dual Ported: simultaneous read/write
- Self Sequencing

The Model 2375 is a Data Stack designed for use with the LeCroy family of ECLine Trigger Processor modules. It is a fast 256×16 memory with separate read and write ECLports, based upon the Fermilab ECL 4 module. A block diagram of the Stack is shown below. The Model 2375 is ideal for rapid processing of lists of data from a variety of sources, such as the ECLport outputs of the PCOS III and FERA system. It may also be used as an Event Buffer in conjunction with these data acquisition systems.

The Write ECLport of the Model 2375 is initialized by a Master Reset (MRST) pulse which resets the internal Read and Write Pointers (RP and WP). The Write ECLport accepts 16-bit datawords at up to a 20 MHz rate. Datawords are latched into the port by an ECL Write Enable (WE) edge and loaded into sequential locations beginning with zero. Fifty nsec later a Write Output Ready (WR) level is asserted. If the write operation is performed during a read operation, the write is delayed by up to 35 nsec. Both operations are correctly performed, however. The upper 3 bits of the dataword may be used by the write Steering Logic to accept or reject the write strobe. A side-panel switch allows each of the eight possibilities to be separately enabled. This allows the stack to be loaded with one type of data (e.g. x) while ignoring another type (e.g. y).

The Model 2375 offers sequential readout. Read operations are initiated by read enable RE. Within 35 nsec, the appropriate dataword is returned along with a Read Ready signal called RR. The Read ECLport may be used without regard to the operation of the Write ECLport. If the read operation is performed during a write operation, the read is delayed by up to 50 nsec. Both operations are correctly performed, however.

Read operations may commence before the write operations have been completed. The All Data In (ADI) signal at the Write ECLport is used to signal that the Write operation is complete.

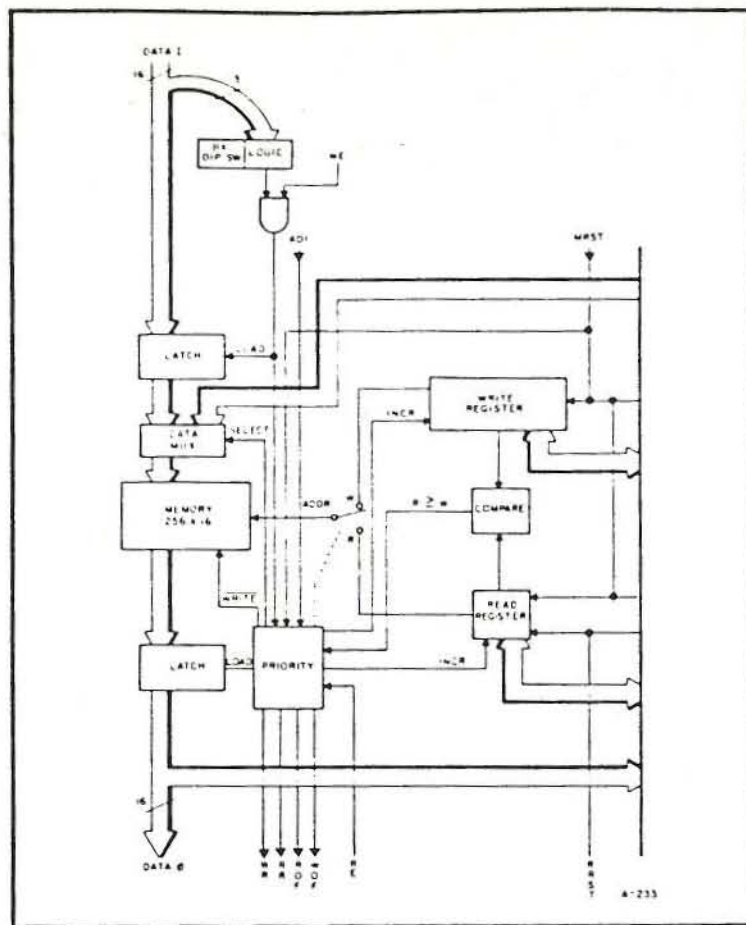
Application of RE pulse returns the dataword and an RR edge and advances the internal Read Pointer. If the Write Pointer is less than the Read Pointer, the Data Stack suspends the dataword response, pending either another write operation or an ADI (all data in) edge. If another dataword is loaded into the Model 2375, it is output in response to the read command alone with an RR edge. If an ADI level is received, Read Overflow (ROF) edge is output. The ROF also resets the Read Pointer so that a subsequent read operation causes the stack to "wrap around" and return the dataword in the bottom of the memory. Stack modules can be wired to act as a series of Nested Do Loops. See Application Note AN-24A.

The Model 2375 is packaged in a single-width CAMAC module. It employs the popular ECLine standard for signals. It provides versatility and sophistication to trigger logic.

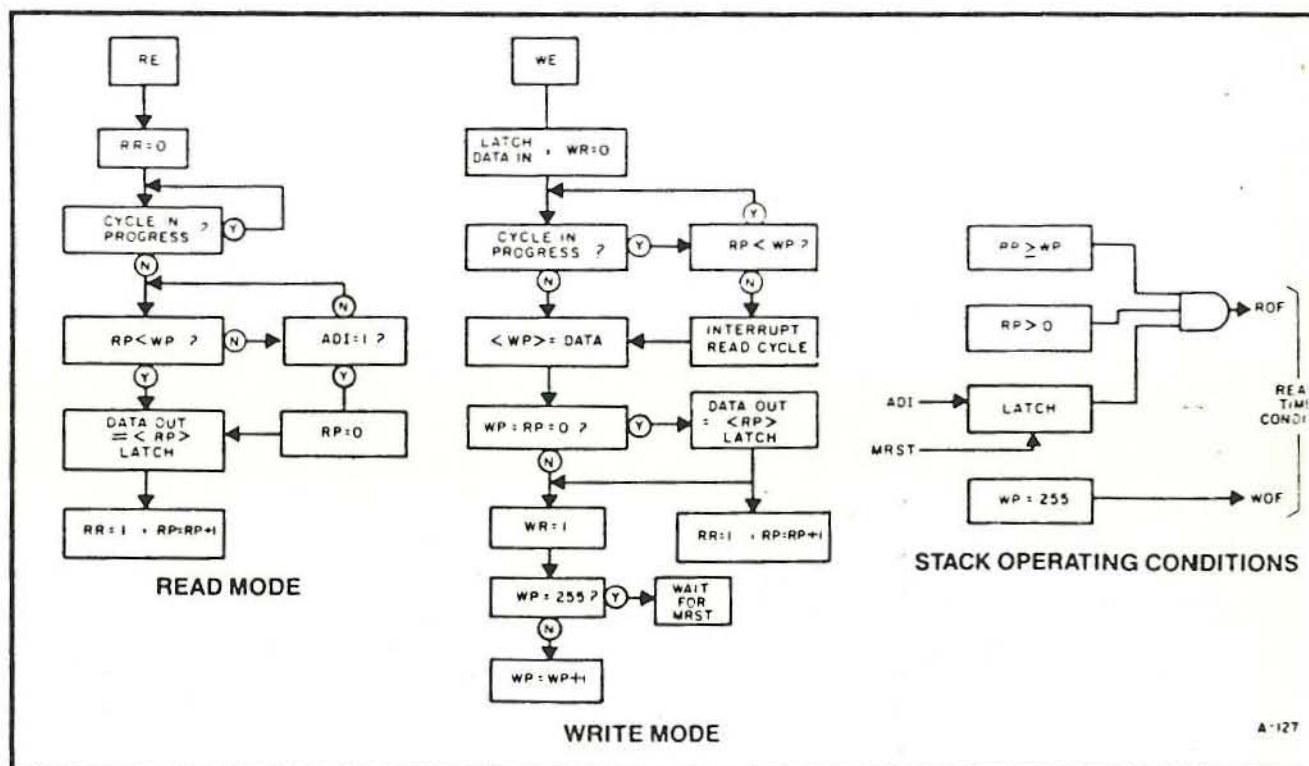
Copyright © June, 1984 by LeCroy Research Systems Corporation

Innovators in Instrumentation

LeCROY RESEARCH SYSTEMS CORPORATION • 700 SOUTH MAIN STREET • SPRING VALLEY, N.Y. 10977
TWX: 710-577-2832 CABLE: LERESCO TELEPHONE: (914) 425-2000



STACK BLOCK DIAGRAM



STACK FLOW CHART

SPECIFICATIONS ECLine Model 2375 DATA STACK MODULE

GLOSSARY OF TERMS

ADI: All Data In—Input
 MRST: Master Reset—Input
 ROF: Read Overflow—Output
 RP: Read Pointer—Internal
 RE: Read Enable—Input
 RR: Read Ready—Output
 RRST: Read Reset—Input
 WOF: Stack Full (write overflow)—Output
 WE: Write Enable—Input
 WP: Write Pointer—Internal
 WR: Write Ready—Output

WRITE ECLport

Write Reset (MRST):

A front-panel differential ECL input via a two-pin connector. Used to set the Write Address Pointer and Read Address Pointer to zero. Minimum width 10 nsec. Reset time 30 nsec.

Write Enable (WE):

A front-panel differential ECL input via two bridged 2-pin connectors. Used to actuate a write operation. The action of the WE leading edge is to write the Dataword presented at the W ECLport and advance the Write Address Pointer by 1. Write operation requires 50 nsec.

W Dataword (DATA I):

A front-panel differential ECL input for a 16-bit dataword. Supplied via an ECLine standard 17-pair header. The Data I must be settled before application of the WE strobe. Minimum duration: 10 nsec.

All Data In (ADI):

A front-panel differential ECL input via two bridged 2-pin connectors. When unused, it is set to the logical 1 condition. An ADI = 0 suspends responses to a Read request if the read address exceeds the Write Address Pointer. A response occurs either when ADI = 1 or after another write operation is performed. If ADI = 1 and the read address exceeds the Write Address Pointer, an ROF pulse is returned and the Read Address pointer is set to zero. Cleared by MRST.

Write Ready (WR):

A front-panel differential ECL output via two bridged 2-pin connectors to indicate the completion of a write operation.

Write Overflow (WOF):

A front-panel differential ECL output level via a 2-pin connector indicates that 256 writes have been performed.

READ ECLport

Read Reset (RRST):

A front-panel differential ECL input via a 2-pin connector used to set the Read Address Pointer to zero. Minimum width: 10 nsec. Reset time 30 nsec.

Read Enable (RE):

A front-panel differential ECL input via two bridged 2-pin connectors. Used to initiate a Stack Read operation and to advance the Read Address Pointer by 1. The leading edge of the RE provides the dataword addressed by the Read Address Pointer at the Read ECLport's Data O Connector. An RR edge is provided 5 nsec after the Data O word is settled. Read operation requires 35 nsec.

R Dataword (Data O):

A front-panel differential ECL output of a 16-bit dataword. Supplied via an ECLine standard 17-pair header.

Read Overflow (ROF):

A front-panel differential ECL output via 2-pin connector. Indicates that the read address exceeds the maximum address written into since the last MRST operation.

Read Ready (RR):

A front-panel differential ECL output via two bridged 2-pin connectors. Indicates that the read operation is complete and that data is valid at the Data O connector.

CAMAC COMMANDS

F(0)•A(0):	Read 16-bit dataword from memory as addressed by Read Pointer (RP). Increment RP.
F(0)•A(1):	Read 8-bit RP
F(0)•A(4):	Read 8-bit Write Pointer (WP)
F(9)•A(0):	Master Reset (MRST)
F(16)•A(0):	Write 16-bit dataword to memory location addressed by WP. Increment WP.
F(16)•A(1):	Write 8-bit RP
F(16)•A(4):	Write 8-bit WP
Q:	A Q = 1 response is generated for F(0)•A(0) and F(16)•A(0). A Q = 0 response is generated for F(0)•A(0) when the $RP \geq WP$ and $RP > 0$. A Q = 0 response is generated for F(16)•A(0) when $WP = 255$.
X:	An X = 1 response is generated for any valid N•F•A.

GENERAL

Packaging:	In conformance with CAMAC standard for nuclear modules (ESONE Committee Report EUR4100 or IEEE Report 583.) RF-shielded CAMAC #1 module.
Power Requirements:	± 6 V, <25 W

SPECIFICATIONS SUBJECT TO CHANGE

ALU MODULE

(Arithmetic and logic unit)

The ALU MODULE is a single width CAMAC standard module which performs eight arithmetic operations and eight logic functions on two 16 bit input words. These operations and functions can be dynamically changed via front panel input signals. The use of an ECL Prom permits to personalize the OP Code: it is possible to choose the group of eight arithmetic operations and the group of eight logic functions among the list in Table#1 and to optimize the number of front panel input commands.

INPUTS:

Note: All inputs accept differential ECL signals, via front panel two pin connectors, 100 Ω input impedance. Removing the 100 Ω input resistor, it is possible to connect inputs of many modules via the same input cable. Unused inputs are set to logical level 0 unless otherwise specified.

Data (A and B):

Two 16-bit inputs via two 2*17 pin ECL standard connectors called A and B.

Strobe A:

Two inputs called A1 and A2. The leading edge of the AND of A1 and A2 strobe pulses latches the A input word on an internal register (reg. A).

Note:

When unused, A2 strobe input is set to a logical level 1; it means it is possible to latch the A input word only with A1 strobe pulse.

Strobe B:

Two inputs called B1 and B2. The leading edge of the AND of B1 and B2 strobe pulses latches the B input word on an internal register (reg. B).

Notes:

- i- When unused, B2 strobe input is set to a logical level 1.
- ii- Depending on the OP Code (see the note of Table#1), the B strobe input can latch the A input word on the register A.

Carry IN:

Used when operations have to be performed on input words wider than 16 bits. The Carry IN accepts the carry information coming from the ALU handling the lower 16 bits.

OP Code:

Four bit inputs; the OP Code defines the operation to be performed; for one ALU MODULE the eight arithmetic operations and the eight logic functions of OP Code are fixed programming an ECL PROM.

OP Code Strobe:

The leading edge of OP Code Strobe pulse latches the OP Code information. The latched OP Code is operative after the coming of the following Strobe A or Strobe B. These Strobes have to come with a minimum delay of 20 nsec. after the OP Code Strobe.

Clear A,B:

Two independent connectors allowing clearing of the input registers A and B.

OUTPUTS:

Note: All outputs generate differential ECL signals suitable for driving 100 Ω twisted pair cables via front panel connectors.

Data (C):

2*17 pin connector provides a 16-bit word (C) resulting from the selected operation performed on the content of register A and register B. The output C is valid at the leading edge of the Data Ready Output pulse and it remains until a new Strobe A or B is applied.

Data Ready Out:

Is generated ,after a delay, from the A and B Strobes depending on the used OP Code.

Carry Out:

Provides a carry signal generated by the operation on A and B. The Carry OUT is valid at the leading edge of the Data Ready Output.

CAMAC COMMANDS AND FUNCTIONS:

- Z or C: Clears the register A, the register B and the OP Code and Status register, enables CAMAC functions.
- X: A X=1 response is generated for any valid CAMAC function.
- Q: A Q=1 response is generated for any executable and enabled CAMAC function.
- F(0)A(0): When CAMAC functions are enabled, reads the OP Code and Status register.
- F(0)A(1): When CAMAC functions are enabled, reads the output C information.
- F(16)A(0): When CAMAC functions are enabled, writes the OP Code and Status information.
- F(24)A(0): Disables CAMAC functions.
- F(25)A(0): When CAMAC functions are enabled, generates a Strobe A and a Strobe B pulses
- F(26)A(0): Enables CAMAC functions.

OP CODE AND STATUS REGISTER:

Note: After power switch on, the content of this register is set to 0.

- bit # 1,4 - OP CODE bits.
- bit # 5 - when "1" the front panel OP CODE input is enabled.
- bit # 6 - this bit has to be set to "1" if operations are performed on large words and the module does not hand the lower 16 bits.
- bit # 7 - when bit#7 and bit#5 are set to "1" the front panel OP CODE information is latched on OP CODE register via the OR of Strobe A and B pulses and not via OP CODE Strobe input signal.

GENERAL:

- Maximum frequency: Data (A and B), OP Code IN, Strobes 25 MHz.
- Propagation Delay: Data IN to Data OUT 40 nsec.
- Packaging: Single width CAMAC standard module.

TABLE # 1

ARITHMETIC OPERATIONS:

- C = B plus A
- C = B plus previous result C (sum and accumulation)
- C = A minus B
- C = A minus previous result C (subtraction and accumulation)
- C = 2 times A
- C = A minus 1
- C = A plus (A AND B)
- C = A plus (A OR B)
- C = A plus (A AND INV B)
- C = A plus (A OR INV B)
- C = (A OR B) plus (A AND INV B)
- C = (A AND B) plus (A OR INV B)
- C = (A AND B) minus 1
- C = (A AND INV B) minus 1

LOGIC FUNCTIONS:

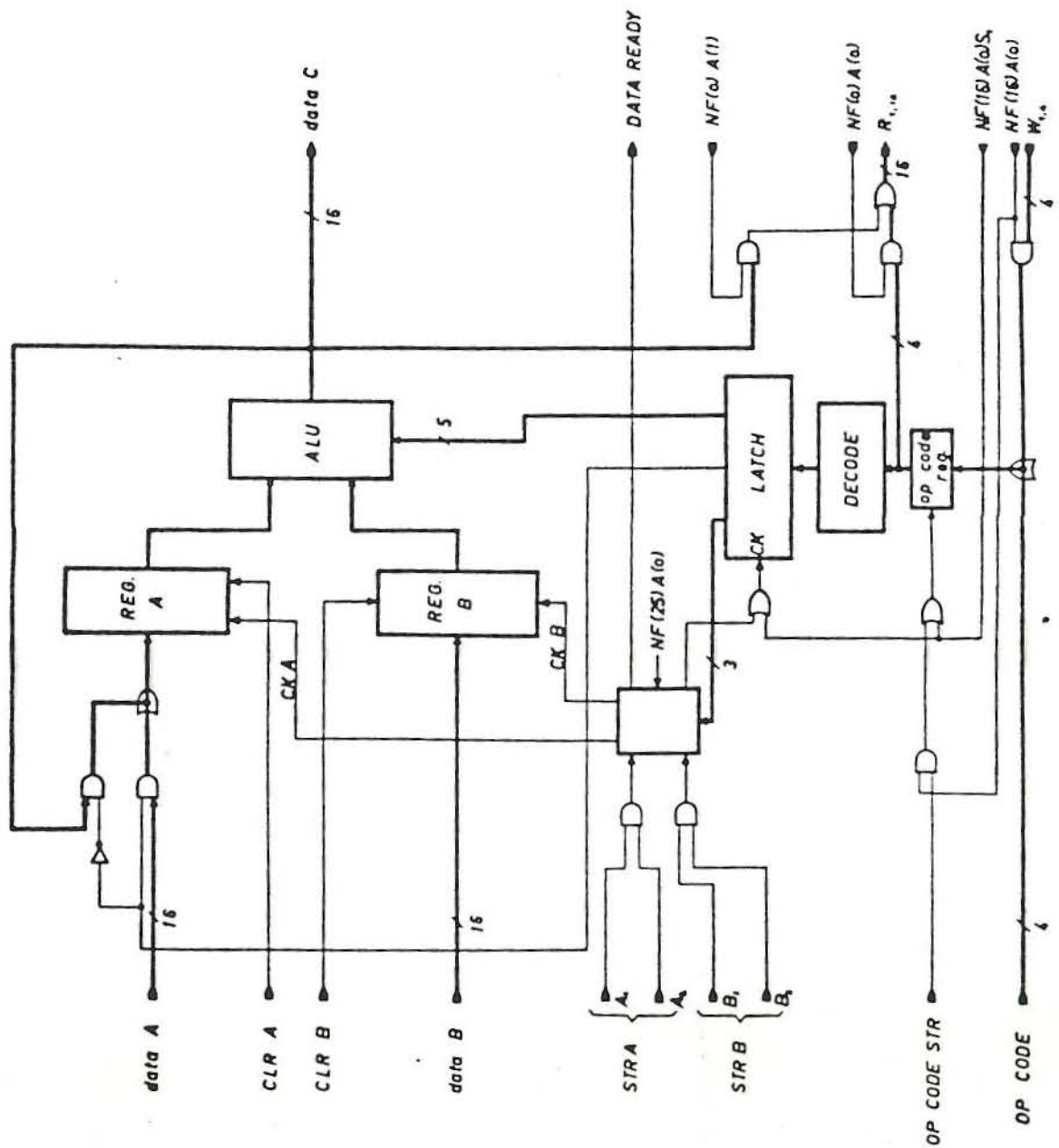
- C = A
- C = B
- C = INV A
- C = INV B
- C = all "1"
- C = all "0"
- C = B AND A
- C = B AND INV A — $\bar{B} + 1$
- C = A AND INV B
- C = (INV A) AND INV B
- C = B OR A
- C = B OR INV A
- C = A OR INV B
- C = (INV A) OR INV B
- C = A EXOR B
- C = A EXAND B

Notes:

- i- Only the Strobe B has to be used for the two operations with accumulation. For the other functions it is possible to choose between two OP Code: with the first one, Strobe A and B pulses latch respectively data A and B on registers A and B; with the second one, Strobe B pulse latches both data A and B on registers A and B.
- ii- OR, AND, EXOR, EXAND, INV are logic operations performed bit by bit.

TABLE # 2
(Suggested OP Code for L3 experiment)

OP Code #	Function	Data A to Reg.A	Data B to Reg.B
0.....	C=B+previous C	Strobe A.....	Strobe B
1.....	C=B+A.....	Strobe A.....	Strobe B
2.....	C=A+B.....	Strobe B.....	Strobe B
3.....	C=A-B.....	Strobe A.....	Strobe B
4.....	C=A-B.....	Strobe B.....	Strobe B
5.....	C=2 times A.....	Strobe A.....	Strobe B
6.....	C=A minus 1.....	Strobe A.....	Strobe B
7.....	C=B-previous C.....	Strobe A.....	Strobe B
8.....	C=B.....	Strobe A.....	Strobe B
9.....	C=A.....	Strobe A.....	Strobe B
10.....	C=all "0".....	Strobe A.....	Strobe B
11.....	C=all "1".....	Strobe A.....	Strobe B
12.....	C=A AND B.....	Strobe A.....	Strobe B
13.....	C=A AND B.....	Strobe B.....	Strobe B
14.....	C=A OR B.....	Strobe A.....	Strobe B
15.....	C=A EXOR B.....	Strobe A.....	Strobe B



BUS SWITCH MODULE
-----INPUTS:

Note: All inputs accept differential ECL signals, via front panel two pin connectors, 100 Ω input impedance. Removing the 100 Ω input resistor, it is possible to connect inputs of many modules via the same input signal cable. Unused inputs are set to logical level 0 unless otherwise specified.

Data (A and B):

Two 16-bit inputs via two 2*17 pin ECLine standard connectors called A and B.

Strobe A:

Two inputs called A1 and A2. The leading edge of the AND of A1 and A2 strobe pulses latches the A input word on an internal register (reg. A).

Note:

When unused, A2 strobe input is set to a logical level 1: it means it is possible to latch the A input word only with A1 strobe pulse.

Strobe B:

Two inputs called B1 and B2. The leading edge of the AND of B1 and B2 strobe pulses latches the B inputs word on an internal register (reg. B)

Note:

When unused, B2 strobe input is set to a logical level 1.

Clear A,B:

Two independent connectors allowing clearing of the input registers A and B.

A/B Switch IN:

The content of input registers A or B is selected when the logic level of A/B Switch Input is 0 or 1 respectively.

Output Gate IN:

A logic level 0 of Output Gate Input sets both differential signals of output C to low level (three state output). When unused, the logic level is 1.

OUTPUTS:

Note: All outputs generate differential ECL signals suitable for driving 100 Ω twisted pair cables via front panel connectors.

Data C:

2*17 pin connector provides a 16-bit word (C) resulting from the selection, via the A/B Switch input signal, of content of register A or B. It is possible the connection of Data C outputs of many Bus Switch Modules with the use of Output Gate Input.

Data Ready Out:

Via patch points, it is possible to choose one or more of the following conditions to perform Data Ready Output pulse:

- 1- Is generated, after a delay, from rising and falling edges of A/B Switch input
- 2- Is performed, after a delay, from the OR of Strobe A and B pulses if they find respectively the A/B Switch Input at 0 or 1 logic level
- 3- Is generated, after a delay, from the rising edge of Output Gate Input pulse.

CAMAC COMMANDS AND FUNCTIONS:

Z or C:	Clears register A and register B, enables CAMAC functions.
X:	A X=1 response is generated for any valid CAMAC function.
Q:	A Q=1 response is generated for any executable and enabled CAMAC functions.
F(0)A(0):	Reads the content of register A when enabled.
F(0)A(1):	Reads the content of register B when enabled.
F(16)A(0):	Writes register A on 16-bit when enabled.
F(16)A(1):	Writes register B on 16-bit when enabled.
F(25)A(0):	Generates a Strobe A and a Strobe B pulses when enabled.
F(24)A(0):	Disables CAMAC functions.
F(26)A(0):	Enables CAMAC functions.

GENERAL:

Maximum frequency: Data (A and B), Strokes - 25 MHz

Propagation Delay: Data IN to Data OUT - 30 nsec.
Switch IN to Data OUT - 20 nsec.
Output Gate IN to Data OUT - 20 nsec.

Packaging: Single width CAMAC standard module

DELAY AND CLOCK GENERATOR

The DELAY AND CLOCK GENERATOR is a single width CAMAC module which generates a square wave (period from 40 nsec. to 190 nsec. with step of 10 nsec.). The square wave is generated between START and STOP front panel input signals. There are four groups of outputs: each group can be delayed from 25 nsec. up to 100 nsec. with step of 5 nsec.

INPUTS:

Start:

Two OR-ed inputs; the first one accepts NIM signals, the second one differential ECL signals.

Stop:

Two OR-ed inputs (NIM and differential ECL signals).

One Shot:

Two OR-ed inputs (NIM and differential ECL signals). It produces one single pulse on the outputs.

OUTPUTS:

Signal Outputs:

Four groups; each group has 8 differential ECL outputs and 1 NIM output.

Busy:

Two outputs (NIM and differential ECL signals)

CAMAC COMMANDS AND FUNCTIONS:

Z or C: Clears registers and flip-flops.

NF(0)A(1): Reads the period word.

NF(0)A(0): Reads the delay word.

NF(16)A(1): Writes the period word (0 for 40 nsec. up to 15 for 190 nsec.)

NF(16)A(0): Writes the delay word (0 for 25 nsec. up to 15 for 100 nsec.):

output group # 1	W1 - W4
output group # 2	W5 - W8
output group # 3	W9 - W12
output group # 4	W13 - W16

ECLINE DRIVER USER'S MANUAL

Preliminary

July, 1986

TABLE OF CONTENTS

- 0. MODULE DESCRIPTION**
- 1. TECHNICAL SPECIFICATIONS**
 - 1.1 Technical data sheet
 - 1.2 Front panel signals
 - 1.3 CAMAC commands
- 2. OPERATING INSTRUCTIONS**
 - 2.1 General advices
 - 2.2 Control and status registers
 - 2.3 Dedicated software interface
- 3. TEST OF UNITS**
 - 3.1 Hardware configuration
 - 3.2 Test program description

SECTION 0

MODULE DESCRIPTION

The ECLine driver module provides two ECLports outputs multiplexed on the standard ECLine bus. Further more, it is designed as a physical switch between the ECLine data stream and simulated datas provided by its own memory and thus consists of two channels with individual frontpanel input and ouput connectors.

In this vue, it offers CAMAC read-write capability in a fast 256 x 16 dual-port memory to ensure generation of programmable data events.

All data lignes, as well as ECLine synchronisation standard signals (RQO-WSO-ADI) will be either generated by the driver or transmitted through the card, according to the function mode set by software. Though, when ECLine transparency is selected, ouputs will strictly follow input signals.

The data transfert will occur simultaneously on both ECLport outputs either on a CAMAC order (internal start) or on synchronisation with an ECL or NIM input signal (external start). A single control word valid for both ports handles data transfert options. All the following data transfert options are encoded in control register and thus may be set by software :

- Data transfert lenght, up to 256 16bit-words/port,
- Synchronisation on internal or external clock ; external clock provided via ECL/NIM clock IN input signal,
- Ajustable data output speed synchronized on internal clock frequency (400, 200, 100, up to 50 ns per dataword),
- Gated or non gated data output.

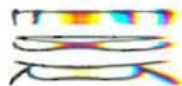
Access to individual channel is provided by CAMAC subaddress A0 and A1.

A block diagram of the ECLine driver is shown on the following page (figure 1). The figure 2 shows the components layout on the card.

The ECLine driver is packaged in a double-width CAMAC module.

C

C



+	-	
.	.	Ground
.	.	IN START ECL
.	.	Ground
.	.	IN CLOCK ECL
.	.	Ground

FRONT PANEL OUTPUT SIGNAL

OUT PORT 1/2

Front panel differential ECL output of a 16-bit dataword on ECLport 1/2 supplied via an ECLine standard 17-pair header

RQ0

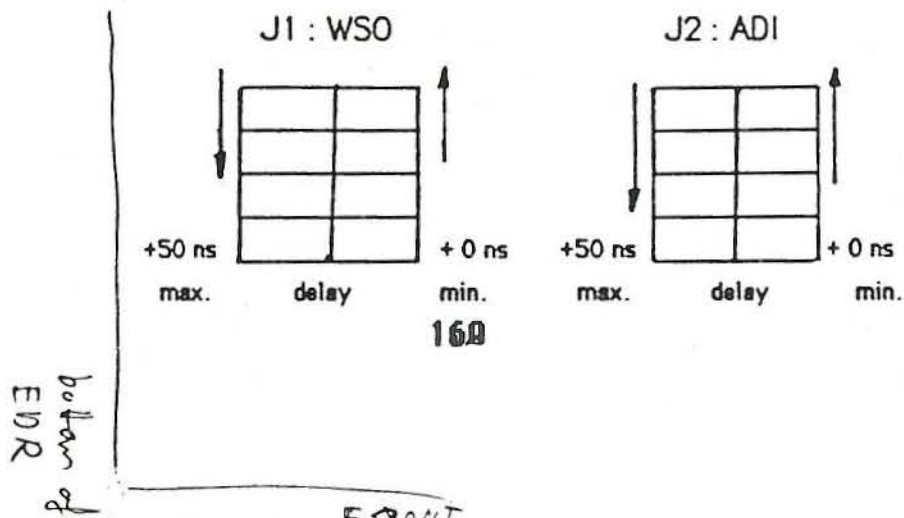
WS0 OUT PORT 1/2

ADI

Front panel differential ECL output of ECLine synchronisation signals associated with data control on ECLport 1/2, supplied via an ECLine 5-pair header connected as follow

+	-	
.	.	RQ0
.	.	Ground
.	.	WS0
.	.	Ground
.	.	ADI

WS0 and ADI output signals width is adjustable from 10 to 60 ns by increment of 5ns up to 50 % of duty cycle, using jumpers J1 and J2 (see figure 2).



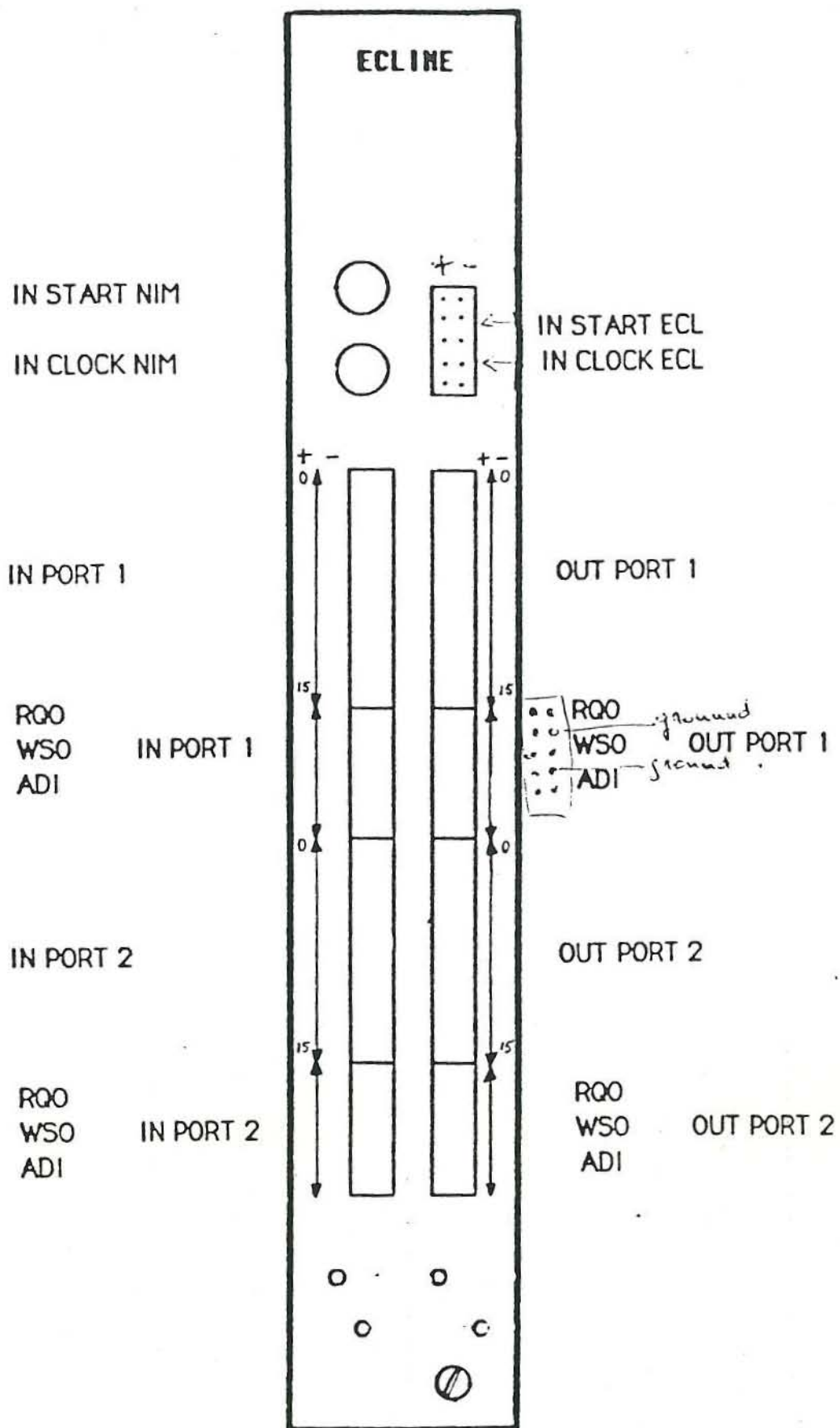


Figure 3. Front panel view

1.3 CAMAC commands:

X = 1

Q = 1 responses are generated on each valid function.

Q = 0 during data transfert; locked on F-25 or Input start signal,
unlocked on ADI signal.

F-9 or Z, or C	A0-A1 (indif.)	- Initiate module. Enable all CAMAC calls except F-25 function (start data transfert); - Set default function mode : ECLine transparency - Reset internal address counter to last value loaded in control register via F-17; do not reset memory contents.
F- 24	A0-A1	- Set driver module on ECLine transparency.
F- 26	A0-A1	- Connect both CAMAC memories output on ECLine bus; - Set driver ready to start transfert of dual-port CAMAC memory contents; - Disable CAMAC access to individual port 1/2 via CAMAC subaddress A0/A1.
F-16	A0/A1	- CAMAC write of a 16-bit dataword on port 1/2 to memory location addressed by internal counter; increment address.
F-0	A0/A1	- CAMAC read of a 16-bit dataword on port 1/2 to memory location addressed by internal counter; increment address.
F-17	A0-A1	- Write a 16-bit control word in control register (single location for both ports).
F-1	A0-A1	- Read status register contents; report current address from internal address counter.

F-25

- A0-A1
- Reset internal counter to last value loaded in control register via F-17;
 - Start transfert of both memories contents on ECLine bus. Data output synchronized either on external or internal clock depending upon option selected (F-26 function mode must be set).

Appendice B:
Listati dei programmi

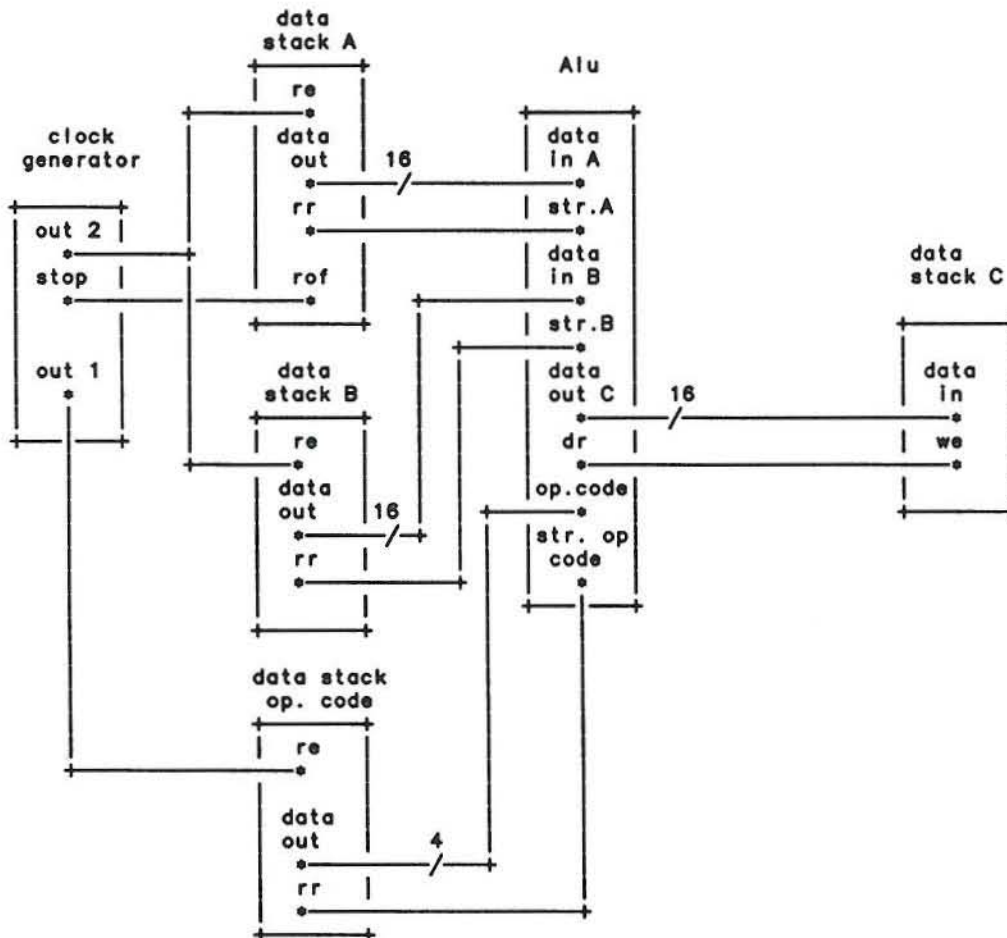
Appendice B1:
Documentazione dei programmi
di prova dei moduli

ALUTEST :

a program to test the Arithmetic and Logic Unit of Rome

P. Pacchiarotti

This program tests the ALU module developed in Rome. We need four datastacks, an ALU and a clock generator for making the test. These modules will be connected as shown below:

**Functionality principles:**

We fill the A and B data stacks of words (these words will be processed by the ALU) by the Camac dataway. In the op. code data stack we load the op. code, if we are using front panel op. code, otherwise we load the op. code via Camac function. After that the clock starts enabling the readout of the op. code stack and after 25ns the readout of A and B stacks (see specification of ALU for this delay). In this way the op. code is operative when arrive data at ALU. The result of alu is stored in the C data stack.

If we don't use the op. code strobe but only the data strobe (status register = 48) the op. code is operative on the second data arriving at the ALU; so we must skip away the first data because it has been computed by ALU with the old op. code.

All is cycled till the last dataword in A or in B is reached. The 'rof' signal generated by A stops the clock. After that the op. code is incremented by one and loaded, all is repeated with the new op. code till we reach the last op. code.

In this way we made four test :

- 1) O.K. test:
automatic test for ALU, we shall discuss it below.
- 2) Sequential test:

two words incremented by 1 are loaded in A and B stacks.
It's possible to choose the starting and ending number for the test.

- 3) Sampling test:
The A word is set to 0000 hex, the B word goes from 0000 hex till FFFF hex by step of 1111 hex. When B reaches FFFF hex A is set to 1111 hex, and so on till A = FFFF.
- 4) Filling one by one:
The user fills the stacks A and B and choose one op. code.

The tests are succesfull if the words in C are the right result of the operation between A and B selected by the current op. code.
All the errors of test 2) and 3) can be written on a file called alu_error.dat.

The ok test is similar to the bus switch one (see bustest.doc), it checks all the functions of Alu automatically giving an ok response if the test has been succesfull or reporting the wrong bits if the test has not been succesfull.

We make these checks:

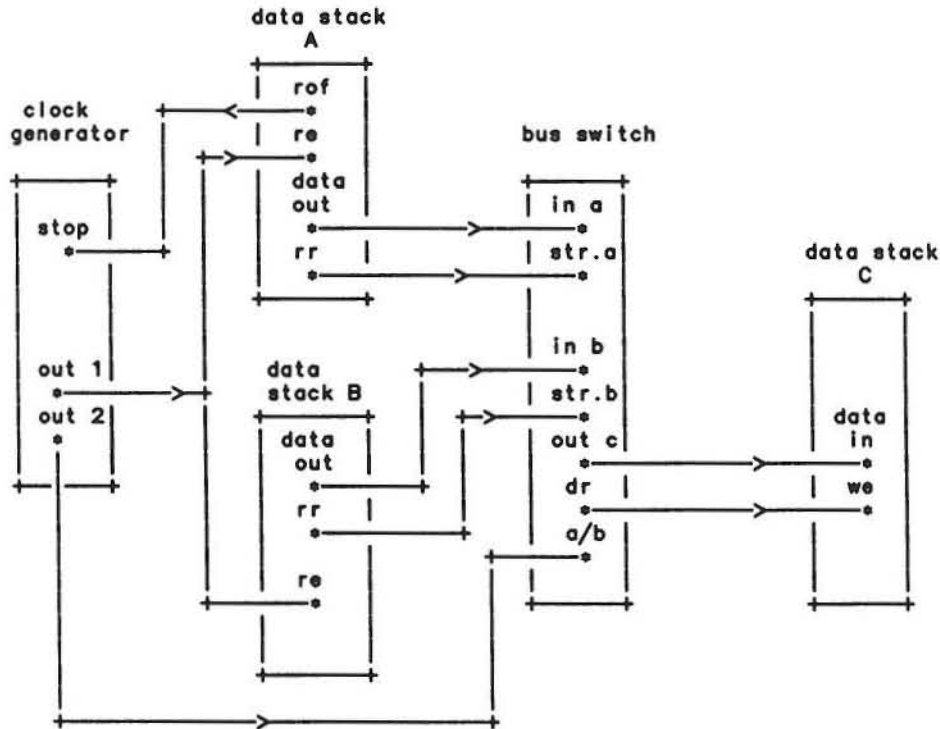
- 1) Cross talk test of ecl port A & B:
we write in A (B) via front panel and with op. code C = A (B)
we read via Camac the C result.
(for ref. on Cross talk see bustest.doc)
- 2) Cross talk test of ecl port C :
like point 1 but now reading C via front panel.
- 3) Sampling test of Camac op. code:
we test the op. code loading it by Camac function.
We load 256 words (as in sampling test) in A and B
and after the Alu processes these words 16 times one for each op. code.
- 4) Sampling test of front panel op. code:
like point 3 but now with front panel input of op. code
with its own strobe.
- 5) Sampling test of front panel op. code with data strobe:
like 4 but now the op code has the same strobe of data A & B.

BUSTEST:

a program to test the bus switch of ROME.

S. Gentile & P. Pacchiarotti

This program tests the bus switch developed in Rome, it's necessary an hardware wiring and several camac modules as show below:



N.B. bus switch jumper for data ready in 1 position

Functionality principles:

The two input datastacks A and B are filled of data by the camac dataway, then it's checked that filling has been successfull reading by camac the content of data stacks. After that the clock starts enabling the readout of the two stacks by front panel. When the data are ready they are written in A and B registers of bus switch. The data are ready (after a delay 20 ns) during all the clock period because the falling edge of 're' enables the readout. So if we use for 'a/b switch' the clock, delayed of 20 ns on out 2 it will switch between the two words (first B, after A). In the last datastack C we store the output of bus switch.

ex. If in A,B we write the following data :

in A: 0,1,2...
in B: 10,11,12...

we should read in C:

in C: 10,0,11,1,12,2...

All is cycled till the last data in A or in B is reached (it happens in same time). The 'rof' signal generated by A stops the clock. The minimum time for the clock is 110 ns, because for writing one word needs 50 ns (we write 2 words every clock cycle).

In this way the following tests are made:

- 1) Sequential test:
two words incremented by 1 are loaded in A & B stacks.
It's possible to choose the starting number in A and B.

- 2) Sampling test:
The A word is set to 0000 hex, the B word goes from 0000 hex till FFFF hex by step of 1111 hex. When B reaches FFFF hex A is set to 1111 hex, and so on till A = FFFF.
- 3) Random numbers test:
A random word (0-65535) is loaded in A and B.
- 4) Filling one by one:
The user fills the stacks.

The test is successful if the words read in output datastack C are in the sequence expected.
All the errors of the test 1) and 2) can be written on a file called error.dat.

The program provides also an automatic test (see o.k. test in the top menu). This test checks all the functions of the bus switch, we made three types of checks :

- 1) Writing and reading via Camac the A and B registers.
- 2) Writing via Ecl ports A,B and reading via Camac (tests if Ecl ports are all right).
- 3) Writing via Ecl ports and reading via the Ecl C port (tests if C is all right).

This test is made writing in both input datastacks A, B two particular sequences of words.

The first called 'Cross talk' is done in this way : we set all the bits to 1 (0) then we write this 16 bit word 000...01 (111...10) for going down (up) all the bits except for 1 (0). After we shift to the left the word and the procedure is repeated . All is cycled till the leftmost position for the 1 (0) is reached.

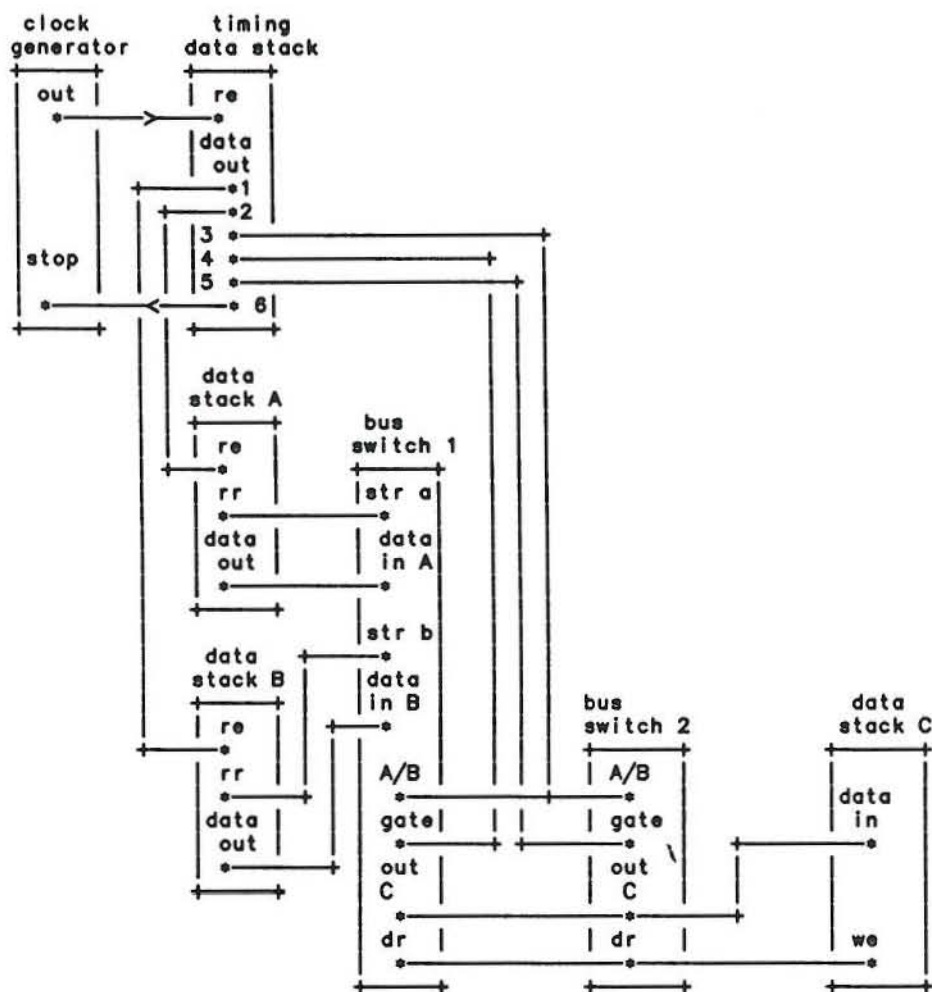
example:

```
word in A and B :      1111111111111111
                        0000000000000001
                        1111111111111111
                        0000000000000010
                        .
                        .
                        .
                        1111111111111111
                        1000000000000000
                        .
                        .
                        .
                        0000000000000000
                        1111111111111110
                        0000000000000000
                        1111111111111101
                        .
                        .
                        .
                        0000000000000000
                        0111111111111111
```

The second sequence is the same described before for Sampling test, it's called Sampling.

If, in one of the three checks, an error occurs is printed a list of the wrong bits. At the end of the ok test is displayed a table wich summarize all the results.

The o.k. test also provides to check the functionality of two bus switch in cascade (like in the 1 level trigger). For making this check it's necessary another bus switch and another data stack (for timing), and a new set of connections of the modules like show below.



N.B. jumper for data ready on 3 position, no pull down resistor on 'dr' and 'out C' in the bus switch 2.

The test is made writing into the two bus switch the following sequence of hex words (Sampling) (A1,B1 bus switch 1; A2,B2 bus switch 2) :

in :	A1	B1	A2	B2
	0000	0000	0000	0000
	0000	1111	1111	0000
	.	.	.	.
	0000	FFFF	FFFF	0000
	1111	0000	0000	1111
	1111	1111	1111	1111
	.	.	.	.
	FFFF	FFFF	FFFF	FFFF

The words are written: A1 B1 in the datastack, A2 and B2 in the second bus switch via Comac.

The timing is made by programming the fourth data stack with a sequence of 8 numbers and taking the first six bits of the data out port. The 8 numbers to load the stack are chosen to form a particular sequence of wave forms on each of the six bits.

This timing provides that the expected sequence in data stack C is :

B1 B2 A1 A2

TEST OF THE LE CROY LR2375 DATA STACK

P. Pacchiarotti

The program Datatest is a test for the Le Croy datastacks.
There is no need of hardware connections, every operation is done
via the CAMAC dataway.

The following tests are made:

- 1) Sequential test of writing and reading in datastack
(via camac).
One word is written in one location of datastack and then
it's incremented by 11 hex (17 dec) and written in the next location,
until the last location.
After the words are read, and it is checked that are the same that were
written.
A status message tells if occurred an error in writing or reading.
- 2) Testing with 101010... and 010101...
This test is the similar of sequential one, but this two particular
patterns of sixteen bit words are used.
The word 101010... is equal at aaaa hex and the word 010101... is
5555 hex .
First is checked the stack with 101010... after with 010101...

This program is useful for working with the datastacks because the user
with it can : read or write in every location, handle pointers and make
a MRST of the stack. Every operations is made by guided menus.

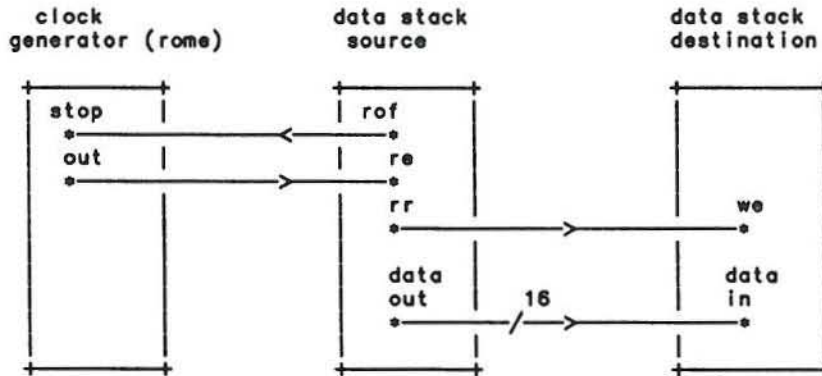
N.B. before all the operations the program must be initialize choosing the
A option of the Top menu.

TRANSTEST:

a program testing the transmission of data
between two datastacks via ECL port.

P. Pacchiarotti

This program tests the readout and readin via ECL ports of the
Le Croy 2375 datastacks. It's necessary an hardware wiring for the two
stacks that is specified below.



How does it work.

The data stack source (dss) is loaded via camac and it's checked that
the loading is been succesfull.
After this a 'start' command is given to clock generator that begins to
enable the dss to readout. When the data is ready the 'rr' signal
enables the data stack destination to be written in. This is cycled till
the last data in dss is transmitted, at this moment the 'rof' generated
by dss stops the clock.

The following tests are made:

- 1) Sequential test:
It's the same used in datatest (see it), but the word is checked in
the two datastacks.
- 2) Testing with 101010... and 010101...
Same as point 1), but with this two particular 16 binary words.

Options avaiible:

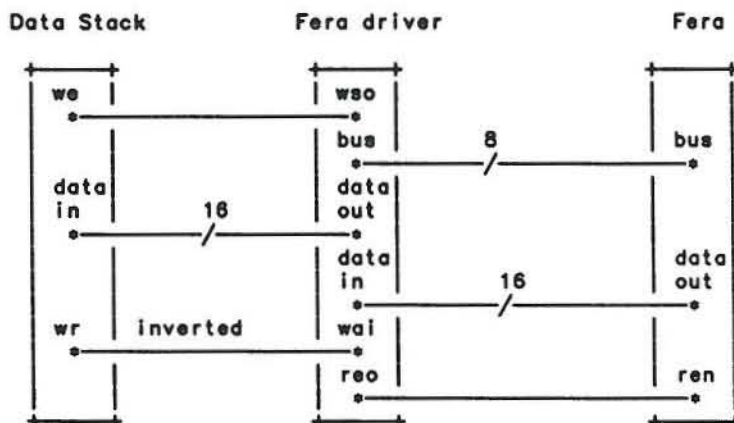
setting the no. of words in transmission.
setting clock period (min. 50 ns time of writing in stack).
setting the i/o in decimal or hexadecimal.

FERATEST:

a program to test Fera and Fera driver modules

P. Pacchiarotti

This program tests the le Croy 4300 Fera and the 4301 Fera driver, we need also a datastack for making the test, they will be connected as shown below :



The following tests are made :

1) Pedestal Test

The pedestals of each channel are measured, the test is not successful if

$$(\text{r.m.s. of pedestal}) > .5$$

where R.m.s. is the root mean square of the pedestal mean , it means that pedestals are too wide.

We also made the histograms of pedestals distribution with the hbook package.

2) Linearity test (linear fit)

We made a linear fit with $X = \text{trv input}$, $Y = \text{fera output}$. The straight line is : $Y = A * X + B$

The test is successful if the chi square of fit is less than 1 AND if the intercept is less than 2 .

3) Linearity tests:

A) Monotonicity test:

we compute the slope for each step of the trv input, the slope is : $m = (y_2 - y_1) / (x_2 - x_1)$

where Y is the adc output and X is the trv input.

It must be less than $2 / (x_2 - x_1)$ that is the error for m computed supposing an error of 1 on the Y readout and 0 on the X.

B) Integral linearity :

is defined as

$$\text{int. lin.} = (\text{out fera} - \text{trv input})$$

and must be less than $2 + 0.0025 * \text{out fera}$ in worst case.

The test is successful if the two linearities are in the given bounds. Because the slope of Adc is not 1 (ideal adc) we must first compute the slope and then multiply the trv input by it.

4) Ecl readout test

The output of fera driver loads a datastack then is compared the contents of datastack with the output of camac function.

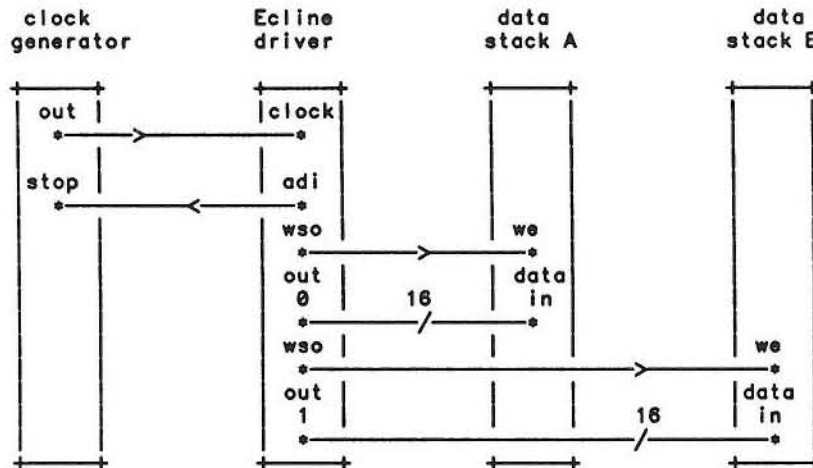
We made three tests :

- A) Test of only Ecl readout (status word = 4400 hex)
- B) Test of Ecl pedestal subtraction (status word = 4500 hex)
- C) Test of zero compression and pedestal subtraction (swr = 4700 hex)

The test is succesfull if the data in the stack are the expected ones, on choosing a particular option of fera output.

EclTest:
 a program to test the Ecline Driver
P. Pacchiarotti

This program is a test for the ecline driver developed at LAPP. We need for making the test two Le Croy Data stacks and a clock generator of INFN-Rome. The modules will be connected as shown below:



With these connections we make the following tests:

- 1) Test of programming Ecline driver:
 The Ecline driver is loaded with two particular binary words (101010... and 010101... one bit up one down) by Camac function then it's checked always by camac function that the loading has been successfull. If an error occurs is displayed at what address it happened.
- 2) Test of the Ecl readout :
 The Ecline driver is loaded always by camac function (both memory banks are loaded). Then, connected the memories to the ecl port, we start the external clock wich enables the readout of Ecline driver on the two Data stacks A and B. We check in the two data stacks that the readout has been succesfull. We make this test with three kind of binary words:
 - A) 101010... and 010101...
 - B) Cross Talk sequence (as in the other tests)
 - C) words in sequence by step of 256

the program also provides some utilities for working with ecline driver as:

- 1) write and read data in a choosed location and bank
- 2) Write and read the control register

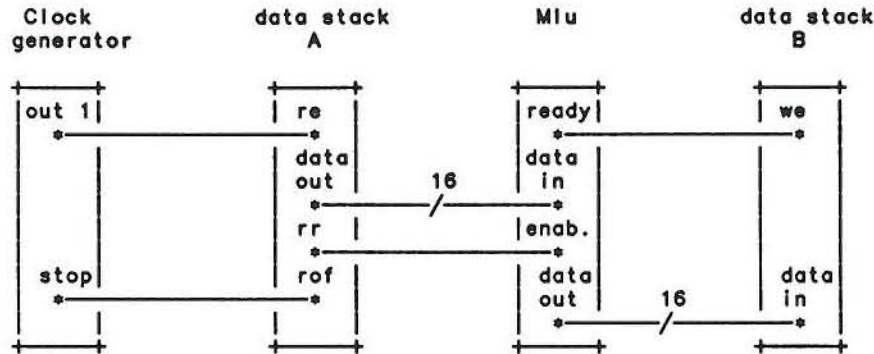
MLUTEST:

a program to test the Memory Lookup Unit

P. Pacchiarotti

This program tests the Le Croy 2372 module (MLU). We need also two Le Croy Data stacks and a Clock generator of INFN-Rome. They will be connected as shown below:

Wiring:



In this way we made two tests:

- 1) Test of programming memory :
the Mlu memory is loaded via camac with two particular binary words (101010...10, 010101...01), then it's checked, reading via camac, that the loading has been successfull. If an error occurs it's printed what is the wrong address. This test can be made choosing one particular dimensionality option.
- 2) Test of Ecl ports readout/in :
It's similar to point one but now, after programming Mlu, the address is loaded via front panel using a data stack (A) output, and the output word of Mlu is loaded in another data stack (b) always via front panel. This test can be made choosing the Mode option of Mlu (strobed or transparent).

The program provides also some utilities for working with an Mlu such as :

- 1) Write and read data in a choosed location.
- 2) Write and read CCr (Camac Control Register).

Appendice B2:
Programmi di prova
dei moduli

```

*****
Bigtest:
      a program to test
      every module of L3
      energy trigger

      by
      F. Cesaroni, S. Gentile,
      P. Pacchiarotti
      jul-86
*****
PROGRAM BIGTEST

implicit integer (a-z)
common/mncomm/nwrds,mnbuf(500)

data nwrds/500/

c
c
c
10      top level Menu

      continue
      call mnclr
      call mnhead('***** Test of Energy Trigger modules *****')
      call mnopt(1,'Help')
      call mnopt(2,'Test of Alu module')
      call mnopt(3,'Test of Bus Switch module')
      call mnopt(4,'Test of Data Stack module')
      call mnopt(5,'Test of Mlu module')
      call mnopt(6,'Test of Fera+Fera driver')
      call mnopt(7,'Test of Ecline driver')
      call mnopt(8,'Exit from program')
      call mndisp
      call mnin(opt_no)
      goto (10,10,10,50,100,200,300,400,500,600,700) opt_no+3

c
c
c
50      Help

      call mneras
      print*,'Help:'
      print*,' '
      print*,'This program is a general test for the modules used'
      print*,'in the 1st level energy trigger of L3 experiment. '
      print*,' '
      print*,'Every test is explained in a file called Test.doc, '
      print*,'there is also specified how many modules are necessary'
      print*,'for making the test and the cabling between the modules.'
      print*,'Good Work.'
      print*,'          L3 Rome'
      print*,' '
      call dikey('Press any key to return to menu')
      goto 10

c
c
c
100     Test of Alu

      continue
      call alutest
      goto 10

c
c
c
200     Test of bus switch

      continue
      call bustest
      goto 10

c
c
c
300     Test of Data stack

      continue
      call mnclr
      call mnhead('***** Test of Data Stack module *****')
      call mnopt(1,'Test of one Data Stack')
      call mnopt(2,'Test of Ecl port Readout/in of two Data Stacks')
      call mndisp
      call mnin(opt_no)
      goto (10,10,300,310,320)opt_no+3

310     call datatest

```

```

        goto 10
320    call transtest
        goto 10
c
c      Test of Mlu
c
400    continue
        call mlutest
        goto 10
c
c      test of fera
c
500    continue
        call feratest
        goto 10
c
c      Test of ecline driver
c
600    continue
        call ecltest
        goto 10
c
c      exit
c
700    continue
        end

```

```

*****
*
*      ALUTEST :
*              program to test the Alu module
*              of Rome
*
*              by Paolo Pacchiarotti
*
*****

```

Subroutine ALUTEST

```

implicit integer(a-z)
integer bin_word(2),vet_op_code(0:1),zero(2)

logical q,ecl_a,ecl_b,ecl_c,ok_test
logical error0(0:15),error1(0:15),error_op_code(0:15)

character*80 form

common/mncomm/nwrds,mnbuf(500)
common / oppars/b,c,alu,dsa,dsb,dsc,dsop,ck,period,num_wo,
+       num_te,video_print,camac_op,hex,data_strobe,
+       all,delay,delay1,ck1
common/register/alu_a_0,alu_a_1,dsa_a_0,dsa_a_1,dsa_a_2,dsb_a_0,
+       dsb_a_1,dsb_a_2,dsc_a_0,dsc_a_1,ck_a_0,ck_a_1,
+       dsop_a_0
common/formats/form,ok_test,error_op_code
common / bits /error0,error1

data nwrds / 500 /

c
c      Reads parameters from oppars
c
        call parfil('alutest.las')
        call parrd

c
c      top level menu
c
10     continue
        ok_test = .false.
        call mnclr
        call mnhead('***** Test of the ALU module : top menu *****')
        call mnopt(1,'O.K. test')
        call mnopt(2,'Sequential test')
        call mnopt(3,'Sampling test')
        call mnopt(4,'Filling one by one')
        call mnonof(5,'Initialize and check the set up',onof)

```



```

call mndisp
call mnin(opt_no)
goto (999,10,10,100,200,300,400,500) opt_no + 3

c
c      o.k. test : menu
c
100  continue

c
c      some initializations
c
ok_test = .true.
bin_word(1) = 1
bin_word(2) = 65534
zero(1) = 65535
zero(2) = 0
do i=0,15
  error0(i) = .false.
  error1(i) = .false.
enddo

call mncir
call mnhead('**** O.K. test : automatic test for the Alu ****')
call mnopt(1,'Cross talk test of Ecl port A and B ')
call mnopt(2,'Cross talk test of Ecl port C ')
call mnopt(3,'Test of op. code via Camac')
call mnopt(4,'Test of op. code external, with op code strobe')
call mnopt(5,'Test of op. code external, with data strobe')
call mnonof(6,' a,b,c,d,e tests all together',alltest)
call mndisp
call mnin(opt_no)
goto (999,10,100,1001,2000,3000,4000,5000,100) opt_no + 3

c
c      test of Ecl port A,B : writing by front panel
c      and reading C (op.code C=A or C=B) via camac
c
1001 continue
call cfsa(26,alu_a_0,dummy,q) ! enables camac function
call mneras
print *, '**** Test of Ecl port A,B ****'
print *, '**** Writing A,B via front panel and reading ****'
print *, '**** C with op.code C = A or C = B ****'
print *, '**** CROSS TALK TEST ****'
print *, ' '
ecl_a = .true.
ecl_b = .true.
print *, ' Now testing Ecl port A and B '
vet_op_code(0) = 8 ! c = b
vet_op_code(1) = 9 ! C = A
do k=0, 1
  call cfsa(16,alu_a_0,vet_op_code(k),q)
  do j=1,2
    in_word = bin_word(j)
    do i=0,15
      call cfsa(9,dsa_a_0,dummy,q) ! mrst of A
      call cfsa(9,dsb_a_0,dummy,q) ! mrst of B
      call cfsa(16,dsa_a_0,zero(j),q)
      call cfsa(16,dsa_a_0,in_word,q)
      call cfsa(16,dsb_a_0,zero(j),q)
      call cfsa(16,dsb_a_0,in_word,q)
      call cfsa(16,dsa_a_1,0,q)
      call cfsa(16,dsb_a_1,0,q)
      call cfsa(25,ck_a_0,dummy,q) ! two shot of clock
      call cfsa(25,ck_a_0,dummy,q)
      call cfsa(0,alu_a_1,out_word,q)
      if (k.eq.0) then
        call testbit(out_word,in_word,ecl_b,error1)
      else
        call testbit(out_word,in_word,ecl_a,error0)
      endif
      in_word = jshftc(in_word,1,16) ! shifts word
    enddo
  enddo
enddo
call print_head
call ecl_resp(ecl_a,ecl_b)
if (alltest.eq.0) goto 100

```

```

c
c      Test of Ecl port C : writing A,B via Ecl
c      reading C via Ecl port
c
2000  continue
      do i=0,15
        error0(i) = .false.
        error1(i) = .false.
      enddo
      call cfsa(26,alu_a_0,dummy,q) ! enables camac function
      call mneras
      print *, '****      Test of Ecl port C      ****'
      print *, '**** Writing A,B via front panel and reading ****'
      print *, '**** C via Ecl port with op.code C = A or C = B**'
      print *, '****      CROSS TALK TEST      ****'
      print *, ' '
      ecl_c = .true.
      print *, ' Now testing Ecl port C '
      vet_op_code(0) = 8   ! c = b
      vet_op_code(1) = 9   ! C = A
      do k=0, 1
        call cfsa(16,alu_a_0,vet_op_code(k),q)
        do j=1,2
          in_word = bin_word(j)
          do l=0,15
            call cfsa(9,dsa_a_0,dummy,q) ! mrst of A
            call cfsa(9,dsb_a_0,dummy,q) ! mrst of B
            call cfsa(9,dsc_a_0,dummy,q) ! mrst of C
            call cfsa(16,dsa_a_0,zero(j),q)
            call cfsa(16,dsa_a_0,in_word,q)
            call cfsa(16,dsb_a_0,zero(j),q)
            call cfsa(16,dsb_a_0,in_word,q)
            call cfsa(16,dsa_a_1,0,q)
            call cfsa(16,dsb_a_1,0,q)
            call cfsa(26,ck_a_0,dummy,q) ! start
            call cfsa(16,dsc_a_1,1,q)   ! rp of C at 1
            call cfsa(0,dsc_a_0,out_word,q) ! read C
            if (k.eq.0) then
              call testbit(out_word,in_word,ecl_c,error1)
            else
              call testbit(out_word,in_word,ecl_c,error0)
            endif
            in_word = jshftc(in_word,1,16) ! shifts word
          enddo
        enddo
      enddo
      call print_head
      call ecl_c_resp(ecl_c)
      if (alltest.eq.0) goto 100

c
c
c      Test of op code via camac
c
3000  continue
      call mneras
      print *, '**** Test of op. code via camac ****'
      print *, '**** Writing op. code via camac ****'
      print *, ' '
      print *, ' Now testing Camac Op. code'
      call test_op_code(1)
      call dkey('Press any key to continue')
      if (alltest.eq.0) goto 100

c
c
c
c      Test of front panel op code with op code strobe
c
4000  continue
      call mneras
      print *, '**** Test of front panel op. code with ****'
      print *, '****      op code strobe      ****'
      print *, ' '
      print *, ' Now testing front panel op code '
      call test_op_code(2)
      call dkey('Press any key to continue')
      if (alltest.eq.0) goto 100

c
c
c
c      Test of front panel op. code with data strobe
c
5000  continue

```

```

call mneras
print*, '**** Test of front panel op. code with      ****'
print*, '**** data strobe (status reg. = 48)      ****'
print*, ' '
print *, ' Now testing front panel op code with data strobe '
call test_op_code(3)
call dikey('Press any key to continue')
goto 100

c
c sequential test
c
200 continue

if (hex.eq.1) then
+ call dihex('Enter starting word (hex) for sequential test',
+ st_word,st_word)
+ call dihex('Enter ending word (hex) for sequential test',
+ end_word,end_word)
else
+ call dldc('Enter starting word (dec) for sequential test ',
+ st_word,st_word)
+ call dldc('Enter ending word (dec) for sequential test',
+ end_word,end_word)
endif

c
c loop for number of test
c
do i=1, num_te
write(unit,*) 'Sequential Test. test no.', i

c
c main loop for sequential test
c
do word = st_word, end_word, num_wo
call cfsa(9, dsa_a_0, dummy, q) ! reset of datastacks
call cfsa(9, dsb_a_0, dummy, q) ! reset of datastacks
start_point = 0
if (data_strobe .eq. 1) then
call cfsa(16, dsa_a_0, dummy, q) ! writes two dummy data in the
call cfsa(16, dsb_a_0, dummy, q) ! stack when op. is strobed by
start_point = 1 ! data strobe
endif

c
c loop for writing
c
do in_word = word, word + num_wo - 1
call cfsa(16, dsa_a_0, in_word, q) ! writes in A
call response(dsa, q, dummy)
call cfsa(16, dsb_a_0, in_word, q) ! writes in B
call response(dsb, q, dummy)
enddo

c
c op code loop
c
do op_code = 0, 15
call write_op_code(op_code)
call cfsa(9, dsc_a_0, dummy, q) ! reset of datastack C
call cfsa(9, dsa_a_0, dummy, q) ! of A
call cfsa(9, dsb_a_0, dummy, q) ! of B
call cfsa(16, dsa_a_2, num_wo+start_point, q) ! reset Wp of A
call cfsa(16, dsb_a_2, num_wo+start_point, q) ! reset wp of B
if (data_strobe.eq.1) then
call cfsa(25, ck_a_0, dummy, q) ! shifts the frst word
endif
call cfsa(0, alu_a_1, res_word, q) ! reads the last result
call cfsa(26, ck_a_0, dummy, q) ! start of clock
call cfsa(16, dsa_a_1, start_point, q) ! reset rp of A
call cfsa(16, dsb_a_1, start_point, q) ! reset rp of B
call cfsa(16, dsc_a_1, start_point, q) ! reset rp of C
write(*,*) '***** Operation code is:', op_code
do pointer = start_point, start_point + num_wo - 1
call cfsa(0, dsa_a_0, out_word_a, q)
call response(dsa, q, pointer)
call cfsa(0, dsb_a_0, out_word_b, q)
call response(dsb, q, pointer)
call cfsa(0, dsc_a_0, out_word_c, q)
call compute(out_word_a, out_word_b, out_word_c, res_word
+ , op_code)
enddo

```



```

        if (video_print.eq.1) call dlkey
    enddo ! op code loop
enddo ! main loop
enddo ! test loop

    goto 10

c
c    Sampling test
c
300    continue

c
c    loop for number of test
c
do i=1, num_te
    write(unit,*) 'Sampling test . Test no.', i
    call cfsa(9,dsa_a_0,dummy,q) ! reset of datastacks
    call cfsa(9,dsb_a_0,dummy,q) ! reset of datastacks
    star_point = 0
    if (data_strobe .eq. 1) then
        call cfsa(16,dsa_a_0,dummy,q) ! writes two dummy data in the
        call cfsa(16,dsb_a_0,dummy,q) ! stack when op. is strobed by
        star_point = 1
    endif

c
c    loop for writing
c
word_a = 0
count = 0
do j=0,15
    word_b = 0
    do k=0,15
        call cfsa(16,dsa_a_0,word_a,q) ! writes in A
        call response(dsa,q,count)
        call cfsa(16,dsb_a_0,word_b,q) ! writes in B
        call response(dsb,q,count)
        call incr(word_b,word_b)
        count = count + 1
        if (num_wo.eq.count) goto 1000
    enddo
    call incr(word_a,word_a)
enddo

1000    continue

c
c    op code loop
c
do op_code = 0,15
    call write_op_code(op_code)
    call cfsa(9,dsc_a_0,dummy,q) ! reset of datastack C
    call cfsa(9,dsa_a_0,dummy,q) ! of A
    call cfsa(9,dsb_a_0,dummy,q) ! of B
    call cfsa(16,dsa_a_2,num_wo+start_point,q) ! reset Wp of A
    call cfsa(16,dsb_a_2,num_wo+start_point,q) ! reset wp of B
    if (data_strobe.eq.1) then
        call cfsa(25,ck_a_0,dummy,q) ! shifts the first word
    endif
    call cfsa(0,alu_a_1,res_word,q) ! reads the last result
    call cfsa(26,ck_a_0,dummy,q) ! start of clock
    call cfsa(16,dsa_a_1,start_point,q) ! reset rp of A
    call cfsa(16,dsb_a_1,start_point,q) ! reset rp of B
    call cfsa(16,dsc_a_1,start_point,q) ! reset rp of C
    write(*,*) '***** Sampling test *****'
    write(*,*) '***** Operation code is:', op_code
    do pointer = start_point, start_point + num_wo - 1
        call cfsa(0,dsa_a_0,out_word_a,q)
        call response(dsa,q,pointer)
        call cfsa(0,dsb_a_0,out_word_b,q)
        call response(dsb,q,pointer)
        call cfsa(0,dsc_a_0,out_word_c,q)
        call response(dsc,q,pointer)
        call compute(out_word_a,out_word_b,out_word_c,res_word
+
,op_code)
    enddo
    if (video_print.eq.1) call dlkey
enddo ! op code loop
enddo ! test loop

```



```

goto 10

c
c
c
400
continue
call mneras
call dlout('***** Filling one by one *****')
call dldec('Enter no. of words',num_wo,num_wo)
call cfsa(9,dsa_a_0,dummy,q) ! reset of datastacks
call cfsa(9,dsb_a_0,dummy,q) ! reset of datastacks
star_point = 0
if (data_strobe.eq.1) then
    call cfsa(16,dsa_a_0,dummy,q) ! writes two dummy data in the
    call cfsa(16,dsb_a_0,dummy,q) ! stack when op. is strobed by
    start_point = 1 ! data strobe
endif
do j=1,num_wo
    write(*,*)'Words no. ',j
    if (hex.eq.1) then
        write(*, '(1x,a22,$)')'enter word A,B (hex): '
        read (*, '(2z5)')word_a,word_b
    else
        write(*, '(1x,a22,$)')'enter word A,B (dec): '
        read (*, '(2i5)')word_a,word_b
    endif
    call cfsa(16,dsa_a_0,word_a,q) ! writes in A
    call response(dsa,q,dummy)
    call cfsa(16,dsb_a_0,word_b,q) ! writes in B
    call response(dsb,q,dummy)
enddo
call dldec('enter the op. code (dec)',op_code,op_code)
call write_op_code(op_code)
call cfsa(9,dsc_a_0,dummy,q) ! reset of datastack C
call cfsa(9,dsa_a_0,dummy,q) ! of A
call cfsa(9,dsb_a_0,dummy,q) ! of B
call cfsa(16,dsa_a_2,num_wo+start_point,q) ! reset Wp of A
call cfsa(16,dsb_a_2,num_wo+start_point,q) ! reset wp of B

if (data_strobe.eq.1) then
    call cfsa(25,ck_a_0,dummy,q) ! shifts the first word
endif
call cfsa(0,alu_a_1,res_word,q) ! reads the last result
call dlkey('Press any key to start the clock ')
call cfsa(26,ck_a_0,dummy,q) ! start of clock
call cfsa(16,dsa_a_1,start_point,q) ! reset rp of A
call cfsa(16,dsb_a_1,start_point,q) ! reset rp of B
call cfsa(16,dsc_a_1,start_point,q) ! reset rp of C
write(unit,*)'***** Filling one by one *****'
write(unit,*)'***** Operation code is:',op_code
do pointer=star_point, start_point + num_wo - 1
    call cfsa(0,dsa_a_0,out_word_a,q)
    call response(dsa,q,pointer)
    call cfsa(0,dsb_a_0,out_word_b,q)
    call response(dsb,q,pointer)
    call cfsa(0,dsc_a_0,out_word_c,q)
    call compute(out_word_a,out_word_b,out_word_c,res_word
    ,op_code)
enddo
if (video_print.eq.1)call dlkey

goto 10

c
c
c
500
continue
call mnclr
call mnhead('Initialize the set up : menu')
call mndec(1,'Branch',b,b)
call mndec(2,'Crate',c,c)
call mndec(3,'Alu in',alu,alu)
call mndec(4,'Data stack A in',dsa,dsa)
call mndec(5,'Data stack B in',dsb,dsb)
call mndec(6,'Data stack C in',dsc,dsc)
call mndec(7,'Data stack for op. code',dsop,dsop)
call mndec(8,'Clock generator in',ck,ck)
call mnnopt(9,' ')
call mndec(10,'Clock period (40-190 ns)',period,period)
call mndec(11,'No. of words in memory (1-255)',num_wo,num_wo)

```



```

c                                     in d.s. (camac_op = 0)
c
SUBROUTINE write_op_code(op_code)

implicit integer (a-z)
integer op_code,camac_op
logical q

+ common / oppars/b,c,alu,dsa,dsb,dsc,dsop,ck,period,num_wo,
+   num_te,video_print,camac_op,hex,data_strobe

+ common/register/alu_a_0,alu_a_1,dsa_a_0,dsa_a_1,dsa_a_2,dsb_a_0,
+   dsb_a_1,dsb_a_2,dsc_a_0,dsc_a_1,ck_a_0,ck_a_1,
+   dsop_a_0

if (camac_op.eq.1) then
  call cfsa(9,dsop_a_0,dummy,q) | reset opcode stack
  call cfsa(26,alu_a_0,dummy,q) | enables camac function
  call cfsa(16,alu_a_0,op_code,q) | writes the op. code and disable
                                | front panel op. code
else
  if (data_strobe.eq.0) then
    call cfsa(16,alu_a_0,16,q) | enables front panel op.code
    call cfsa(9,dsop_a_0,dummy,q) | reset of datastacks
    do i=1, num_wo
      call cfsa(16,dsop_a_0,op_code,q) | writes op_code in d.s.
      call response(dsop,q,dummy)
    enddo
  else
    call cfsa(16,alu_a_0,48,q) | enables data strobe
    call cfsa(9,dsop_a_0,dummy,q) | reset of datastacks
    do i=1,num_wo
      call cfsa(16,dsop_a_0,op_code,q) | writes op_code in d.s.
      call response(dsop,q,dummy)
    enddo
  endif
endif

return
end

c
c
c Subroutine compute: compute the operation between A and B
c and prints the result
c
SUBROUTINE COMPUTE(word_a,word_b,word_c,res_word,op_code)

implicit integer(a-z)
character*8 status
character*80 form
logical ok_test,error_op_code(0:15)
+ common / oppars/b,c,alu,dsa,dsb,dsc,dsop,ck,period,num_wo,
+   num_te,video_print,camac_op,hex,data_strobe
+ common/formats/form,ok_test,error_op_code

goto (20,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15) op_code + 1

c
c compute word for each op code
c
20 res_word = res_word + word_b
if (res_word.ge.65536) res_word = res_word - 65536
goto 100
1 res_word = word_a + word_b
if (res_word.ge.65536) res_word = res_word - 65536
goto 100
2 res_word = word_a + word_b
if (res_word.ge.65536) res_word = res_word - 65536
goto 100
3 res_word = word_a - word_b
if (res_word.lt.0)res_word= 65536 + res_word
goto 100
4 res_word = word_a - word_b
if (res_word.lt.0)res_word= 65536 + res_word
goto 100
5 res_word = 2 * word_a
if (res_word.ge.65536) res_word = res_word - 65536
goto 100
6 res_word = word_a - 1

```



```

        if (res_word.lt.0)res_word= 65536 + res_word
        goto 100
7      res_word =res_word - word_b
        if (res_word.lt.0)res_word= 65536 + res_word
        goto 100
8      res_word = word_b
        goto 100
9      res_word = word_a
        goto 100
10     res_word = 0
        goto 100
11     res_word = 65535
        goto 100
12     res_word = land(word_a,word_b)
        goto 100
13     res_word = land(word_a,word_b)
        goto 100
14     res_word = lor(word_a,word_b)
        goto 100
15     res_word = leor(word_a,word_b)
        goto 100

100    if (res_word.ne.word_c) then
        status = 'error'
      else
        status = 'o.k.'
      endif
      if (video_print.eq.1) then
        unit = 5
      else
        unit = 10
      endif

      if (.not.ok_test) then
        if ((unit.eq.10.and.status.eq.'error').or.(unit.eq.5)) then
          write(unit,form)'Op.code: ',op_code,' Word: in A:',word_a,
+ ' In B:',word_b,' Check:',res_word,' Result on C:',word_c,
+ ' Status: ',status
        endif
      else
        if (status.eq.'error') then
          error_op_code(op_code) = .true.
        endif
      endif
      return

      end

      subroutine incr (jjin,jjout)
C.....
C      Incrementa una parola a 16 bits 4
C      alla volta cosi:
C          0000
C          1111
C          2222
C          ....
C          FFFF
C.....

      implicit integer (a-z)

      jjout = jjin + 4369
      return
      end

C
C      Subroutine testbit
C
C      test two words bit by bit

      subroutine testbit(word1,word2,equal,errors)

      integer*4 word1,word2
      logical equal,errors(0:15)

      do i=0,15
        if (jibits(word1,i,1).ne.jibits(word2,i,1)) then
          equal = .false.
          errors(i) = .true.
        endif
      enddo

```



```

character*80 form

logical ok_test,error_op_code(0:15)

common/formats/form,ok_test,error_op_code
common / oppars/b,c,alu,dsa,dsb,dsc,dsop,ck,period,num_wo,
+       num_te,video_print,camac_op,hex,data_strobe
common/register/alu_a_0,alu_a_1,dsa_a_0,dsa_a_1,dsa_a_2,dsb_a_0,
+       dsb_a_1,dsb_a_2,dsc_a_0,dsc_a_1,ck_a_0,ck_a_1,
+       dsop_a_0

do i=0,15
    error_op_code(i) = .false.
enddo
call cfsa(9,dsa_a_0,dummy,q) ! reset of datastacks
call cfsa(9,dsb_a_0,dummy,q) ! reset of datastacks

c
c      loop for writing
c
count = 0
word_a = 0
do j=0,15
    word_b = 0
    do k=0,15
        call cfsa(16,dsa_a_0,word_a,q) ! writes in A
        call response(dsa,q,count)
        call cfsa(16,dsb_a_0,word_b,q) ! writes in B
        call response(dsb,q,count)
        count = count + 1
        call incr(word_b,word_b)
    enddo
    call incr(word_a,word_a)
enddo

c
c      op code loop
c
do op_code = 0,15
    if (num_test.eq.1) then
        camac_op = 1
        call write_op_code(op_code) ! CaMAC op code
    endif
    if (num_test.eq.2) then
        camac_op = 0
        data_strobe = 0
        call write_op_code(op_code) ! extern op code
    endif
    if (num_test.eq.3) then
        camac_op = 0
        data_strobe = 1
        call write_op_code(op_code) ! op. with data strobe
    endif
    start_pointer = 0
    call cfsa(9,dsc_a_0,dummy,q) ! reset of datastack C
    call cfsa(9,dsa_a_0,dummy,q) ! of A
    call cfsa(9,dsb_a_0,dummy,q) ! of B
    call cfsa(16,dsa_a_2,255,q) ! reset Wp of A
    call cfsa(16,dsb_a_2,255,q) ! reset wp of B
    if (data_strobe .eq.1) then
        start_pointer = 1 ! shifts one words if op.code is
        call cfsa(25,ck_a_0,dummy,q) ! strobed by data strobe
    endif

    call cfsa(0,alu_a_1,res_word,q) ! reads the last result
    call cfsa(26,ck_a_0,dummy,q) ! start of clock
    call cfsa(16,dsa_a_1,start_pointer,q) ! reset rp of A
    call cfsa(16,dsb_a_1,start_pointer,q) ! reset rp of B
    call cfsa(16,dsc_a_1,start_pointer,q) ! reset rp of C
    do pointer = start_pointer, 255
        call cfsa(0,dsa_a_0,out_word_a,q)
        call response(dsa,q,pointer)
        call cfsa(0,dsb_a_0,out_word_b,q)
        call response(dsb,q,pointer)
        call cfsa(0,dsc_a_0,out_word_c,q)
        call compute(out_word_a,out_word_b,out_word_c,res_word
+       ,op_code)
    enddo
enddo ! op code loop

```

```

do i=0, 15
  if (error_op_code(i)) then
    write(*,*)'Error on op. code no.',i
  else
    write(*,*)'OP. CODE no.',i,' it's ok!'
  endif
enddo

return
end

C      TESTBUS
C
C      PROGRAM TO TEST THE BUS SWITCH
C
C      S. Gentile
C
C      20/2/86
C
C      Experimental set-up consist of:
C      1. clock generator      N=10
C      2. data stack in(A)     N=8
C      3. data stack in(B)     N=12
C      4. Bus switch module    N=21
C      5. data stack out       N=6
C
C
C      subroutine bustest
C      IMPLICIT INTEGER (A-Z)
C      REAL RAN,SECNDS
C      character*13 a13
C      character*8 a1,a2,a3,a5,a4
C      CHARACTER*8,STATUS(2)/' O.K. ',' ERROR'/
C      INTEGER*2 INT2(2)
C      EQUIVALENCE (INT2(1),KRESULT)
C      integer*4 bin_word(2),zero(2),bus(0:1)
C      logical camac(0:1,2),ecl_c(2),ecl_a(2),ecl_b(2),q
C      logical error0(0:15),error1(0:15),error2(0:15),error3(0:15)
C      + ,error4(0:15),error5(0:15)
C
C      DIMENSION DA(300),DB(300),DC(300),RA(300),RB(300),
C      + RER(300),STOP(300)
C      common / bits /error0,error1,error2,error3,error4,error5
C      COMMON/MNCOMM/NWRDS,MNBUF(1000)
C      COMMON/OPPARS/PERIOD,KTEST,NTRAS,LOOP,IPW,B,C,SCLK,SMEM1,SMEM2
C      + ,SBUS,SDST,bus2,ds,ck1,delay,delay1
C
C
C      DATA NWRDS/1000/
C      DATA PERIOD,KTEST,NTRAS,LOOP,JRESO,ERRCO,IPW
C      +/80,1,1,5,0,1,0/
C
C      Inizializzazione Camac da spostare
C      nel main
C
C      CALL PARFIL('bustest.LAS')
C      CALL PARRD
C
C      CALL MNBUFF(MNBUF,1000)
C
C      1 CONTINUE
C      YN=0
C      CALL MNCLR
C      CALL MNHEAD('Top menu','Program to test the Bus-switch')
C      CALL MNOPT (1,'Initialize the set-up')
C      call mnopt (2,'O.K. Test')
C      CALL MNOPT (3,'Sequential test')
C      CALL MNOPT (4,'Sampling test')
C      CALL MNOPT (5,'Random numbers')
C      CALL MNOPT (6,'Filling one-by-one')
C      CALL MNOpt (7,'Check of B,C,N')
C      CALL MNOPT (8,'STOP')
C      CALL MNDISP
C      CALL MNIN(IOPT)
C      IOPT = IOPT + 3

```

```

GO TO (999,1,1,2,70,20,30,40,50,19,999),IOPT

C
C
C
C
19 CONTINUE
YN = 1
C
CALL MNCLR
CALL MNHEAD('Bus switch test -set up configuration')
CALL MNDEC (1,'Branch',B,B)
CALL MNDEC (2,'Crate ',C,C)
CALL MNDEC (3,'Clock',SCLK,SCLK)
CALL MNDEC (4,'Data stack input 1',SMEM1,SMEM1)
CALL MNDEC (5,'Data stack input 2',SMEM2,SMEM2)
CALL MNDEC (6,'Data stack out',SDST,SDST)
CALL MNDEC (7,'Bus switch',SBUS,SBUS)
call mndec (8,'Second bus switch',bus2,bus2)
call mndec (9,'Timing data stack (only for two bus switch)',ds,ds)
call mnnopt (10,'Set up only for TAU1-2 measurement')
call mndec (11,'Clock Generator 2 in',ck1,ck1)
call mndec (12,'Delay on out 3 clock 1 (o.s. clock2)',delay,delay)
call mndec (13,'Delay on out 1 clock 2 and on out 2 + half period'
+ ,delay1,delay1)

C
C
CALL MNDISP
CALL MNIN(MOPT)
MOPT = MOPT +3
GO TO (10,10,19,19,19,19,19,19,19,19,19,19,19,19,19,19),MOPT

C
10 CONTINUE
CALL INIZ_CAMAC(b,c)

CALL CDREG( CLOCK0 , B, C, SCLK, 0)
CALL CDREG( CLOCK1 , B, C, SCLK, 1)
CALL CDREG( DS1A0 , B, C, SMEM1, 0)
CALL CDREG( DS1A1 , B, C, SMEM1, 1)
CALL CDREG( DS1A2 , B, C, SMEM1, 2)
CALL CDREG( DS2A0 , B, C, SMEM2, 0)
CALL CDREG( DS2A1 , B, C, SMEM2, 1)
CALL CDREG( DS2A2 , B, C, SMEM2, 2)
CALL CDREG( DS3A0 , B, C, SDST, 0)
CALL CDREG( DS3A1 , B, C, SDST, 1)
CALL CDREG( DS3A2 , B, C, SDST, 2)
call cdreg( ds4a0 , b, c, DS, 0)
call cdreg( ds4a1 , b, c, DS, 1)
call cdreg( ds4a2 , b, c, DS, 2)
CALL CDREG( BUS0 , B, C, SBUS, 0)
CALL CDREG( BUS1 , B, C, SBUS, 1)
CALL CDREG( BUS2_a_0 , B, C, BUS2, 0)
CALL CDREG( BUS2_a_1 , B, C, BUS2, 1)

C
C
C
C
CALL CFSA(24,CLOCK0,0,Q) I STOP CLOCK
call inizeclock(b,c,scik,period,25,60,delay,25,0) I set delay on out 2
call inizeclock(b,c,ck1,40,delay1,delay1+period/2,25,25,0)
CALL CFSA(9,DS1A0,0,Q) I RESET DS1
CALL CFSA(9,DS2A0,0,Q) I RESET DS2
CALL CFSA(9,DS3A0,0,Q) I RESET DS3
goto 1

C
C
C
2 CALL MNCLR
CALL MNHEAD('Inizialization parameter')
CALL MNYENO (1,'Debug printing on video ',IPW,IPW)
CALL MNDEC (2,'Clock period ns ',PERIOD,PERIOD)
CALL MNDEC (3,'How many test for each set',KTEST,KTEST)
CALL MNDEC (4,'No. of words in memory',LOOP,LOOP)
CALL MNDEC (5,'No.of words trans.bus switch',NTRAS,NTRAS)

C
CALL MNDISP
CALL MNIN(NOPT)
NOPT = NOPT + 3
GO TO (999,10,2,2,2,2,2,2,2),NOPT

C
C

```



```

C
C
20  CONTINUE
C..... CECK SEQUENZIALE A LUNGO TERMINE
    RANDOM = 0
    ISAMP = 0
    ISEQ = 1
    CALL DLOUT ('SEQUENTIAL TEST- LONG TERM')
    BMINZ = 0
    DATMAX = 65535
    JAIN = 0
    PRINT 9081,BMINZ,JAIN
9081  FORMAT(1X,' A INIZ,B INIZ?  A,B,',
+ 2Z8,' ',',$',)
    READ 9082,AA,BB
    IF (AA.GT.0)JAIN =AA
    IF (BB.GT.0)BMINZ = BB
    BMAXZ = BMINZ + LOOP - 1
C
9082  FORMAT(2Z8)
    if (ipw.eq.0) then
        OPEN(UNIT=3,FILE='ERROR.DAT',TYPE='NEW',SHARED)
        write(3,'(a80)') ' Test sul bus switch risultati :'
    endif
C.....
    GO TO 60
C
C
C
30  CONTINUE
C..... CECK SAMPLING .....
    RANDOM = 0
    ISEQ=0
    ISAMP = 1
    CALL DLOUT(' SAMPLING TEST- LONG TERM ')
    BMINZ = 0
    DATMAX = 65535
    JAIN = 0
    if (ipw.eq.0) then
        OPEN(UNIT=3,FILE='ERROR.DAT',TYPE='NEW',SHARED)
        write(3,'(a80)') ' Test sul bus switch risultati :'
    endif
    GO TO 60
C
C
C
40  CONTINUE
C
C..... RANDOM
    LRAN = SECNDS(0.) I GET START VALUE
    CALL DLOUT(' RANDOM NUMBERS - LONG TERM ')
    RANDOM = 1
    ISEQ = 0
    ISAMP = 0
    GO TO 60
C
C
50  CONTINUE
C..... FILLING DATA ONE-BY-ONE
C
    CALL DLOUT(' FILLING DATA ONE-BY-ONE ')
    CALL DLDEC ('No. of words in memory',LOOP,LOOP)
    CALL DLDEC ('No.of words trans.bus switch',NTRAS,NTRAS)
    ISAMP = 0
    RANDOM =0
    ISEQ = 0
C
    GO TO 60
C
C
C
c
c
c
c
c
70  continue
    bin_word(1) = 1
    bin_word(2) = 65534
    zero(1) = 65535
    zero(2) = 0

```

```

75      alltest=1
      call cfsa(9,ds1a0,0,q)          ! reset ds1
      call cfsa(9,ds2a0,0,q)          ! reset ds2

c      set all errorx arrays to false

      do i =0,15
        error0(i) = .false.
        error1(i) = .false.
        error2(i) = .false.
        error3(i) = .false.
        error4(i) = .false.
        error5(i) = .false.
      enddo

      call mncir
      call mnhead('O.K. test : an automatic test for the bus switch ')
      call mnopt(1,'Cross talk test of Camac functions ')
      call mnopt(2,'Cross talk test of Ecl port A and B')
      call mnopt(3,'Cross talk test of Ecl port C')
      call mnopt(4,'Sampling test of Camac functions')
      call mnopt(5,'Sampling test of Ecl port A and B')
      call mnopt(6,'Sampling test of Ecl port C')

      call mnyeno(7,' a,b,c,d,e,f tests all together',alltest,alltest)
      call mnopt(8,'Test of two bus switch in cascade')
      call mndisp
      call mnin(opt_n)
      goto (999,1,1,1000,2000,3000,4000,5000,6000,75,7000),opt_n + 3

*
*      Cross talk test
*
1000    continue

c
c      Reading and writing registers A,B via Camac
c
      call cfsa(26,bus0,dummy,q) ! enables Camac functions
      print *, '***** Test of Camac functions of the bus switch *****'
      print *, '***** CROSS TALK TEST *****'
      call dlout(' ')
      bus(0) = bus0
      bus(1) = bus1
      camac(0,1) = .true.
      camac(1,1) = .true.

c
c      test of A (k=0) B (k=1)
c
      do k=0,1
        if (k.eq.0) then
          print *, 'Now writing (F16A0) and reading (F0A0) A register'
        else
          print *, 'Now writing (F16A1) and reading (F0A1) B register'
        endif

        do j=1,2
          in_word = bin_word(j)
          do i=0,15
            call cfsa(16,bus(k),zero(j),q) ! set to all 1 or 0
            if (.not.q) call dlout('no Q response on write unit:',sbus)
            call cfsa(16,bus(k),in_word,q) ! writes in A or B
            if (.not.q) call dlout('no Q response on write unit:',sbus)
            call cfsa(0,bus(k),out_word,q) ! reads A or B
            if (.not.q) call dlout('no Q response on read unit:',sbus)

            if (k.eq.0) then
              call testbit(out_word,in_word,camac(k,1),error0) ! test the word
            else
              call testbit(out_word,in_word,camac(k,1),error1)
            endif

            in_word = jshftc(in_word,1,16) ! shifts the word
          enddo
        enddo
      enddo
      call print_head
      call camac_resp(camac(0,1),camac(1,1))
      if (alltest.eq.0) goto 75

```

```

c
c      Writing by front panel and reading by Camac
c
c
2000  continue
      call cfsa(26,bus0,dummy,q) ! enables Camac functions
      call cfsa(16,clock1,7,q) ! set clock period
      call dlout('***** Test of Front panel writing *****')
      print *, '***** Writing A and B via front panel reading them
+ via Camac functions *****'
      print *, '***** CROSS TALK TEST *****'
      ecl_a(1) = .true.
      ecl_b(1) = .true.
      call dlout(' Now testing Ecl port of A and B ')

      do j=1,2
        in_word = bin_word(j)

        do i=0,15
          call cfsa(9,ds1a0,dummy,q) ! reset of a datastack
          call cfsa(9,ds2a0,dummy,q) ! idem for B
          call cfsa(16,ds1a0,zero(j),q) ! writes in A
          if (.not.q) call dlout('no Q response on write unit:',smem1)
          call cfsa(16,ds1a0,in_word,q) ! writes in A
          if (.not.q) call dlout('no Q response on write unit:',smem1)
          call cfsa(16,ds2a0,zero(j),q) ! writes in B
          if (.not.q) call dlout('no Q response on write unit:',smem2)
          call cfsa(16,ds2a0,in_word,q) ! writes in B
          if (.not.q) call dlout('no Q response on write unit:',smem2)
          call cfsa(16,ds1a1,0,q) ! rp = 0
          call cfsa(16,ds2a1,0,q) ! rp = 0
          call cfsa(26,clock0,dummy,q) ! start clock

          call cfsa(0,bus0,out_word,q) ! read A of B.S.
          if (.not.q) call dlout('no Q response on read unit:',sbus)
          call testbit(out_word,in_word,ecl_a(1),error2)
          call cfsa(0,bus1,out_word,q) ! read B of B.S.
          if (.not.q) call dlout('no Q response on read unit:',sbus)
          call testbit(out_word,in_word,ecl_b(1),error3)
          in_word = jishftc(in_word,1,16)
        enddo
      enddo
      call print_head
      call ecl_resp(ecl_a(1),ecl_b(1))
      print *, ' '

      if (alltest.eq.0) goto 75

c
c      Writing and reading by front panel
c
c
3000  continue
      call cfsa(24,bus0,dummy,q) ! disables camac functions
      call dlout('***** Test of front panel reading ***** ')
      call dlout('***** Writing A and B ,reading C via front panel ****')
      print *, '***** CROSS TALK TEST *****'
      call dlout(' ')
      ecl_c(1) = .true.
      do j=1,2
        in_word_a = bin_word(j)
        in_word_b = bin_word(j)

        do i=0,15
          call cfsa(9,ds1a0,dummy,q) ! reset of a datastack
          call cfsa(9,ds2a0,dummy,q) ! idem for B
          call cfsa(9,ds3a0,dummy,q) ! and C
          call cfsa(16,ds1a0,zero(j),q) ! writes in A
          if (.not.q) call dlout('no Q response on write unit:',smem1)
          call cfsa(16,ds1a0,in_word_a,q) ! writes in A
          if (.not.q) call dlout('no Q response on write unit:',smem1)
          call cfsa(16,ds2a0,zero(j),q) ! writes in B
          if (.not.q) call dlout('no Q response on write unit:',smem2)
          call cfsa(16,ds2a0,in_word_b,q) ! writes in B
          if (.not.q) call dlout('no Q response on write unit:',smem2)
          call cfsa(16,ds1a1,0,q) ! rp = 0
          call cfsa(16,ds2a1,0,q) ! rp = 0

          call cfsa(26,clock0,dummy,q) ! start clock

          call cfsa(16,ds3a1,2,q) ! reset rp of ds3

```



```

        call cfsa(0,ds3a0,out_word,q) I read ds3
        call testbit(in_word_b,out_word,ecl_c(1),error4)
        call cfsa(0,ds3a0,out_word,q) I read ds3
        call testbit(in_word_a,out_word,ecl_c(1),error5)
        in_word_a = jishftc(in_word_a,1,16)
        in_word_b = jishftc(in_word_b,1,16)
    enddo
enddo
call print_head
call ecl_c_resp(ecl_c(1))
if (alltest.eq.0) goto 75

*
*
*
c
    Sampling test

    set all errorx arrays to false

    do i = 0,15
        error0(i) = .false.
        error1(i) = .false.
        error2(i) = .false.
        error3(i) = .false.
        error4(i) = .false.
        error5(i) = .false.
    enddo

4000    continue

c
c
c
    Reading and writing registers A,B via Camac

    call cfsa(9,ds1a0,0,q)          I reset ds1
    call cfsa(9,ds2a0,0,q)          I reset ds2
    call cfsa(26,bus0,dummy,q) I enables Camac functions
    call dlout('***** Test of Camac functions of the bus switch *****')
    call dlout('***** SAMPLING TEST *****')
    call dlout(' ')
    bus(0) = bus0
    bus(1) = bus1
    camac(0,2) = .true.
    camac(1,2) = .true.

c
c
c
    test of A (k=0) B (k=1)

do k=0,1
    if (k.eq.0) then
        print *, 'Now writing (F16A0) and reading (F0A0) A register'
    else
        print *, 'Now writing (F16A1) and reading (F0A1) B register'
    endif

    do j=1,2
        in_word = 0
        do i=0,15
            call cfsa(16,bus(k),in_word,q) I writes in A or B
            if (.not.q) call dlout('no Q response on write unit:',sbus)
            call cfsa(0,bus(k),out_word,q) I reads A or B
            if (.not.q) call dlout('no Q response on read unit:',sbus)

            if (k.eq.0) then
                call testbit(out_word,in_word,camac(k,2),error0) I test the word
            else
                call testbit(out_word,in_word,camac(k,2),error1)
            endif

            call incr(in_word,in_word) I increments the word
        enddo
    enddo
enddo
call print_head
call camac_resp(camac(0,2),camac(1,2))
print *, ' '

if (alltest.eq.0) goto 75

c
c
c
c
    Writing by front panel and reading by Camac

5000    continue

```



```

call cfsa(26,bus0,dummy,q) ! enables Camac functions
call cfsa(16,clock1,7,q) ! set clock period
print *, '***** Test of Front panel writing *****'
print *, '***** Writing A and B via front panel reading them *****'
+ via Camac functions *****
print *, '***** SAMPLING TEST *****'
ecl_a(2) = .true.
ecl_b(2) = .true.
call dlout(' Now testing Ecl port of A and B ')

in_word_a = 0
do j=0,15
  in_word_b = 0

  do i=0,15
    call cfsa(9,ds1a0,dummy,q) ! reset of a datastack
    call cfsa(9,ds2a0,dummy,q) ! idem for B
    call cfsa(16,ds1a0,in_word_a,q) ! writes in A
    if (.not.q) call dlout('no Q response on write unit:',smem1)
    call cfsa(16,ds2a0,in_word_b,q) ! writes in B
    if (.not.q) call dlout('no Q response on write unit:',smem2)
    call cfsa(16,ds1a1,0,q) ! rp = 0
    call cfsa(16,ds2a1,0,q) ! rp = 0
    call cfsa(26,clock0,dummy,q) ! start clock

    call cfsa(0,bus0,out_word,q) ! read A of B.S.
    if (.not.q) call dlout('no Q response on read unit:',sbus)
    call testbit(out_word,in_word_a,ecl_a(2),error2)
    call cfsa(0,bus1,out_word,q) ! read B of B.S.
    if (.not.q) call dlout('no Q response on read unit:',sbus)
    call testbit(out_word,in_word_b,ecl_b(2),error3)
    call incr(in_word_b,in_word_b)
  enddo
  call incr(in_word_a,in_word_a)
enddo
call print_head
call ecl_resp(ecl_a(2),ecl_b(2))
print *, ' '

if (alltest.eq.0) goto 75

c
c
c
6000
Writing and reading by front panel
continue
call cfsa(24,bus0,dummy,q) ! disables camac functions
call dlout('***** Test of front panel reading *****')
call dlout('***** Writing A and B ,reading C via front panel *****')
call dlout('***** SAMPLING TEST *****')
call dlout(' ')
ecl_c(2) = .true.
in_word_a = 0
do j=0,15
  in_word_b = 0

  do i=0,15
    call cfsa(9,ds1a0,dummy,q) ! reset of a datastack
    call cfsa(9,ds2a0,dummy,q) ! idem for B
    call cfsa(9,ds3a0,dummy,q) ! and C
    call cfsa(16,ds1a0,in_word_a,q) ! writes in A
    if (.not.q) call dlout('no Q response on write unit:',smem1)
    call cfsa(16,ds2a0,in_word_b,q) ! writes in B
    if (.not.q) call dlout('no Q response on write unit:',smem2)
    call cfsa(16,ds1a1,0,q) ! rp = 0
    call cfsa(16,ds2a1,0,q) ! rp = 0

    call cfsa(26,clock0,dummy,q) ! start clock

    call cfsa(16,ds3a1,0,q) ! reset rp of ds3
    call cfsa(0,ds3a0,out_word,q) ! read ds3
    call testbit(in_word_b,out_word,ecl_c(2),error4)
    call cfsa(0,ds3a0,out_word,q) ! read ds3
    call testbit(in_word_a,out_word,ecl_c(2),error5)
    call incr(in_word_b,in_word_b)
  enddo
  call incr(in_word_a,in_word_a)
enddo
call print_head
call ecl_c_resp(ecl_c(2))
c

```

```

c      summary of the test
c
if (alltest.eq.1) then
  call dkey('Press any key to see the summary of results')
  write(*, '(1x,/,/,/,/)')
  print *, '***** SUMMARY OF RESULTS *****'
  write(*, '(1x,/,/,/,/)')
  print *, 'Test:      | Camac | Camac | Ecl | Ecl | Ecl '
  print *, '              | fun. A | fun. B | port A | port B | port C '
  print *, '-----+-----+-----+-----+-----+-----'
  do k=1,2
    if (k.eq.1) then
      a13 = 'Cross talk'
    else
      a13 = 'Sampling'
    endif

    if (camac(0,k)) then
      a1 = '| o.k.'
    else
      a1 = '| bad '
    endif
    if (camac(1,k)) then
      a2 = '| o.k.'
    else
      a2 = '| bad '
    endif
    if (ecl_a(k)) then
      a3 = '| o.k.'
    else
      a3 = '| bad '
    endif
    if (ecl_b(k)) then
      a4 = '| o.k.'
    else
      a4 = '| bad '
    endif
    if (ecl_c(k)) then
      a5 = '| o.k.'
    else
      a5 = '| bad '
    endif
    print *, a13, a1, a2, a3, a4, a5
  enddo
endif
write(*, '(1x,/,/,/,/)')
call dkey('Press any key to continue')
goto 75

```

```

c
c
c      Test of two bus switch in cascade
c
c
c      continue
7000 call inizeclock(b,c,sclk,60,25,25,25,25) ! set clock period
      call cfsa(26,bus2_a_0,dummy,q)

c
c      Programming timing data stack
c
      call cfsa(9,ds4a0,dummy,q) ! mstr of timing stack
      call cfsa(16,ds4a0,19,q)   ! writes the timing word
      if (.not.q) call dlout('No q response on write unit:',ds)
      call cfsa(0,ds4a0,out_word,q) ! reads word
      if (out_word.ne.19) call dlout('Error on writing 19')

      call cfsa(16,ds4a0,21,q)   ! writes the timing word
      if (.not.q) call dlout('No q response on write unit:',ds)
      call cfsa(0,ds4a0,out_word,q) ! reads word
      if (out_word.ne.21) call dlout('Error on writing 21')

      call cfsa(16,ds4a0,12,q)   ! writes the timing word
      if (.not.q) call dlout('No q response on write unit:',ds)
      call cfsa(0,ds4a0,out_word,q) ! reads word
      if (out_word.ne.12) call dlout('Error on writing 12')

      call cfsa(16,ds4a0,20,q)   ! writes the timing word
      if (.not.q) call dlout('No q response on write unit:',ds)
      call cfsa(0,ds4a0,out_word,q) ! reads word
      if (out_word.ne.20) call dlout('Error on writing 20')

```

```

call cfsa(16,ds4a0,16,q)  I writes the timing word
if (.not.q) call dlout('No q response on write unit:',ds)
call cfsa(0,ds4a0,out_word,q) I reads word
if (out_word.ne.16) call dlout('Error on writing 16')

call cfsa(16,ds4a0,8,q)  I writes the timing word
if (.not.q) call dlout('No q response on write unit:',ds)
call cfsa(0,ds4a0,out_word,q) I reads word
if (out_word.ne.8) call dlout('Error on writing 8')

call cfsa(16,ds4a0,16,q)  I writes the timing word
if (.not.q) call dlout('No q response on write unit:',ds)
call cfsa(0,ds4a0,out_word,q) I reads word
if (out_word.ne.16) call dlout('Error on writing 16')

call cfsa(16,ds4a0,48,q)  I writes the timing word
if (.not.q) call dlout('No q response on write unit:',ds)
call cfsa(0,ds4a0,out_word,q) I reads word
if (out_word.ne.48) call dlout('Error on writing 48')

call cfsa(16,ds4a1,0,q) I reset read pointer

print *, '***** Test of two bus switch in cascade *****'
print *, '***** Be carefull of hardware connections,*****'
print *, '***** now must be changed (see doc.) *****'
call dkey('Press any key when you are ready')
print *, 'Now test in progress '
ecl_c(2) = .true.
do i=0,15
    error4(i)=.false.
    error5(i)=.false.
enddo
in_word_a = 0
do j=0,15
    in_word_b = 0

    do i=0,15
        call cfsa(9,ds1a0,dummy,q) I reset of a datastack
        call cfsa(9,ds2a0,dummy,q) I idem for B
        call cfsa(9,ds3a0,dummy,q) I and C
        call cfsa(16,ds1a0,in_word_a,q) I writes in A
        if (.not.q) call dlout('no Q response on write unit:',smem1)
        call cfsa(16,ds2a0,in_word_b,q) I writes in B
        if (.not.q) call dlout('no Q response on write unit:',smem2)
        call cfsa(16,bus2_a_0,in_word_b,q) I writes in bus2 a
        if (.not.q) call dlout('No Q response on write unit:',bus2)
        call cfsa(16,bus2_a_1,in_word_a,q) I writes b
        if (.not.q) call dlout('No Q response on write unit:',bus2)

        call cfsa(16,ds1a1,0,q) I rp =0
        call cfsa(16,ds2a1,0,q) I rp = 0
        call cfsa(25,clock0,dummy,q) I one shot to clock
        call cfsa(26,clock0,dummy,q) I start clock

        call cfsa(16,ds3a1,0,q) I reset rp of ds3
        call cfsa(0,ds3a0,out_word,q) I read ds3
        call testbit(in_word_b,out_word,ecl_c(2),error4)
        call cfsa(0,ds3a0,out_word,q) I read ds3
        call testbit(in_word_a,out_word,ecl_c(2),error4)
        call cfsa(0,ds3a0,out_word,q) I read ds3
        call testbit(in_word_a,out_word,ecl_c(2),error5)
        call cfsa(0,ds3a0,out_word,q) I read ds3
        call testbit(in_word_b,out_word,ecl_c(2),error5)
        call incr(in_word_b,in_word_b)
    enddo
    call incr(in_word_a,in_word_a)
enddo
call print_head
if (ecl_c(2)) then
    call dlout('**** The transmission has been succesfull !****')
    call dkey('Press any key to continue')
else
    call dlout('Errors in transmission on bit #')
    do i =0, 15
        if (error5(i).or.error4(i)) write(*,'(40x,i2)'),i
    enddo
    call dkey('Press any key to continue')
endif
call inizeclock(b,c,sclk,period,25,45,25,25) I set delay on out 2

```



```

      goto 75
60  CONTINUE
C----- FILLING MEMORY ONE BY ONE
      IF (ISAMP+RANDOM+ISEQ.EQ.0) THEN
      DO N=1,LOOP
      PRINT 8030,N,DA(N),DB(N)
8030  FORMAT(1X,'ENTER WORD:',I4,' WORD-A(H)=' ,Z5
+ , ' WORD-B(H)=' ,Z5,' :: ', $)

      READ 9030,WDS1,WDS2
9030  FORMAT(2Z8)
C
      DA(N)=WDS1
      DB(N)=WDS2
C
      CALL CFSA(16,DS1A0,DA(N),Q)
      IF(.NOT.Q) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',
+SMEM1)
      CALL CFSA(16,DS2A0,DB(N),Q)
      IF(.NOT.Q) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',
+SMEM2)
      ENDDO
C
C
C
C
      CALL CFSA(16,DS1A1,0,Q)          IRESET READ POINTER
      CALL CFSA(16,DS2A1,0,Q)
      DO N=1,LOOP
      CALL CFSA(0,DS1A0,WDS1,Q)
      IF(.NOT.Q) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM1)
      CALL CFSA(0,DS2A0,WDS2,Q)
      IF(.NOT.Q) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM2)
C
C
      PRINT 8040,N,WDS1,WDS2
8040  FORMAT(1X,'WORD:',I5
+ , ' IN-A:',Z8,' IN-B:',Z8)
      ENDDO
      ENDIF
C
C
C----- SEQUENTIAL TESTS
C
      IF (ISEQ .EQ. 1) THEN
      CONT = 0
      BUFF = JAIN
805  CONTINUE
      JAIN = BUFF
      JA = JAIN
      JBMIN = BMINZ
      JBMAX = BMAXZ
C.....          STARTING POINT FOR SEQ. DATA
810  CONTINUE
      DO 830 JB =JBMIN,JBMAX

      DA(N)=JA
      DB(N)=JB
      CALL CFSA(16,DS1A0,DA(N),Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',
+SMEM1)
      CALL CFSA(16,DS2A0,DB(N),Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',
+SMEM2)
      JA = JA + 1
830  CONTINUE
      PRINT 8091,JBMIN,JBMAX,JAIN,(JA-1)
8091  FORMAT(1X,'MIN B: ',I8,' MAX B: ',I8,' MIN A: ',I8,' MAX A:',
+I8)
      CALL CFSA (16,DS1A1,0,Q)          IRESET READ POINTER
      CALL CFSA (16,DS2A1,0,Q)          IRESET READ POINTER
C
      DO N=1,LOOP
      CALL CFSA(0,DS1A0,WDS1,Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM1)

```



```

      CALL CFSA(0,DS2A0,WDS2,Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM2)
C
C
      IF(IPW.NE.0) PRINT 8040,N,WDS1,WDS2
      ENDDO
      ENDIF
C
C----- SAMPLING TESTS
829  CONTINUE
      IF (ISAMP.EQ. 1) THEN
C..... Condizioni iniziali
C
      JB = BMINZ
      JA = JAIN
C..... STARTING POINT FOR SAMP.DATA
      N = 0
831  CONTINUE
      N = N + 1
      DA(N)=JA
      DB(N)=JB
      CALL CFSA(16,DS1A0,DA(N),Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',
+SMEM1)
      CALL CFSA(16,DS2A0,DB(N),Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',
+SMEM2)
      IF(N.EQ.LOOP)GO TO 832
      CALL INCR (JB,JBOUT)
      JB = JBOUT
      IF (JA .GT. DATMAX) GO TO 832
      IF(JB.GT.DATMAX) THEN
        CALL INCR (JA,JAOUT)
        JA = JAOUT
        JB = BMINZ
      ENDIF
C
C
      GO TO 831
832  CONTINUE
      JB = JBMAX
      PRINT 8091,JBMIN,JBMAX,JA
      IF(ISAMP .EQ. 1) THEN
        IF(JBMIN.EQ.0.OR.JBMAX.EQ.DATMAX)THEN
          DO JJ = 1,DATMAX,LOOP
            JTES =JJ -1
            IF(JTES.EQ.JA.and.ipw.eq.0)WRITE(3,8091),JBMIN,JBMAX,JA
          ENDDO
        ENDIF
      ENDIF
C
C
C
C
      CALL CFSA(16,DS1A1,0,Q)      I RESET READ POINTER
      CALL CFSA(16,DS2A1,0,Q)
      DO N=1,LOOP
      CALL CFSA(0,DS1A0,WDS1,Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM1)
      CALL CFSA(0,DS2A0,WDS2,Q)
      IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM2)
C
C
      IF(IPW.NE.0) PRINT 8040,N,WDS1,WDS2
      ENDDO
      ENDIF
C.....
C----- RANDOM DATA
C
      IF(RANDOM.EQ.1) THEN
C
C
      DO N=1,LOOP
      DA(N)=RAN(LRAN)*65536.
      DB(N)=RAN(LRAN)*65536
      CALL CFSA(16,DS1A0,DA(N),Q)
      IF(.NOT.Q) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',

```

```

+SMEM1)
CALL CFSA(16,DS2A0,DB(N),Q)
IF(.NOT.Q) CALL DLOUT(' NO Q reponse ON WRITE:UNIT:',
+SMEM2)
ENDDO
ENDIF
C
C
C
C
C
PRINT 4055,cont,LOOP
4055 FORMAT(1X,'TEST:',I8,' #WORDS:',I5)
CALL CFSA( 9,DS3A0,0,Q)
C
C
RESET READ POINTERS AND SET WP TO NO. OF DATA TO XMIT
C
CALL CFSA(16,DS1A1,0,Q)
CALL CFSA(16,DS2A1,0,Q)

CALL CFSA(16,DS1A2,(NTRAS),Q)
CALL CFSA(16,DS2A2,(NTRAS),Q)
C
PRD=(PERIOD-40)/10
CALL CFSA(16,CLOCK1,PRD,Q) I SET CLOCK PERIOD
CALL CFSA(26,CLOCK0,0,Q) I START
C
C
READ RESULT
C
CALL DLOUT(' ')
CALL DLOUT(' ' RESULT ' ')
CALL DLOUT(' ')
C
C
C
CALL CFSA(9 ,DS1A0,0,Q)
CALL CFSA(9 ,DS2A0,0,Q)
CALL CFSA(9 ,DS3A0,0,Q)
C
CALL CFSA(16,DS3A2,255,Q)
CALL CFSA(16,DS1A2,255,Q)
CALL CFSA(16,DS2A2,255,Q)
CALL CFSA(16,DS1A1,0,Q)
CALL CFSA(16,DS2A1,0,Q)
CALL CFSA(16,DS3A1,0,Q)
C
NTOT = NTRAS*2
DO N=1,NTRAS
CALL CFSA(0,DS1A0,RA(N),Q)
IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM1)
CALL CFSA(0,DS2A0,RB(N),Q)
IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SMEM2)
ENDDO
DO N=1,NTOT
CALL CFSA(0,DS3A0,RER(N),Q)
IF(.NOT.Q.AND.IPW.NE.0) CALL DLOUT(' NO Q reponse ON READ :UNIT:',
+SDST)
ENDDO
C
C
C
DO 250 N=1,NTOT
C
NDIV = N / 2
NN = NDIV*2
IF (NN.NE.N)JRESULT =RB(NDIV+1)
IF (NN.EQ.N)JRESULT =RA(NDIV)
C
KRESULT=JRESULT
INT2(2)=0
ERRC=1
IF(RER(N).NE.KRESULT) ERRC=2
IF(IPW.NE.0)PRINT 2010,
+N,RA(N),RB(N),RER(N),KRESULT,STATUS(ERRC)
C
C
IF(IPW.EQ.0.AND.ERRC.EQ.2) THEN
C

```

```

C
C
C
PRINT 2010,N,RA(N),RB(N),RER(N),KRESULT,STATUS(ERRC)
IF((ISEQ.EQ.1.OR.ISAMP.EQ.1).and.ipw.eq.0)WRITE(3,2010)N,RA(N),RB(N)
+,RER(N),KRESULT,STATUS(ERRC)
C
2010 FORMAT(1X,'WORD=',I4,' A=',Z5,' B=',Z5
+ , ' RESULT=',Z5,' CHECK:',Z5,A8)
C
C
ENDIF
C
C
C
C
JRESULT=RER(N)
JRESO = RER(N)
ERRCO = ERRC
250 CONTINUE
C
C
C
C
C
INCREMENTATION OF LIMITS
C
IF(ISEQ.EQ.1.AND.JBMAX.LT.DATMAX) THEN
JBMIN = JBMIN + LOOP
JBMAX = JBMAX + LOOP
JAIN = JAIN + LOOP
CALL CFSA(9,DS1A0,0,Q) I RESET DS1
CALL CFSA(9,DS2A0,0,Q) I RESET DS2
CALL CFSA(9,DS3A0,0,Q) I RESET DS3
GO TO 810
ENDIF
C
C
IF (ISEQ.EQ.1) THEN
PRINT 8200
8200 FORMAT(1X,' *** END **')
CONT = CONT + 1
write (3,'(a20,i4)') ' fine test ',cont
CALL CFSA(9,DS1A0,0,Q) I RESET DS1
CALL CFSA(9,DS2A0,0,Q) I RESET DS2
CALL CFSA(9,DS3A0,0,Q) I RESET DS3
IF (KTEST.ge.CONT) GOTO 805
ENDIF
C
C
C
C
c = ISAMP+RANDOM+ISEQ
IF(c.EQ.1.or.c.eq.0) THEN
cont = 0
CALL DLKEY('Press ANY KEY to continue')
CALL CFSA(9,DS1A0,0,Q) I RESET DS1
CALL CFSA(9,DS2A0,0,Q) I RESET DS2
CALL CFSA(9,DS3A0,0,Q) I RESET DS3
GO TO 1
ENDIF
999 CONTINUE
CALL PARWT
return
END
C
c
c
c
subroutine camac_resp:
prints all the errors bit by bit
c
subroutine camac_resp(camac0,camac1)
logical camac0,camac1
logical error0(0:15),error1(0:15),error2(0:15),error3(0:15)
+,error4(0:15),error5(0:15)
common / bits /error0,error1,error2,error3,error4,error5
if (camac0.and.camac1)then
call dlout('Camac dataway it's ok !')

```



```

c      a program to test the Le Croy 2375 data stack
c
c      by Paolo Pacchiarotti
c
c*****

      subroutine DATATEST

      implicit integer (a-z)
      integer*4 written(0:255),bin_word(2)
      character*8 status
      character*80 form1,form2,form3
      logical resp

      common/mncomm/nwrds,mnbuf(500)

      common/oppars/b,c,n,outf

      data nwrds/500/

c
c      Begin of program
c

      call parfil('datatest.las')
      call parrd

c
c      Camac initialization
c
      call iniz_camac(1,2)

c
c      Top Menu
c

10      continue
      yesno=0
      call mncir
      call mnhead ('Program to test the data stack','      Top Menu  ')
      call mnopt (1,'Initialize the set up')
      call mnopt (2,'Test writing & reading into datastack')
      call mnopt (3,'Handling pointers')
      call mnopt (4,'Write & Read data')
      call mnopt (5,'Check the set up')
      call mnopt (6,'Master Reset')
      call mnopt (7,'Stop')
      call mndisp
      call mnin(opt_n)
      opt_n = opt_n + 3
      goto (9999,10,10,100,200,300,400,500,600,9999),opt_n

c
c      Get module registers
c

100      continue
      call cdreg (data_a_0,b,c,n,0)
      call cdreg (data_a_1,b,c,n,1)
      call cdreg (data_a_2,b,c,n,2)

c
c      initialize formats
c
      if (outf.eq.0) then
         form1='(1x,a9,i3,2(a8,i5),a12,a8)'
         form2='(1x,a9,i3,a7,i5)'
         form3='(i5)'
      else
         form1='(1x,a9,i3,2(a8,z4),a12,a8)'
         form2='(1x,a9,i3,a7,z4)'
         form3='(z5)'
      endif

      goto 10

c
c      Test Write & Read in datastack : menu

```

```

c
200  continue
    call mncir
    call mnhead (' Test writing & reading in datastack')
    call mnopt (1,'Sequential test by 17 (hex = 11)')
    call mnopt (2,'Testing with 1010... and 0101...')
    call mndisp
    call mnin (opt_n)
    opt_n = opt_n +3
    goto (9999,10,10,210,220),opt_n

c
c
c
c
c
Sequential test :
    write and read a sequence of number.
    The sequence is 0000,0011,0022...

210  continue
    call cfsc (9,data_a_0,dummy,resp)  ! MRST
    word = 0
    do i=0,255
        written(i)=word
        call cfsc(16,data_a_0,word,resp) ! writes word into data stack
        call response(n,resp,i)
        word = word + 17                ! increments word
    enddo

    do i=0,255
        call cfsc(0,data_a_0,word,resp) ! reads in data stack
        call response (n,resp,i)
        if (word.eq.written(i)) then
            status = ' O.K. '
        else
            status = ' ERROR '
        endif
        write (*,form1)'Pointer: ',i,' Write: ',written(i),' Read: '
+       ,word,' Status: ',status

    enddo
    call dikey('Press any key to continue')
    goto 10

c
c
c
Test with 1010101010101010 and 0101010101010101

220  continue
    bin_word(2)=21845 ! decimal for 0101010101010101
    bin_word(1)=43690 ! decimal for 1010101010101010
    do k=1,2
        word=bin_word(k)
        call cfsc (9,data_a_0,dummy,resp)
        do i=0,255
            written(i)=word
            call cfsc(16,data_a_0,word,resp) ! writes in
            call response(n,resp,i)
        enddo
        do i=0,255
            call cfsc(0,data_a_0,word,resp)
            call response(n,resp,i)
            if (word.eq.written(i)) then
                status = ' O.K. '
            else
                status = ' ERROR '
            endif
            write (*,form1)'Pointer: ',i,' Write: ',written(i),' Read: '
+            ,word,' Status: ',status
        enddo
        call dikey('Press any key to continue')
        enddo
        goto 10

c
c
c
c
c
Handling Pointers : menu

300  continue
    call mncir

```



```

      goto 400
c
c      Check the set up : menu
c
500      continue
      call mnclr
      call mnhead ('      Set up configuration: ')
      call mndec (1,'Branch',b,b)
      call mndec (2,'Crate',c,c)
      call mndec (3,'Data Stack in:',n,n)
      call mndec (4,'i/o of data in hex (1) or in decimal(0)',outf,outf)
      call mndisp
      call mnin(opt_n)
      opt_n = opt_n + 3
      goto (9999,100,10,500,500,500,500),opt_n

c
c      Master Reset
c
600      continue
      call cfsa(9,data_a_0,dummy,resp)
      goto 10

c
c      STOP
c
9999      continue
      call parwt 1 save parameters
      return
      end

*****
*
*      TRANSTEST:
*      a program to test the data transmission
*      via front panel between two data stacks
*
*      by Paolo Pacchiarotti
*
*
*      Symbols:
*      ck: clock generator
*      dss : data stack source
*      dsd : data stack destination
*
*****

c
c
      subroutine TRANSTEST
      implicit integer (a-z)
      integer*4 written(0:255),data(0:255),bin_word(2),zero(2)
      character*8 status
      logical resp
      character*80 form1

      common/mncomm/nwrds,mnbuf(500)
      common/oppars/b,c,ck,dss,dsd,per,num_words,outf,delay,delay1
+      ,ck1,errflag,delay0
      data nwrds/500/

c
c      Begin of program
c

      call parfil('transtest.las')
      call parrd

c
c      Top menu
c
10      continue
      yesno=0
      call mnclr
      call mnhead ('  Program to test the data transmission', '      Top
+      Menu')
      call mnopt (1,'Initialize the set up')

```



```

call mnopt (2,'Transmission of data via front panel')
call mnopt (3,'Check the set up')
call mnopt (4,'Stop')
call mndisp
call mnin(opt_n)
opt_n = opt_n + 3
goto (9999,10,10,100,200,300,9999),opt_n

c
c
c
100 Initialize set up : gets module registers
      camac initialization
      continue

      call iniz_camac(b,c)
      call cdreg (dss_a_0,b,c,dss,0)
      call cdreg (dss_a_1,b,c,dss,1)
      call cdreg (dss_a_2,b,c,dss,2)
      call cdreg (dsd_a_0,b,c,dsd,0)
      call cdreg (dsd_a_1,b,c,dsd,1)
      call cdreg (dsd_a_2,b,c,dsd,2)
      call cdreg (ck_a_0,b,c,ck,0)
      call cdreg (ck_a_1,b,c,ck,1)
      nper =(per-40)/10
      seed = 13
      call inizclock(b,c,ck,per,delay0,delay,25,25,0)
      call inizclock(b,c,ck1,40,delay1,25,25,25,0)

c
c
c
      set up of i/o formats

      if (outf.eq.0) then      ! decimal
          form1='(1x,a8,i3,2(a26,i5),/,a24,i5,35x,a9,a8)'
      else                    ! hex
          form1='(1x,a8,i3,2(a26,z4),/,a24,z4,35x,a9,a8)'
      endif

      goto 10

c
c
c
200 Test of Trasmission : menu
      continue
      call mnclr
      call mnhead('Test of transmission ')
      call mnopt(1,'Sequential test by 17 (hex = 11)')
      call mnopt(2,'Test with 1010... and 0101...')
      call mnopt(3,'Random number test ')
      call mnopt(4,'Sampling test')
      call mnopt(5,'Cross talk test')
      call mndisp
      call mnin(opt_n)
      opt_n = opt_n + 3
      goto (9999,10,10,210,220,230,240,250),opt_n

c
c
c
210 Sequential test
      continue

c
c
c
      Write data stack source (dss) via camac

      call cfsc(9,dsd_a_0,dummy,resp) ! MRST
      call cfsc(9,dss_a_0,dummy,resp) ! MRST
      word = 0
      do i=0,num_words
          written(i)=word
          call cfsc(16,dss_a_0,word,resp) ! writes in ds
          call response (dss,resp,i)
          call cfsc(0,dss_a_0,data(i),resp) ! reads what is written
          call response (dss,resp,i)
          word = word + 17
      enddo

c
c
c
      reset the read pointer

      call cfsc(16,dss_a_1,0,resp)

c
c
c
      Start by clock

```

```

c      call cfsa(26,ck_a_0,dummy,resp)
c
c      Read the destination data stack
c
      call cfsa(16,dsd_a_1,0,resp) ! reset rp
      do i=0,num_words
        call cfsa(0,dsd_a_1,pointer,resp)
        call cfsa(0,dsd_a_0,word,resp)
        call response(dsd,resp,i)
        if (word.eq.written(i).and.written(i).eq.data(i)) then
          status = ' O.K. '
        else
          status = ' ERROR '
        endif
        if(errflag.eq.0.or.(errflag.eq.1.and.status.eq.' ERROR'))then
          write(*,form1)'Pointer:',pointer,' Word in source (write) :'
+         ,written(i),' Word in source (read) :',data(i),
+         ' Word in destination:',word,'Status: ',status
        endif
      enddo
      call dikey('Press any key to continue')
      goto 10
c
c      Cross talk test
c
250    continue
c
c      Write data stack source (dss) via camac
c
      call cfsa(9,dsd_a_0,dummy,resp) ! MRST
      call cfsa(9,dss_a_0,dummy,resp) ! MRST
      bin_word(1)=1
      bin_word(2)=65534
      zero(1) = 65535
      zero(2) = 0
      do j=1,2
        in_word = bin_word(j)
        do i=0,31,2
          written(i+32*(j-1))= zero(j)
          call cfsa(16,dss_a_0,zero(j),resp) ! writes in ds
          call response (dss,resp,i)
          call cfsa(0,dss_a_0,data(i+32*(j-1)),resp) ! reads what is written
          call response (dss,resp,i)
          written(i+1+32*(j-1)) = in_word
          call cfsa(16,dss_a_0,in_word,resp)
          call response (dss,resp,i)
          call cfsa(0,dss_a_0,data(i+1+32*(j-1)),resp) ! reads what is written
          call response (dss,resp,i)
          in_word = jshftc(in_word,1,16)
        enddo
      enddo
c
c      reset the read pointer
c
      call cfsa(16,dsd_a_1,0,resp)
c
c      Start by clock
c
      call cfsa(26,ck_a_0,dummy,resp)
c
c      Read the destination data stack
c
      call cfsa(16,dsd_a_1,0,resp) ! reset rp
      do i=0,63
        call cfsa(0,dsd_a_1,pointer,resp)
        call cfsa(0,dsd_a_0,word,resp)
        call response(dsd,resp,i)
        if (word.eq.written(i).and.written(i).eq.data(i)) then
          status = ' O.K. '
        else
          status = ' ERROR '
        endif
        if(errflag.eq.0.or.(errflag.eq.1.and.status.eq.' ERROR'))then
          write(*,form1)'Pointer:',pointer,' Word in source (write) :'
+         ,written(i),' Word in source (read) :',data(i),
+         ' Word in destination:',word,'Status: ',status
        endif
      enddo

```

```

        call dikey('Press any key to continue')
        goto 10
c
c      Sampling test
c
240    continue

c
c      Write data stack source (dss) via camac
c
        call cfsa(9,dsd_a_0,dummy,resp) ! MRST
        call cfsa(9,dss_a_0,dummy,resp) ! MRST
        word = 0
        do i=0,15
            written(i)=word
            call cfsa(16,dss_a_0,word,resp) ! writes in ds
            call response (dss,resp,i)
            call cfsa(0,dss_a_0,data(i),resp) ! reads what is written
            call response (dss,resp,i)
            word = word + '1111'x
        enddo

c
c      reset the read pointer
c
        call cfsa(16,dss_a_1,0,resp)

c
c      Start by clock
c
        call cfsa(26,ck_a_0,dummy,resp)

c
c      Read the destination data stack
c
        call cfsa(16,dsd_a_1,0,resp) ! reset rp
        do i=0,15
            call cfsa(0,dsd_a_1,pointer,resp)
            call cfsa(0,dsd_a_0,word,resp)
            call response(dsd,resp,i)
            if (word.eq.written(i).and.written(i).eq.data(i)) then
                status = ' O.K. '
            else
                status = ' ERROR '
            endif
            if(errflag.eq.0.or.(errflag.eq.1.and.status.eq.' ERROR'))then
                write(*,form1)'Pointer:',pointer,' Word in source (write) :'
+               ,written(i),' Word in source (read) :',data(i),
+               ' Word in destination:',word,'Status: ',status
            endif
        enddo
        call dikey('Press any key to continue')
        goto 10

c
c      Test with 1010101010101010 and 0101010101010101
c
220    continue
        bin_word(1)=21845 ! decimal for 0101010101010101
        bin_word(2)=43690 ! decimal for 1010101010101010
        do k=1,2
            word=bin_word(k)
            call cfsa (9,dsd_a_0,dummy,resp)
            call cfsa (9,dss_a_0,dummy,resp)
            do i=0,num_words
                written(i)=word
                call cfsa(16,dss_a_0,word,resp) ! writes in
                call response(dss,resp,i)
                call cfsa(0,dss_a_0,data(i),resp) ! reads what is written
                call response (dss,resp,i)
            enddo

c
c      reset the read pointer
c
        call cfsa(16,dss_a_1,0,resp)

c
c      Start clock
c

```

```

call cfsc(26,ck_a_0,dummy,resp)

c
c
c   Read the destination data stack
c
c   call cfsc(16,dsd_a_1,0,resp) lreset rp
c   do i=0,num_words
c     call cfsc(0,dsd_a_1,pointer,resp)
c     call cfsc(0,dsd_a_0,word,resp)
c     call response(dsd,resp,i)
c     if (word.eq.written(i).and.written(i).eq.data(i)) then
c       status = ' O.K. '
c     else
c       status = ' ERROR '
c     endif
c     if(errflag.eq.0.or.(errflag.eq.1.and.status.eq.' ERROR'))then
c       write(*,form1)'Pointer:',pointer,' Word in source (write) :',
+       ,written(i),' Word in source (read) :',data(i),
+       ' Word in destination:',word,'Status: ',status
c     endif
c   enddo
c   call dikey('Press any key to continue')
c   enddo
c   goto 10

c
c   random number test
c
c
c230  continue
c
c   Write data stack source (dss) via camac
c
c   call cfsc(9,dsd_a_0,dummy,resp) l MRST
c   call cfsc(9,dss_a_0,dummy,resp) l MRST
c   word = ran(seed)*65536
c   do i=0,num_words
c     written(i)=word
c     call cfsc(16,dss_a_0,word,resp) l writes in ds
c     call response (dss,resp,i)
c     call cfsc(0,dss_a_0,data(i),resp) l reads what is written
c     call response (dss,resp,i)
c     word = ran(seed)*65536
c   enddo

c
c   reset the read pointer
c
c   call cfsc(16,dss_a_1,0,resp)

c
c   Start by clock
c
c   call cfsc(26,ck_a_0,dummy,resp)

c
c   Read the destination data stack
c
c   call cfsc(16,dsd_a_1,0,resp) l reset rp
c   do i=0,num_words
c     call cfsc(0,dsd_a_1,pointer,resp)
c     call cfsc(0,dsd_a_0,word,resp)
c     call response(dsd,resp,i)
c     if (word.eq.written(i).and.written(i).eq.data(i)) then
c       status = ' O.K. '
c     else
c       status = ' ERROR '
c     endif
c     if(errflag.eq.0.or.(errflag.eq.1.and.status.eq.' ERROR'))then
c       write(*,form1)'Pointer:',pointer,' Word in source (write) :',
+       ,written(i),' Word in source (read) :',data(i),
+       ' Word in destination:',word,'Status: ',status
c     endif
c   enddo
c   call dikey('Press any key to continue')
c   goto 10

c
c   Check the set up
c
c300  continue
c   call mnclr
c   call mnhead(' Set up configuration ')

```



```

      call mndec(1,'Branch',b,b)
      call mndec(2,'Crate',c,c)
      call mndec(3,'Source data stack in',dss,dss)
      call mndec(4,'Destination data stack in',dsd,dsd)
      call mndec(5,'Clock generator in',ck,ck)
      call mndec(6,'Clock Period (ns)',per,per,40,190)
      call mndec(7,'No. of words in transmission (0-255)',num_words,
+      num_words)
      call mndec(8,'I/O of data in hex(1) or decimal(0)',outf,outf)
      call mndec(9,'Print all(0) or only errors (1)',errflag,errflag)
      call mnnopt(10,' Set up only for TAU 1-2 measurement')
      call mndec(11,' Clock generator 2 in',ck1,ck1)
      call mndec(12,'Delay on out 1 clock 1 (RE DS)',delay0,delay0)
      call mndec(13,'Delay on out 2 clock 1 (o.s. clock 2)'
+      ,delay,delay,25,100)
      call mndec(14,'Delay on out 1 clock 2',delay1,delay1,25,100)
      call mndisp
      call mnin(opt_n)
      opt_n = opt_n + 3
      goto (9999,100,10,300,300,300,300,300,300,300,300,300,300,300,300,300
+      ,300,300,300),opt_n

c
c      STOP
c
9999  continue
      call parwt 1 save parameters
      return
      end

*****
*      MLUTEST:                                *
*      program to test the Le Croy 2372        *
*      Memory Lookup Unit                      *
*                                              *
*      by P. Pacchiarotti                     *
*                                              *
*      july 86                                *
*****

      Subroutine MLUTEST

      implicit integer (a-z)

      integer bin_word(2),written(0:255),zero(2)

      character*80 form1,form2,form3,form4

      logical q

      common/mncomm/nwrds,mnbuf(500)
+      common/oppars/b,c,mlu,dsa,dsb,ck,per,dim,mode,delay,delay1
      ,all,ck1

      data nwrds/500/

c
c      read parameters from mlutest.las
c
      call parfil('mlutest.las')
      call parrd

c
c      Top level menu
c
10    continue
      call mnclr
      call mnhead('***** Test of Mlu module *****')
      call mnopt(1,'Test of programming MLU')
      call mnopt(2,'Test of Ecl ports')
      call mnopt(3,'Read and write in Mlu')
      call mnopt(4,'Read and write CCR (Camac Control Register)')
      call mnopt(5,'Initialize and check the set up')
      call mndisp
      call mnin(opt_no)
      goto (999,10,10,100,200,300,400,500) opt_no + 3

c
c      Test of programming Mlu
c

```

```

100  continue
      call mneras
      print*, 'Test of Memory with 1010... ($AAAA) and 0101... ($5555)'
      print*, '
      ccr = dim + 16                                ! computes CCR in inhibit mode
      call cfsa(16, mlu_a_2, ccr, q) ! load CCR
      bin_word(1) = 'AAAA'x                        ! hex for 101010...
      bin_word(2) = '5555'x                        ! hex for 010101...
      mem_ok = .true.
      do j=1, 2
        write(*, '(1x, a29, z5, a21, i1, 5a)')
+         '**** Now testing memory with ',
+         bin_word(j), ' and dimensionality : ', dim, ' ****'
        do car=0, 4095
          call cfsa(16, mlu_a_1, car, q) ! writes car
          call cfsa(16, mlu_a_0, bin_word(j), q) ! writes in Mlu
          if (.not. q.and. car.ne. 4095) write(*, *) 'No Q on write Mlu'
        enddo
        call cfsa(16, mlu_a_1, 0, q) ! reset Car
        top = (65536/(2**dim) - 1)
        do car=0, top
          call cfsa(0, mlu_a_0, out_word, q) ! read mlu
          if (.not. q.and. car.ne. top) write(*, *) 'No Q on read Mlu'
          dim_word = 2**dim ! dimension of output word
          if (dim.eq.0) then
            bit_pos = car / 4096 ! compute the position of tested bit
            in_word = jibits(bin_word(j), bit_pos, 1) ! extract the bit
          else
            in_word = jibits(bin_word(j), 0, dim_word)
          endif
          out_word = jibits(out_word, 0, dim_word)

          if (out_word.ne. in_word) then
+            write(*, form1) 'Error at Car:',
+            car, 'Write : ', in_word, ' Read : ', out_word
            mem_ok = .false.
          endif
        enddo
      enddo
      if (mem_ok) then
        write(*, *) 'Memory it''s ok !'
      else
        write(*, *) 'Errors in memory'
      endif
      call dikey('Press any key to continue')
      goto 10

c
c  Test of Ecl ports
c

200  continue
      call mnclr
      call mnhead('Test of MLU Ecl readout')
      call mnopt(1, 'Test with 1010... and 0101...')
      call mnopt(2, 'Sampling test')
      call mnopt(3, 'Cross talk test')
      call mndisp
      call mnin(opt_no)
      goto (999, 10, 200, 210, 220, 230) opt_no+3

210  call mneras
      print*, 'Test of Ecl ports with 1010... ($AAAA) and 0101... ($5555)'
      print*, '
      bin_word(1) = 'AAAA'x                        ! hex for 101010...
      bin_word(2) = '5555'x                        ! hex for 010101...
      ecl_ok = .true.
      do j=1, 2
        write(*, '(1x, a29, z5, a12, i1, 5a)')
+         '**** Now testing ports with ',
+         bin_word(j), ' and mode : ', mode, ' ****'
        ccr = 4 + 2 * 8                                ! computes CCR in inhibit mode
        call cfsa(16, mlu_a_2, ccr, q) ! load CCR
        do car=0, 4095
          call cfsa(16, mlu_a_1, car, q) ! writes car
          call cfsa(16, mlu_a_0, bin_word(j), q) ! writes in Mlu
          if (.not. q.and. car.ne. 4095) write(*, *) 'No Q on write Mlu'
        enddo
        ccr = 4 + mode * 8                                ! computes CCR in mode

```

```

call cfsc(16,mlu_a_2,ccr,q) ! load CCR
do car=0,4095,256
  call cfsc(9,dsa_a_0,dummy,q) ! mrst of dsa
  call cfsc(9,dsb_a_0,dummy,q) ! mrst of dsb
  do addr=car, car+255
    call cfsc(16,dsa_a_0,addr,q) ! write address in dsa
    if(.not.q.and.addr.ne.255)
+      write(*,*)'No Q on writing Data stack A'
  enddo
  call cfsc(26,ck_a_0,dummy,q) ! start clock
  call cfsc(16,dsb_a_1,0,q) ! reset rp
  do addr=car,car+255
    call cfsc(0,dsb_a_0,out_word,q)
    if(.not.q.and.addr.ne.255)
+      write(*,*)'No Q on read Data stack B'
    if(out_word.ne.bin_word(j)) then
+      write(*,form1)'Error at Car:',
      addr,' Write :',bin_word(j),' Read :',out_word
+      ecl_ok = .false.
    endif
  enddo
enddo
enddo
if (ecl_ok) then
  write(*,*)' Ecl ports readout it's ok!'
else
  write(*,*)' Errors on Ecl ports'
endif
call dikey('Press ANY KEY to continue')
goto 200

c
c
c
220 Sampling test
      continue
      call mneras
      write(*, '(1x,a40,a12,i1,5a)')
+      '**** Now testing ports with sampling test',
+      ' and mode : ',mode,' ****'
      ccr = 4 + 2 * 8 ! computes CCR in inhibit mode
      call cfsc(16,mlu_a_2,ccr,q) ! load CCR
      word = 0
      do car=0, 15
        call cfsc(16,mlu_a_1,car,q) ! writes car
        written(car) = word
        call cfsc(16,mlu_a_0,word,q) ! writes in Mlu
        word = word + '1111'x
      enddo
      ccr = 4 + mode * 8 ! computes CCR in mode
      call cfsc(16,mlu_a_2,ccr,q) ! load CCR

      call cfsc(9,dsa_a_0,dummy,q) ! mrst of dsa
      call cfsc(9,dsb_a_0,dummy,q) ! mrst of dsb
      do addr=0, 15
        call cfsc(16,dsa_a_0,addr,q) ! write address in dsa
      enddo
      call cfsc(26,ck_a_0,dummy,q) ! start clock
      call cfsc(16,dsb_a_1,0,q) ! reset rp
      do car=0,15
        call cfsc(0,dsb_a_0,out_word,q)
        if(out_word.ne.written(car)) then
+          write(*,form1)'Error at Car:',
          car,' Write :',written(car),' Read :',out_word
+          ecl_ok = .false.
        endif
        if(all.eq.1.and.(out_word.eq.written(car))) then
+          write(*,form1), ' O.K. at Car:',
          car,' Write :',written(car),' Read :',out_word
+          ecl_ok = .true.
        endif
      enddo
      call dikey
      goto 200

c
c
c
230 Cross talk test
      continue
      call mneras
      bin_word(1) = 1
      bin_word(2) = 65534
      zero(1) = 65535
      zero(2) = 0

```



```

        write(*,'(1x,a45,a12,i1,5a)')
+         '**** Now testing ports with cross talk test',
+         ' and mode : ',mode,' ****'
do j=1,2
    ccr = 4 + 2 * 8           ! computes CCR in inhibit mode
    call cfsa(16,mlu_a_2,ccr,q) ! load CCR
    word = bin_word(j)
    do car=0, 31, 2
        call cfsa(16,mlu_a_1,car,q) ! writes car
        written(car) = zero(j)
        call cfsa(16,mlu_a_0,zero(j),q) ! writes in Mlu
        call cfsa(16,mlu_a_1,car+1,q) ! writes car
        written(car+1) = word
        call cfsa(16,mlu_a_0,word,q) ! writes in Mlu
        word = jshftc(word,1,16) ! shifts word
    enddo
    ccr = 4 + mode * 8       ! computes CCR in mode
    call cfsa(16,mlu_a_2,ccr,q) ! load CCR

    call cfsa(9,dsa_a_0,dummy,q) ! mrst of dsa
    call cfsa(9,dsb_a_0,dummy,q) ! mrst of dsb
    do addr=0, 31
        call cfsa(16,dsa_a_0,addr,q) ! write address in dsa
    enddo
    call cfsa(26,ck_a_0,dummy,q) ! start clock
    call cfsa(16,dsb_a_1,0,q) ! reset rp
    do car=0,31
        call cfsa(0,dsb_a_0,out_word,q)
        if(out_word.ne.written(car)) then
+           write(*,form1)'Error at Car:',
+           car,' Write : ',written(car),' Read : ',out_word
        endif
        if(all.eq.1.and.(out_word.eq.written(car))) then
+           write(*,form1),' O.K. at Car:',
+           car,' Write : ',written(car),' Read : ',out_word
        endif
    enddo
enddo
call dkey
goto 200

c
c
c
300      continue
      call mnclr
      call mnhead('***** Write and read Data in Mlu *****')
      call mnopt(1,'Read data')
      call mnopt(2,'Write data')
      call mndisp
      call mnin(opt_no)
      goto (999,10,300,310,320)opt_no+3

c
c
c
310      continue
      call dldc('No. of data',num_w,num_w)
      call dldc('Starting address ',start_add,start_add)
      ccr = dim + 16
      call cfsa(16,mlu_a_2,ccr,q) ! load ccr
      do add=start_add, start_add+num_w-1
          call cfsa(16,mlu_a_1,add,q)
          call cfsa(0,mlu_a_0,out_word,q)
          if (.not.q)write(*,*)'No Q on read Mlu'
          dim_word = 2**dim
          out_word = jibits(out_word,0,dim_word)
          write(*,form2)'Address (car): ',add,
+           ' Word : ',out_word
      enddo
      call dkey('Press any key to continue')
      goto 300

c
c
c
320      continue
      write(*,*)'***** Be carefull write is performed only with',
+           ' dimensionality of 4 *****'
      print *, ' '
      call dldc('No. of data',num_w,num_w)

```



```

call dldc('Starting address ',start_add,start_add)
ccr = 4 + 16
call cfsa(16,mlu_a_2,ccr,q) ! load ccr
do add=start_add, start_add+num_w-1
  call cfsa(16,mlu_a_1,add,q)
  write(*,form3)'Address:',add,' Data: '
  read(*,form4)in_word
  call cfsa(16,mlu_a_0,in_word,q)
  if(.not.q)write(*,*)'No Q on write Mlu'
enddo
call dkey('Press any key to continue')
goto 300

c
c
c
400
  continue
  call mncir
  call mnhead('***** Handling CCR *****')
  call mnopt(1,'Read Camac Control Register')
  call mnopt(2,'Write Camac Control Register')
  call mndisp
  call mnin(opt_no)
  goto (999,10,400,410,420) opt_no+3

c
c
c
  Read CCR

410
  call cfsa(0,mlu_a_2,out_ccr,q)
  dim1 = jibits(out_ccr,0,3)
  mode1 = jibits(out_ccr,3,2)
  write(*,*)'Mode is :',mode1
  write(*,*)'Dimensionality is :',dim1
  call dkey('Press any key to continue')
  goto 400

c
c
c
  Write CCR

420
  continue
  call dldc('Enter mode option (0-2)',mode1,mode1)
  call dldc('Enter dimensionality option (0-4)',dim1,dim1)
  in_ccr = dim1 + 8*mode1
  call cfsa(16,mlu_a_2,in_ccr,q)
  call dkey('Press any key to continue')
  goto 400

c
c
c
  Initialize set up

500
  continue
  call mncir
  call mnhead('***** Set up menu *****')
  call mndec(1,'Branch',b,b)
  call mndec(2,'Crate',c,c)
  call mndec(3,'Mlu in',mlu,mlu)
  call mndec(4,'Data stack A (mlu input)',dsa,dsa)
  call mndec(5,'Data stack B (mlu output)',dsb,dsb)
  call mndec(6,'Clock generator in',ck,ck)
  call mndec(7,'Clock period (40-190ns)',per,per)
  call mnnopt(8,' ')
  call mndec(9,'Dimensionality of mlu words (0,1,2,3 or 4)',dim,
+    dim)
  call mndec(10,'Mode for ecl readout: strobed(0) transparent(1)',
+    mode,mode)
  call mndec(11,'I/O of data in hex(1) or in decimal(0)',hex,hex)
  call mndec(12,'Print all(1) or only errors(0)',all,all)
  call mnnopt(13,'Set up only for Tau1-2 measurement')
  call mndec(14,'Clock generator 2 in',ck1,ck1)
  call mndec(15,'Delay on out 2 clock 1',delay,delay)
  call mndec(16,'Delay on out 1 clock 2',delay1,delay1)
  call mndisp
  call mnin(opt_no)
  goto (999,510,500,500,500,500,500,500,500,500,500,500,
+    500,500,500,500,500)opt_no+3

c
c
  Camac and module initialization

```

```

c
510      continue
      call iniz_camac(b,c)

      call cdreg(dsa_a_0,b,c,dsa,0)
      call cdreg(dsa_a_1,b,c,dsa,1)
      call cdreg(dsa_a_2,b,c,dsa,2)

      call cdreg(dsb_a_0,b,c,dsb,0)
      call cdreg(dsb_a_1,b,c,dsb,1)
      call cdreg(dsb_a_2,b,c,dsb,2)

      call cdreg(mlu_a_0,b,c,mlu,0)
      call cdreg(mlu_a_1,b,c,mlu,1)
      call cdreg(mlu_a_2,b,c,mlu,2)
      call cdreg(mlu_a_3,b,c,mlu,3)
      call cdreg(mlu_a_4,b,c,mlu,4)

      call cdreg(ck_a_0,b,c,ck,0)
      call inizclock(b,c,ck,per,25,delay,25,25,0)
      call inizclock(b,c,ck1,40,delay1,25,25,25,0)

      if (hex.eq.1) then
         form1 = '(1x,a13,i5,a7,z5,a6,z5,a8)'
         form2 = '(1x,a14,i5,a7,z5)'
         form3 = '(1x,a8,i5,a7,$)'
         form4 = '(z5)'
      else
         form1 = '(1x,a13,i5,a7,i5,a6,i5,a8)'
         form2 = '(1x,a14,i5,a7,i5)'
         form3 = '(1x,a8,i5,a7,$)'
         form4 = '(i5)'
      endif

      goto 10

c
c      stop
c
999      continue
      call parwt
      return
      end

*****
*
*          FERATEST:
*
*          program to test the Fera module (Irs 4300) +
*          the Fera driver module (Irs 4301)
*
*          by P. Pacchiarotti      Jun-86
*
*****

Subroutine FERATEST

  implicit integer (a-z)
  integer fera_a(0:15),mean_ped(0:15),ped_sum(0:15),pedmax(0:15),
+    pedmin(0:15),fera_out(0:15),n(0:15)

  real*8 delta,int_lin(0:15),m,m_max(0:15),m_min(0:15)
+    ,slope(0:15),int_lin_ref,a,b1,stan_dev,sum_x(0:15)
+    ,sum_y(0:15),sum_sqa_x(0:15),sum_sqa_y(0:15),
+    sum_xy(0:15),m1,x1(0:15),x2(0:15),y1(0:15),
+    y2(0:15),m_sum(0:15),delta_m,err,chi_sqa,
+    ped_dev,real_num

  logical q, ped_ok(0:15),ok_slope(0:15),a_test,b_test,
+    ok_int_lin(0:15),eci(0:15),ped_sub(0:15),zero_com(0:15)
+    ,ok_ovrflw(0:15)

  character*8 status,status1,status2

  real pedlow,pedup

  common/hmem(10000)
  common/mncomm/nwrds,mnbuf(500)
  common/oppars/b,c,fera,fd,ds,num_mis,fera_step,no_bit

```

```

data nwrds / 500 /
data pedlow,pedup /0.,100./
c
c reads parameters from oppars
c
call parfil('feratest.las')
call parrd
c
c Top level menu
c
a_test = .false. ! the A test is not executed
b_test = .false. ! also the B
100 continue
call mnclr
call mnhead('***** Test of Fera module : Top menu *****')
call mnopt(1,'Pedestal test + pedestal histogram')
call mnopt(2,'Linearity test')
call mnopt(3,'Monotonicity and Integral linearity test')
call mnopt(4,'Ecl readout test')
call mnopt(5,'Initialize and check the set up')
call mnopt(6,'Stop')
call mndisp
call mnin (opt_no)
goto (999,10,10,100,300,200,400,500,999) opt_no + 3
c
c Pedestal Test + histogramming
c
1000 continue
call hlmit(5000)
c
c booking for hbook
c
do i=1,16
  call hbook1(i,'PEDESTALS CHANNEL #$',100,pedlow,pedup)
enddo
call hbstat(0)
a_test = .true.
call mneras
write(*,*)'***** Pedestal test is running *****'
write(*,*)' '
call cfsc(9,fd_a_0,dummy,q) ! clear fera driver
call cfsc(16,fd_a_0,0,q) ! set trv = 0
c
c reset some variables
c
do i=0,15
  pedmax(i)=0
  pedmin(i)=65535
  ped_sum(i) = 0
  ped_ok(i) = .true.
  sum_sqa_x(i) = 0.
  sum_x(i) = 0.
enddo
do i=1, num_mis
  call cfsc(9,fera_a(0),dummy,q) ! clr fera
  call cfsc(16,fera_a(0),'4000'x,q) ! sets lam and camac readout
  if (.not.q) write(*,*)'No lam setting at fera'
  call cfsc(25,fera_a(0),dummy,q)
  call cfsc(8,fera_a(0),dummy,q)
  if (.not.q) goto 110 ! test l.a.m.
  do chan=0, 15
    call cfsc(2,fera_a(chan),ped,q)
    if (.not.q) write(*,*)' No Q on reading Fera'
    call hfill((chan+1),float(ped))
    pedmax(chan) = jmax0(pedmax(chan),ped)
    pedmin(chan) = jmin0(pedmin(chan),ped)
    ped_sum(chan) = ped_sum(chan) + ped
    sum_sqa_x(chan) = sum_sqa_x(chan) + dfloatj(ped * ped)
    sum_x(chan) = sum_x(chan) + dfloatj(ped)
  enddo
enddo
write(*,*)'Channel #| min ped.|mean ped.| max ped.| r.m.s. |',
+ 'status |'

```

```

+ write(*,*)'-----+',
+
do i=0,15
  mean_ped(i) = ped_sum(i)/num_mis      I compute the mean pedestal
  real_num = dfloat(num_mis)
  ped_dev =sqrt(sum_sqa_x(i)/num_mis-(sum_x(i)/num_mis)**2)

  if (ped_dev.gt.0.5) then
    ped_ok(i) =.false.
  endif
  if (ped_ok(i)) then
    status = 'O.K.'
  else
    status = 'BAD'
  endif

  write(*, '(4(3x,i5,2x),3x,f6.3,5x,a8)') i, pedmin(i), mean_ped(i),
+                                     pedmax(i), ped_dev, status
enddo
call dikey('Press any key to continue')
call diyeno('Do you want to see the histograms',yes,yes)
if (yes.ne.1) goto 10

c
c
c
printing histograms

do i=1,16
  call hprint(i)
  call dikey('press any key to continue')
enddo

goto 10

c
c
c
monotonicity and integral linearity test
200 continue
call mneras
if ((.not.a_test).or.(.not.b_test)) then
  write(*,*)'**** You must first execute the A and B test ****'
  call dikey
  goto 10
endif
+ write(*,*)'***** Now testing monotonicity and Integral',
+ 'linearity *****'
write(*,*)' :
delta_m = 2.5 / (trv_step /2.) I max error for slopes
do i=0, 15
  n(i) = 0
  m_sum(i) = 0.
  m_max(i) = 0.
  m_min(i) = 65535
  ok_slope(i) = .true.
  int_lin(i)=0.
  ok_int_lin(i)=.true.
  ok_ovrflw(i)=.false.
enddo

call cfsa(16,fera_a(0),'4000'x,q) I write status word

c
c
c
now test int. linearity and monotonicity

if (no_bit.eq.10) then
  top = 2100
else
  top = 4000
endif
do trv = 0, top, trv_step
  call cfsa(16,fd_a_0,trv,q) I write trv
  if (trv.eq.0) call dikey
  call cfsa(9,fd_a_0,dummy,q) I clr fera via front panel
  call cfsa(25,fera_a(0),dummy,q) I starts test
210 call cfsa(8,fera_a(0),dummy,q) I test lam
  if (.not.q) goto 210

  do chan =0, 15
    call cfsa(2,fera_a(chan),fera_out(chan),q)
    if (.not.q) write(*,*)' No Q on reading Fera'
    if (fera_out(chan).eq.2047) then
      ok_ovrflw(chan) = .true.
    endif
  enddo
enddo

```



```

        goto 1
    endif
    n(chan) = n(chan) + 1
    y2(chan) = fera_out(chan)
    x2(chan) = trv / 2
    if (trv.ge.1) then
        delta = abs(fera_out(chan) - mean_ped(chan) -
            (trv*slope(chan)/2)) ! int. linearity
+       int_lin_ref = (2. + 0.0025*fera_out(chan)) ! tolerance for int. lin.
        if (delta.gt.int_lin_ref) then
            ok_int_lin(chan) = .false.
            write(*,*)'Error: Int. lin. is:',delta,' on channel'
+       ,chan
            write(*,*)' at adc:',fera_out(chan)
+       ,mean_ped(chan),' at dac:'
+       ,(trv*slope(chan)/2)
        endif
        int_lin(chan) = dmax1(delta,int_lin(chan))
        m1 = (y2(chan) - y1(chan))/(x2(chan) - x1(chan))
        m_max(chan) = dmax1(m1,m_max(chan))
        m_min(chan) = dmin1(m_min(chan),m1)
        m_sum(chan) = m_sum(chan) + m1
        err = abs(m1-slope(chan)) ! comp. the diff. of slope
        if (err.gt.delta_m) then
            ok_slope(chan) = .false.
            write(*,*)'Error : slope is ', m1,' on channel'
+       ,chan
            write(*,*)' at adc:',y2(chan)-mean_ped(chan),
+       ' at dac:',x2(chan)
            print*, 'dac old:',x1(chan), ' adc old:',y1(chan)-
+       mean_ped(chan)
        endif
        x1(chan) = x2(chan)
        y1(chan) = y2(chan)
        continue
    enddo
+   enddo
    print *, 'Ch. | max slope | min slope | mean slope |',
+   ' int. lin. | status | over flow |'
    print *, '-----+-----+-----+-----+',
+   '-----+-----+-----+-----+'
    do chan=0, 15
        if (ok_slope(chan).and.ok_int_lin(chan)) then
            status = 'O.K.'
        else
            status = 'BAD'
        endif
        if (ok_ovrflw(chan)) then
            status1 = 'O.K.'
        else
            status1 = 'BAD'
        endif
        m = m_sum(chan) / (n(chan)) ! comp. the mean slope
        write(*, '(1x,12,5x,f6.3,5x,f6.3,9x,f6.3,9x,f6.3,5x,
+       a8,1x,a8)')
+       chan,
+       m_max(chan),m_min(chan),m,int_lin(chan),
+       status,status1
    enddo
    call dikey('Press any key to continue')
    goto 10
c
c   Linearity Test:
c   perform a straight line fit  $y = a + b \cdot x$ 
c   where X is trv (dac out), Y the adc value
c
300 continue
    b_test = .true.
    call mneras
    if (.not.a_test) then
        write(*,*)'**** You must first execute the A test ****'
        call dikey
        goto 10
    endif
    print*, '***** Now testing linearity *****'
    print*, ' '
    do chan = 0,15
        n(chan) = 0
        sum_x(chan) = 0

```

```

sum_y(chan) = 0
sum_sqa_x(chan) = 0
sum_sqa_y(chan) = 0
sum_x_y(chan) = 0
enddo
call cfsc(16,fera_a(0),'4000'x,q) ! write status word
if (no_bit.eq.10) then
  top = 2100
else
  top = 4000
endif
do trv = 0, top, trv_step
  call cfsc(16,fd_a_0,trv,q) ! write trv
  call cfsc(9,fd_a_0,dummy,q) ! clr fera via front panel
  call cfsc(25,fera_a(0),dummy,q) ! starts test
  call cfsc(8,fera_a(0),dummy,q) ! test lam
  if (.not.q) goto 310
  do chan = 0, 15
    call cfsc(2,fera_a(chan),fera_output,q)
    if (.not.q) write(*,*) 'No Q on reading Fera'
    if (fera_output.eq.2047) then
      goto 2
    endif
    n(chan) = n(chan) + 1
    fera_output = fera_output - mean_ped(chan)
    new_trv = trv/2 ! the dac output is half of the adc input
    sum_x(chan) = new_trv + sum_x(chan)
    sum_y(chan) = fera_output + sum_y(chan)
    sum_sqa_x(chan) = new_trv * new_trv + sum_sqa_x(chan)
    sum_sqa_y(chan) = fera_output*fera_output+sum_sqa_y(chan)
    sum_x_y(chan) = new_trv * fera_output + sum_x_y(chan)
  2 continue
  enddo
enddo
print *, 'Channel # | slope | intercept | chi square | status |'
print *, '-----|-----|-----|-----|-----|'
do i = 0, 15
  c
  c computes intercept
  c
  a = sum_y(i)*sum_sqa_x(i)-sum_x(i)*sum_x_y(i)
  a = a/(n(i) * sum_sqa_x(i) - sum_x(i)*sum_x(i))
  c
  c computes slope
  c
  b1 = (n(i)*sum_x_y(i) -sum_x(i)*sum_y(i))
  b1 = b1/(n(i)*sum_sqa_x(i)-sum_x(i)*sum_x(i))
  slope(i) = b1 ! save the slope for further tests
  c
  c computes chi square
  c
  stan_dev = sum_sqa_y(i)+b1*b1*sum_sqa_x(i)+2*a*b1*sum_x(i)
  stan_dev = stan_dev+n(i)*a*a-2*b1*sum_x_y(i)-2*a*sum_y(i)
  chi_sqa = stan_dev/n(i)

  if (a.gt.2.or.chi_sqa.gt.1) then
    status = 'error'
  else
    status = ' O.K.'
  endif
  write(*, '(1x,i2,10x,f5.3,6x,f6.3,6x,f6.3,6x,a8)')
  + i,b1,a,chi_sqa,status
enddo
call dikey('Press any key to continue')
goto 10

c
c
c
400
c
c Ecl readout test
c
continue
call mneras
if (.not.a_test) then
  write(*,*) '**** You must first execute the A test ****'
  call dikey
  goto 10
endif
print *, '***** Test of Ecl read out *****'
print *, ' '
do j=0,15
  ecl(j) = .true.

```

```

ped_sub(j) = .true.
zero_com(j) = .true.
enddo

do case = 1,3
  call cfsa(9,fera_a(0),dummy,q) | clr fera: busy = 0
  if (case.eq.1) then
    call cfsa(16,fera_a(0),'4400'x,q) | ecl readout
    write(*,*)'***** Now testing Ecl readout *****'
    write(*,*)' '
  endif
  if (case.eq.2) then
    call cfsa(16,fera_a(0),'4500'x,q) | ecl ped. subtr.
    write(*,*)'***** Now testing pedestal subtraction *****'
    write(*,*)' '
    do i=0,15
      call cfsa(17,fera_a(i),mean_ped(i),q) | writes pedestal memory
      if(.not.q)write(*,*)' No Q on writing pedestals at fera'
    enddo
  endif
  if (case.eq.3) then
    call cfsa(16,fera_a(0),'4700'x,q) | ecl zero compr. +
                                     | ped. subtraction
    write(*,*)'***** Now testing zero compression *****'
    write(*,*)' '
  endif

  if (.not.q) write(*,*)' No Q on writing status word at fera'
  do trv = 0,3500,50
    call cfsa(9,fera_a(0),dummy,q) | clr fera
    call cfsa(16,fd_a_0,trv,q)      | load trv to fd
    call cfsa(9,ds_a_0,dummy,q)    | mst data stack
    call cfsa(25,fera_a(0),dummy,q) | start test
    call cfsa(16,ds_a_1,0,q)       | reset rp of d.s.
    do chan=0, 15
      call cfsa(2,fera_a(chan),out_fera,q)
      if (.not.q) write(*,*)'No Q on reading fera'
      if (case.eq.1) then
        call cfsa(0,ds_a_0,out_ds,q)
        if (.not.q) write(*,*)'No Q on reading data stack'
        if (out_fera.ne.out_ds) then
          ecl(chan) = .false.
        endif
      endif
      if (case.eq.2) then
        call cfsa(0,ds_a_0,out_ds,q)
        if (.not.q) write(*,*)'No Q on reading data stack'
        out_fera = out_fera - mean_ped(chan)
        if (out_fera.ne.out_ds) ped_sub(chan) = .false.
      endif
      if (case.eq.3) then
        if (chan.eq.0) call cfsa(0,ds_a_0,header,q) | reads head word
        out_fera = out_fera - mean_ped(chan)
        if (out_fera.ne.0) then
          call cfsa(0,ds_a_0,out_ds,q)
          if (.not.q)write(*,*)'No Q on reading data stack'
          out_fera = out_fera + 2*11* chan | add subadd at fera output
          if ((out_fera.ne.out_ds).or(.not.q)) then
            zero_com(chan) = .false.
          endif
        endif
      endif
    endif
  enddo
enddo
enddo
print*,'Channel | Ecl readout | Ecl pedest. subtr. | Ecl zero'
+ print*,' , compr. |'
+ print*,'-----+-----+-----+-----'
+ print*,'-----+-----+-----+-----'
do chan = 0 ,15
  if (ecl(chan)) then
    status = 'O.K.'
  else
    status = 'Error'
  endif
  if (ped_sub(chan)) then
    status1 = 'O.K.'
  else
    status1 = 'Error'
  endif
endif

```

```

        if (zero_com(chan)) then
            status2 = 'O.K.'
        else
            status2 = 'Error'
        endif

        write(*, '(6x,i2,5x,a8,10x,a8,10x,a8)') chan, status, status1,
+          status2
    enddo
    call dlkey('Press any key to continue')
    goto 10
c
c      Initialization and checking of set up
c
500    continue
    call mnclr
    call mnhead('**** Initialize the set up menu ****')
    call mndec(1, 'Branch', b, b)
    call mndec(2, 'Crate', c, c)
    call mndec(3, 'Fera in', fera, fera)
    call mndec(4, 'Fera driver in', fd, fd)
    call mndec(5, 'Data stack in', ds, ds)
    call mndec(6, 'Number of measures in pedestal test', num_mis, num_mis)
    call mndec(7, 'Trv step in linearity tests', fera_step, fera_step)
    call mndec(8, '10 or 11 bit fera ', no_bit, no_bit)
    call mndisp
    call mnin(opt_no)
    goto (999, 510, 500, 500, 500, 500, 500, 500, 500, 500) opt_no + 3
c
c      Camac and module initialization
c
510    continue
    call inlz_camac(b, c)
    call cdreg(ds_a_0, b, c, ds, 0)
    call cdreg(ds_a_1, b, c, ds, 1)
    call cdreg(ds_a_2, b, c, ds, 2)

    call cdreg(fd_a_0, b, c, fd, 0)

    do i=0, 15
        call cdreg(fera_a(i), b, c, fera, i)
    enddo

    trv_step = fera_step * 2 ! set step for trv of f.d. dac

    goto 10
c
c      stop
c
999    continue
    call parwt
    return
end

```

```

*****
*      ECITEST:                                *
*      program to test the Ecline driver        *
*                                          *
*      by P. Pacchiarotti                     *
*                                          *
*      july 86                                *
*****

```

Subroutine ECLTEST

implicit integer (a-z)

integer bin_word(2), ecl_a(0:1), zero(2)

character*80 form1, form2, form3, form4

logical q, mem_ok, ecl_ok_0, ecl_ok_1


```

common/mncomm/nwrds,mnbuf(500)
common/oppars/b,c,ecl,dsa,dsb,ck,per,dim,mode,delay,delay1

data nwrds/500/

c
c
c      read parameters from ecltest.las
c
c      call parfil('ecltest.las')
c      call parrd
c
c      Top level menu
c
c
10      continue
      call mnclr
      call mnhead('***** Test of Ecline driver module *****')
      call mnopt(1,'Test of programming Ecline driver')
      call mnopt(2,'Test of Ecl ports')
      call mnopt(3,'Read and write in Ecline driver')
      call mnopt(4,'Read and write Control Register')
      call mnopt(5,'Initialize and check the set up')
      call mndisp
      call mnin(opt_no)
      goto (999,10,10,100,200,300,400,500) opt_no + 3

c
c
c      Test of programming ecline driver
c
100      continue
      call mneras
      do bank=0,1
        printe,'Test of Memory with 1010... ($AAAA) and 0101...($5555)'
        printe,' Now testing memory port ',bank
        printe,' '
        call cfsa(9,ecl_a(0),dummy,q) ! reset of ecline
        call cfsa(17,ecl_a(0),0,q) ! set control register addr = 0
        bin_word(1) = 'AAAA'x ! hex for 101010...
        bin_word(2) = '5555'x ! hex for 010101...
        mem_ok = .true.
        do j=1,2
          write(*,'(1x,a29,z5,5a)')
          +      '***** Now testing memory with ',
          +      bin_word(j),' *****'
          do point=0, 255
            call cfsa(16,ecl_a(bank),bin_word(j),q) ! writes in Ecl
            if (.not.q) write(*,*) 'No Q on write ecline'
          enddo
          call cfsa(17,ecl_a(0),0,q) ! reset addrr
          do point=0,255
            call cfsa(1,ecl_a(0),addr,q)
            if (.not.q) write(*,*) 'No Q on reading pointer'
            call cfsa(0,ecl_a(bank),out_word,q) ! read ecline
            if (.not.q) write(*,*) 'No Q on read ecline'
            in_word = bin_word(j)
            if (out_word.ne.in_word) then
              +      write(*,form1)'Error at Address:',
              +      addr,'Write :',in_word,' Read :',out_word
              mem_ok = .false.
            endif
          enddo
        enddo
      enddo
      if (mem_ok) then
        write(*,*) ' Memory it''s ok !'
      else
        write(*,*) ' Errors in memory'
      endif
      call dkey('Press any key to continue')
      goto 10

c
c
c      Test of Ecl ports
c
200      continue
      call mnclr
      call mnhead('***** Test of Ecl readout *****')
      call mnopt(1,'Test of Ecl readout with AAAA$ and 5555$')
      call mnopt(2,'Cross Talk Test of Ecl readout')
      call mnopt(3,'Sequential test of Ecl readout')

```

```

call mndisp
call mnin(opt_no)
goto (10,10,200,210,220,230) opt_no+3

c
c
c
210
continue
call mneras
print*, 'Test of Ecl ports with 1010... ($AAAA) and 0101... ($5555)'
print*, ' Now testing memory port 0 and 1'
print*, ' '
bin_word(1) = 'AAAA'x      ! hex for 101010...
bin_word(2) = '5555'x      ! hex for 010101...
ecl_ok_0 = .true.
ecl_ok_1 = .true.
do j=1,2
  write(*, '(1x,a29,z5,5a)')
+   '**** Now testing ports with ',
+   bin_word(j), ' ****'
  cont_word = 2**10*gate
  call cfsa(17,ecl_a(0),cont_word,q) ! load contr word
  do point=0, 255
    call cfsa(16,ecl_a(0),bin_word(j),q) ! writes in ecline 0
    if (.not.q) write(*,*) 'No Q on write ecline memory 0'
  enddo
  call cfsa(17,ecl_a(0),cont_word,q) ! reset pointer
  do point=0, 255
    call cfsa(16,ecl_a(1),bin_word(j),q) ! writes in ecline 1
    if (.not.q) write(*,*) 'No Q on write ecline memory 1'
  enddo
  call cfsa(26,ecl_a(0),dummy,q) ! connect both mem to ecl ports

  call cfsa(9,dsa_a_0,dummy,q) ! mrst of dsa
  call cfsa(9,dsb_a_0,dummy,q) ! mrst of dsb
  call cfsa(25,ecl_a(0),dummy,q) ! start at ecline
  call cfsa(26,ck_a_0,dummy,q) ! start clock
  call cfsa(16,dsb_a_1,0,q) ! reset rp of A
  call cfsa(16,dsa_a_1,0,q) ! reset rp of B
  do addr=0,255
    call cfsa(0,dsb_a_0,out_wordb,q)
    call cfsa(0,dsa_a_0,out_worda,q)
    if(out_worda.ne.bin_word(j)) then
+      write(*,form1) 'Error at Address:',
+      addr, ' Write :', bin_word(j), ' Read :', out_word
      ecl_ok_0 = .false.
      print *, 'On port no. 0'
    endif
    if(out_wordb.ne.bin_word(j)) then
+      write(*,form1) 'Error at Address:',
+      addr, ' Write :', bin_word(j), ' Read :', out_word
      ecl_ok_1 = .false.
      print *, 'On port no. 1'
    endif
  enddo
enddo
if (ecl_ok_0.and.ecl_ok_1) then
  write(*,*) 'Ecl ports readout it''s ok!'
else
  if (.not.ecl_ok_0) write(*,*) 'Errors on port 0 !'
  if (.not.ecl_ok_1) write(*,*) 'Errors on port 1 !'
endif
call dkey('Press ANY KEY to continue')
goto 200

c
c
c
220
continue
call mneras
print*, '***** Cross talk test of ecl ports *****'
print*, ' Now testing memory port 0 and 1'
print*, ' '
bin_word(1) = 1
bin_word(2) = 65534
zero(1) = 65535
zero(2) = 0
ecl_ok_0 = .true.
ecl_ok_1 = .true.
do j=1,2
  do k=0,1
    in_word = bin_word(j)

```

```

do i=0,15
  write(*,'(1x,a29,z5,5a)')
+   '**** Now testing ports with ',
+   in_word,' ****'
  cont_word = 2**10*gate
  call cfsa(17,ecl_a(0),cont_word,q) ! load contr word
  do point=0, 127
    if (k.eq.0) then
      call cfsa(16,ecl_a(0),in_word,q) ! writes in ecline 0
      if (.not.q) write(*,*)' No Q on write ecline memory 0'
      call cfsa(16,ecl_a(0),zero(j),q) ! writes in ecline 0
      if (.not.q) write(*,*)' No Q on write ecline memory 0'
    else
      call cfsa(16,ecl_a(0),zero(j),q) ! writes in ecline 0
      if (.not.q) write(*,*)' No Q on write ecline memory 0'
      call cfsa(16,ecl_a(0),in_word,q) ! writes in ecline 0
      if (.not.q) write(*,*)' No Q on write ecline memory 0'
    endif
  enddo
  call cfsa(17,ecl_a(0),cont_word,q) ! reset pointer
  do point=0, 127
    if (k.eq.0) then
      call cfsa(16,ecl_a(1),in_word,q) ! writes in ecline 1
      if (.not.q) write(*,*)' No Q on write ecline memory 1'
      call cfsa(16,ecl_a(1),zero(j),q) ! writes in ecline 1
      if (.not.q) write(*,*)' No Q on write ecline memory 1'
    else
      call cfsa(16,ecl_a(1),zero(j),q) ! writes in ecline 1
      if (.not.q) write(*,*)' No Q on write ecline memory 1'
      call cfsa(16,ecl_a(1),in_word,q) ! writes in ecline 1
      if (.not.q) write(*,*)' No Q on write ecline memory 1'
    endif
  enddo
  call cfsa(26,ecl_a(0),dummy,q) ! connect both mem to ecl ports

  call cfsa(9,dsa_a_0,dummy,q) ! mst of dsa
  call cfsa(9,dsb_a_0,dummy,q) ! mst of dsb
  call cfsa(25,ecl_a(0),dummy,q) ! start at ecline
  call cfsa(26,ck_a_0,dummy,q) ! start clock
  call cfsa(16,dsb_a_1,0,q) ! reset rp of A
  call cfsa(16,dsa_a_1,0,q) ! reset rp of B
  do addr=0,255
    call cfsa(0,dsb_a_0,out_wordb,q)
    call cfsa(0,dsa_a_0,out_worda,q)
    if (out_worda.eq.0.or.out_worda.eq.65535) goto 1000
    if(out_worda.ne.in_word) then
      write(*,form1)'Error at Address:',
+      addr,' Write :',in_word,' Read :',out_word
      ecl_ok_0 = .false.
      print *, 'On port no. 0'
    endif
    if(out_wordb.ne.in_word) then
+      write(*,form1)'Error at Address:',
+      addr,' Write :',in_word,' Read :',out_word
      ecl_ok_1 = .false.
      print *, 'On port no. 1'
    endif
  enddo
1000  in_word = jshftc(in_word,1,16)
      enddo
      enddo
      enddo
      if (ecl_ok_0.and.ecl_ok_1) then
        write(*,*)' Ecl ports readout it's ok!'
      else
        if (.not.ecl_ok_0) write(*,*)'Errors on port 0 !'
        if (.not.ecl_ok_1) write(*,*)'Errors on port 1 !'
      endif
      call dkey('Press ANY KEY to continue')
      goto 10
c
c
c
230  continue
      call mneras
      print*, '**** Sequential test by 256 ****'
      print*, ' Now testing memory port 0 and 1'
      print*, ' '
      ecl_ok_0 = .true.
      ecl_ok_1 = .true.

```



```

cont_word = 2*10*gate
call cfsa(17,ecl_a(0),cont_word,q) ! load contr word
do point=0, 65535, 256
  call cfsa(16,ecl_a(0),point,q) ! writes in ecline 0
  if (.not.q) write(*,*) 'No Q on write ecline memory 0'
enddo
call cfsa(17,ecl_a(0),cont_word,q) ! reset pointer
do point=0, 65535, 256
  call cfsa(16,ecl_a(1),point,q) ! writes in ecline 1
  if (.not.q) write(*,*) 'No Q on write ecline memory 1'
enddo
call cfsa(26,ecl_a(0),dummy,q) ! connect both mem to ecl ports

call cfsa(9,dsa_a_0,dummy,q) ! mst of dsa
call cfsa(9,dsb_a_0,dummy,q) ! mst of dsb
call cfsa(25,ecl_a(0),dummy,q) ! start at ecline
call cfsa(26,ck_a_0,dummy,q) ! start clock
call cfsa(16,dsb_a_1,0,q) ! reset rp of A
call cfsa(16,dsa_a_1,0,q) ! reset rp of B
do addr=0,65535, 256
  call cfsa(0,dsb_a_0,out_wordb,q)
  call cfsa(0,dsa_a_0,out_worda,q)
  if(out_worda.ne.addr) then
    write(*,form1)'Error at Address:',
+   addr,' Write :',bin_word(j),' Read :',out_word
    ecl_ok_0 = .false.
    print *, 'On port no. 0'
  endif
  if(out_wordb.ne.addr) then
+   write(*,form1)'Error at Address:',
    addr,' Write :',bin_word(j),' Read :',out_word
    ecl_ok_1 = .false.
    print *, 'On port no. 1'
  endif
enddo
if (ecl_ok_0.and.ecl_ok_1) then
  write(*,*) 'Ecl ports readout it's ok!'
else
  if (.not.ecl_ok_0) write(*,*) 'Errors on port 0 !'
  if (.not.ecl_ok_1) write(*,*) 'Errors on port 1 !'
endif
call dikey('Press ANY KEY to continue')
goto 200

c
c
c
300
  continue
  call mnclr
  call mnhead('***** Write and read Data in Ecline driver *****')
  call mnopt(1,'Read data')
  call mnopt(2,'Write data')
  call mnopt(3,'Connect both memories to ecl port (F26)')
  call mndisp
  call mnin(opt_no)
  goto (999,10,300,310,320,330)opt_no+3

c
c
c
310
  continue
  call dldc('Memory 0 or 1',bank,bank)
  call dldc('No. of data',num_w,num_w)
  call dldc('Starting address ',start_add,start_add)
  reg_buff = ecl_a(bank)
  call cfsa(9,ecl_a(0),dummy,q) ! reset camac functions
  call cfsa(17,reg_buff,start_add,q)
  do add=start_add, start_add+num_w-1
    call cfsa(0,reg_buff,out_word,q)
    if (.not.q) write(*,*) 'No Q on read ecline'
+   write(*,form2)' Address :',add,
    ' Word :',out_word
  enddo
  call dikey('Press any key to continue')
  goto 300

c
c
c
320
  continue
  call dldc('Memory 0 or 1',bank,bank)

```



```

call dldc('No. of data',num_w,num_w)
call dldc('Starting address ',start_add,start_add)
call cfsa(9,ecl_a(0),dummy,q) ! resset camac function
reg_buff = ecl_a(bank)
call cfsa(17,reg_buff,start_add,q)
do add=start_add, start_add+num_w-1
  write(*,form3)'Address:',add,' Data: '
  read(*,form4)in_word
  call cfsa(16,reg_buff,in_word,q)
  if(.not.q)write(*,*)'No Q on Ecline'
enddo
call dkey('Press any key to continue')
goto 300

c
c      connect memories
c
330  call cfsa(26,ecl_a(0),dummy,q)
      if(.not.q)write(*,*)' No Q on F26 A0 on Ecline'
      call dkey('Press any key to continue')
      goto 300

c
c      Read and write CCR
c
400  continue
      call cfsa(1,ecl_a(0),cont_word,q)
      if(.not.q) then
        write(*,*)'No Q on reading Control register'
        call dkey
      endif
      address = jibits(cont_word,0,8)
      int_clock = jibits(cont_word,9,1)
      gated = jibits(cont_word,10,1)
      bit_per = jibits(cont_word,12,4)
      period = 50 * bit_per
      call mncir
      call mnhead('***** Handling Control register *****')
      call mndec(1,'Address',address,address)
      call mndec(2,'Clock Internal(1) or external(0)',int_clock,
+      int_clock)
      call mnyeno(3,'Gated Data ',gated,gated)
      call mndec(4,'Internal clock period',period,period)
      call mndisp
      call mnin(opt_no)
      goto (999,10,400,410,410,410,410) opt_no+3

c
c      Write control register
c
410  bit_per = period / 50
      cont_word1 = address+2**9*int_clock+2**10*gated+2**12*bit_per
      call cfsa(17,ecl_a(0),cont_word1,q)
      if(.not.q) then
        write(*,*)'No Q on writing Control register'
        call dkey
      endif
      goto 400

c
c      initialize set up
c
500  continue
      call mncir
      call mnhead('***** Set up menu *****')
      call mndec(1,'Branch',b,b)
      call mndec(2,'Crate',c,c)
      call mndec(3,'Ecl driver in',ecl,ecl)
      call mndec(4,'Data stack A (ecline output port 0)',dsa,dsa)
      call mndec(5,'Data stack B (ecline output port 1)',dsb,dsb)
      call mndec(6,'Clock generator in',ck,ck)
      call mndec(7,'External Clock period (40-190ns)',per,per)
      call mnnopt(8,' ')
      call mndec(9,'Gated (1) or non gated (0) mode ',gate,
+      gate)
      call mndec(10,'Internal (1) or external (0) clock',
+      inter,inter)
      call mndec(11,'I/O of data in hex(1) or in decimal(0)',hex,hex)
      call mnnopt(12,'Set up for TAU1-2 measurements')
      call mndec(13,'Clock generator 2 in',ck1,ck1)
      call mndec(14,'Delay on out 2 clock 1',delay,delay)

```


Appendice B3:
Programmi di prova
dei blocchi funzionali

DUA0:[STUD.PAC.CAMAC]TGROUP1.FOR;10

```
*****
* 1test : *
* test del group 1 *
* *
*****
```

PROGRAM aTEST

```
implicit integer(a-z)
integer fera1_a(0:15),fera2_a(0:15),out1(0:31),out2(0:15)
+ ,fera3_a(0:15),fera4_a(0:15),out3(0:15),out4(0:15)
+ ,multi_a(0:15)
logical Q
```

call iniz_camac(1,1)

```
do i=0,15
  call cdreg(fera1_a(i),1,1,9,i)
  call cdreg(fera2_a(i),1,1,19,i)
  call cdreg(fera3_a(i),1,1,8,i)
  call cdreg(fera4_a(i),1,1,20,i)
  call cdreg(multi_a(i),1,1,23,i)
enddo
```

```
call cdreg(ck_a_0,1,1,3,0)
call cdreg(dsa_0,1,1,15,0)
call cdreg(dsa_1,1,1,15,1)
call cdreg(dsa_2,1,1,15,2)
call cdreg(mlua_0,1,1,16,0)
call cdreg(mlua_1,1,1,16,1)
call cdreg(mlua_2,1,1,16,2)
call cdreg(mlua_3,1,1,16,3)
call cdreg(mlua_4,1,1,16,4)
```

* programm MLU on STROBED mode but without any corrections

```
*
*
* we want dimension 4 and mode inhibit to write ccr
* 10100 in ccr
*
```

```
call cfsc(16,mlua_2,20,q)
* no corrections
*
```

```
do i=0,2047
  call cfsc(16,mlua_1,i,q) ! writes address of mlu
  call cfsc(16,mlua_0,i,q) ! writes the content in the previous address
enddo
```

```
*
* + 3 % corrections
*
```

```
do i=2048, 4095
  call cfsc(16,mlua_1,i,q) ! writes address of mlu
  j = i-2048
  j = nint(j * 1.03)
  call cfsc(16,mlua_0,j,q) ! writes the content in the previous address
enddo
```

* back in strobed mode

```
*
* call cfsc(16,mlua_2,4,q)
*
* call dlyeno('prg pedestals',si,si)
* if (si.eq.0) goto 100
```

```
*
*
* EXT multi dac to 0
*
```

```
do jj=0,15
  call cfsc(16,multi_a(jj),0,q)
  if (.not.q)write(*,*)' No Q on writing DAC'
enddo
```

```
*
*
* call cfsc(9,fera1_a(0),dummy,q)
* call cfsc(16,fera1_a(0),'4400'x,q)
```



```

if(.not.q) write(*,*)' No Q on wr st. wor.'

call cfsa(9,fera2_a(0),dummy,q)
call cfsa(16,fera2_a(0),'4400'x,q)
if(.not.q) write(*,*)' No Q on wr st. wor.'

call cfsa(9,fera3_a(0),dummy,q)
call cfsa(16,fera3_a(0),'4400'x,q)
if(.not.q) write(*,*)' No Q on wr st. wor.'

call cfsa(9,fera4_a(0),dummy,q)
call cfsa(16,fera4_a(0),'4400'x,q)
if(.not.q) write(*,*)' No Q on wr st. wor.'
call dkey('now reading pedestals remove analog inputs')

call cfsa(9,dsa_0,dummy,q)
call cfsa(26,ck_a_0,dummy,q)
call cfsa(16,dsa_1,0,q)
do i=0,15
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera1_a(i),out1(i),q)
  print *, ' fera a1:', out1(i), ' Data stack:',outds
  if(abs(outds-out1(i)).gt.0) write(*,*)'Error A1'
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera2_a(i),out2(i),q)
  print *, ' fera b1:', out2(i), ' Data stack:',outds
  if(abs(outds-out2(i)).gt.0) write(*,*)'Error B1'
enddo
do i=0,15
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera3_a(i),out3(i),q)
  print *, ' fera a2:', out3(i), ' Data stack:',outds
  if(abs(outds-out3(i)).gt.0) write(*,*)'Error A2'
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera4_a(i),out4(i),q)
  print *, ' fera b2:', out4(i), ' Data stack:',outds
  if(abs(outds-out4(i)).gt.0) write(*,*)'Error B2'
enddo
call cfsa(9,fera1_a(0),dummy,q)
call cfsa(9,fera2_a(0),dummy,q)
call cfsa(9,fera3_a(0),dummy,q)
call cfsa(9,fera4_a(0),dummy,q)
do i=0,15
  call cfsa(17,fera1_a(i),out1(i),q)
  if (.not.q) write(*,*)' no Q on wr ped on 1'
  call cfsa(17,fera2_a(i),out2(i),q)
  if (.not.q) write(*,*)' no Q on wr ped on 2'
  call cfsa(17,fera3_a(i),out3(i),q)
  if (.not.q) write(*,*)' no Q on wr ped on 3'
  call cfsa(17,fera4_a(i),out4(i),q)
  if (.not.q) write(*,*)' no Q on wr ped on 4'
enddo

*
*   external DAC
*
*
100 call dkey('put the analog input')
call dldc('what is ext dac value',vdac,vdac)
do i=0,15
  call cfsa(16,multi_a(i),vdac,q)
  if(.not.q)write(*,*)'No q reponse on wr DAC'
enddo

*
call cfsa(9,fera1_a(0),dummy,q)
call cfsa(9,fera2_a(0),dummy,q)
call cfsa(9,fera3_a(0),dummy,q)
call cfsa(9,fera4_a(0),dummy,q)

call cfsa(16,fera1_a(0),'4D00'x,q)
if(.not.q) write(*,*)' No Q on wr st. wor.'

call cfsa(16,fera2_a(0),'4D00'x,q)
if(.not.q) write(*,*)' No Q on wr st. wor.'

call cfsa(16,fera3_a(0),'4D00'x,q)
if(.not.q) write(*,*)' No Q on wr st. wor.'

call cfsa(16,fera4_a(0),'4D00'x,q)
if(.not.q) write(*,*)' No Q on wr st. wor.'

```

```

call cfsa(9,dsa_0,dummy,q)
call cfsa(26,ck_a_0,dummy,q)
call cfsa(16,dsa_1,0,q)
do i=0,15
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera1_a(i),out1(i),q)
  print *, ' fera a1:', out1(i), ' Data stack:', outds
  if(abs(outds-out1(i)).gt.0) write(*,*)'Error A1'
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera2_a(i),out2(i),q)
  print *, ' fera b1:', out2(i), ' Data stack:', outds
  if(abs(outds-out2(i)).gt.0) write(*,*)'Error B1'
enddo
do i=0,15
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera3_a(i),out3(i),q)
  print *, ' fera a2:', out3(i), ' Data stack:', outds
  if(abs(outds-out3(i)).gt.0) write(*,*)'Error A2'
  call cfsa(0,dsa_0,outds,q)
  call cfsa(2,fera4_a(i),out4(i),q)
  print *, ' fera b2:', out4(i), ' Data stack:', outds
  if(abs(outds-out4(i)).gt.0) write(*,*)'Error B2'
enddo

end

```

DUA0:[STUD.PAC.CAMAC]ELMTEST.FOR;3

```

*****
*
*      Elm_rfi_test :
*                  test delle somme
*                  R fi
*
*****
PROGRAM ELM_RFI_TEST

    implicit integer (a-z)
    logical q
    integer bgo_word(1:64)

*
*      camac initialization
*
    call iniz_camac(1,1)
    call cdreg(ds50_a_0,1,1,8,0)
    call cdreg(ds50_a_1,1,1,8,1)
    call cdreg(ds50_a_2,1,1,8,2)

    call cdreg(ds4_a_0,1,1,14,0)
    call cdreg(ds4_a_1,1,1,14,1)
    call cdreg(ds4_a_2,1,1,14,2)

    call cdreg(alu_a_0,1,1,12,0)
    call cdreg(alu_a_1,1,1,12,1)

    call cdreg(ck_a_0,1,1,3,0)
    call cdreg(ed_a_0,1,1,4,0)

*
*      load ds50 with 0-2047 random words
*
    seed = 1
100 call cfsa(9,ds50_a_0,dummy,q) ! mrst ds 50
    call dlyeno('Random word',rye,rye)
    call dlyeno('Sequential Word',sy,sye)
    do i=1, 64
        if(rye.eq.1) bgo_word(i) = 2048 * ran(seed)
        if (sy.eq.1) bgo_word(i) = i-1
        print *, 'Loading : ', bgo_word(i)
        call cfsa(16,ds50_a_0,bgo_word(i),q)
        if(.not.q)write(*,*)'No Q on write ds50'
    enddo
    call cfsa(16,ds50_a_1,0,q) ! reset rp to 0

    call cfsa(25,alu_a_0,dummy,q) ! enables camac function of alu
    call cfsa(16,alu_a_0,2,q)      ! op. code = 2

*
*      START
*
    call cfsa(25,ed_a_0,dummy,q) ! start to ecline driver
    call cfsa(26,ck_a_0,dummy,q) ! start clock

*
*      Check of results
*
    call cfsa(16,ds4_a_1,0,q) ! rp = 0
    do i=1, 32
        call cfsa(0,ds4_a_0,rfi_word,q)
        write(*, '(2(1x,a4,i2,1x,i5,1x))')
+         'word', i, bgo_word(i), 'word', i+32, bgo_word(i+32)
        if ((bgo_word(i)+bgo_word(i+32)).ne.rfi_word) then
+         write(*, '(30x,a10,i5,a10)')
+         'Result : ', rfi_word, '      Error'
        else
+         write(*, '(30x,a10,i5,a10)')
+         'Result : ', rfi_word, '      OK'
        endif
    enddo
    call dlyeno('another test',yes,yes)
    if (yes.eq.1) then
        call dldc(' Enter seed for random (odd)', seed, seed)
        goto 100
    
```

endif
end

DUA0:[STUD.PAC.CAMAC]CLUSTER_TEST.FOR;22

```

*****
*                                     *
*      Test cluster elm rfi          *
*                                     *
*****
PROGRAM CLUSTER_TEST

implicit integer (a-z)

integer elmrfi(0:32),hadarfi(0:32),hadbrfi(0:32)

character*15 status
character*5 response
logical q

call iniz_camac(1,1)

call cdreg(ds_a_0,1,1,21,0)
call cdreg(ds_a_1,1,1,21,1)
call cdreg(ds_a_2,1,1,21,2)

call cdreg(ds7_a_0,1,1,8,0)
call cdreg(ds7_a_1,1,1,8,1)
call cdreg(ds7_a_2,1,1,8,2)

call cdreg(alu35_a_0,1,1,9,0)
call cdreg(alu35_a_1,1,1,9,1)

call cdreg(mlu25_a_0,1,1,10,0)
call cdreg(mlu25_a_1,1,1,10,1)
call cdreg(mlu25_a_2,1,1,10,2)

call cdreg(ds17_a_0,1,1,12,0)
call cdreg(ds17_a_1,1,1,12,1)
call cdreg(ds17_a_2,1,1,12,2)

call cdreg(alu36_a_0,1,1,13,0)
call cdreg(alu36_a_1,1,1,13,1)

call cdreg(mlu26_a_0,1,1,14,0)
call cdreg(mlu26_a_1,1,1,14,1)
call cdreg(mlu26_a_2,1,1,14,2)

call cdreg(ds18_a_0,1,1,16,0)
call cdreg(ds18_a_1,1,1,16,1)
call cdreg(ds18_a_2,1,1,16,2)

call cdreg(alu37_a_0,1,1,17,0)
call cdreg(alu37_a_1,1,1,17,1)

call cdreg(mlu27_a_0,1,1,18,0)
call cdreg(mlu27_a_1,1,1,18,1)
call cdreg(mlu27_a_2,1,1,18,2)

call cdreg(mlu28_a_0,1,1,20,0)
call cdreg(mlu28_a_1,1,1,20,1)
call cdreg(mlu28_a_2,1,1,20,2)

call cdreg(ck_a_0,1,1,2,0)

call cdreg(ed_a_0,1,1,4,0)

print*, '*****      TEST of R PHI CLUSTER      *****'

call dlyeno('only errors ',error,error)
call dlyeno('Dump mlu's ',yes,yes)
if (yes.eq.0) goto 100

Load DS, Dump MLU's, write op_code

*
*
*      dump mlu 25
*
*      ccr = 4 + 16      ! inhibit mode and dim=4
call cfsa(16,mlu25_a_2,ccr,q)
*

```

```

*      load memory with approx 13bit —> 5 table
*
do i=0, 4095
  word1 = jnint(i/256.)
  word2 = jnint((i+4096)/256.) * 256
  inp_word = word1 + word2
  call cfssa(16,mlu25_a_1,i,q)
  call cfssa(16,mlu25_a_0,inp_word,q)
  if (.not.q) write(*,*)' No Q on write MLU 25'
enddo
call cfssa(16,mlu25_a_2,3,q) ! dim=3 mode 0
*
*      Dump MLU 26 & 27
*
ccr = 4 + 16 ! inhibit mode and dim=4
call cfssa(16,mlu26_a_2,ccr,q)
call cfssa(16,mlu27_a_2,ccr,q)
*
*      load memory with approx 13bit —> 4 table
*      13 bit = 200 GeV —> 4
*
do i=0, 4095
  word = 0
  do j=0, 3*4096, 4096
    if (j.lt.2*4096) then
      word = word + jnint((i+j)/512.)*2*(j*4/4096)
    endif
  enddo
  call cfssa(16,mlu26_a_1,i,q)
  call cfssa(16,mlu27_a_1,i,q)
  call cfssa(16,mlu26_a_0,word,q)
  if (.not.q) write(*,*)' No Q on write MLU 26'
  call cfssa(16,mlu27_a_0,word,q)
  if (.not.q) write(*,*)' No Q on write MLU 27'
enddo
call cfssa(16,mlu26_a_2,2,q) ! dim=2 mode 0
call cfssa(16,mlu27_a_2,2,q) ! idem for 27
*
*      dump MIU 28
*
ccr = 4 + 16 ! inhibit mode and dim=4
call cfssa(16,mlu28_a_2,ccr,q)
*
*      load memory for elmrfl threshold
*      and Had a,b threshold set to zero for TEC and SCIN
*      Mlu has 15 input 2 outputs
*      output meanings:
*      0: No cluster
*      1: only elm cluster
*      2: only had cluster
*      3: had+elm cluster
*
do i=0, 4095
  word = 0
  do j=0, 7*4096, 4096
    elm_t = jibits(i+j,0,5)
    had_a_t = jibits(i+j,5,4)
    had_b_t = jibits(i+j,9,4)
    tec = jibits(i+j,13,1)
    scin = jibits(i+j,14,1)
    if (elm_t.ge.8) then ! elm threshold = 8 —> 100GeV
      elm_clust = 1
    else
      elm_clust = 0
    endif
    if (had_a_t.ge.4.and.had_b_t.ge.4) then ! had a threshold = 4 —> 100 GeV
      had_clust = 1
    else
      had_clust = 0
    endif
    word = word + 2*(j*2/4096)*(elm_clust + had_clust*2)
  enddo
  call cfssa(16,mlu28_a_1,i,q) ! load address
  call cfssa(16,mlu28_a_0,word,q) ! load word
  if(.not.q)write(*,*)' No q on write mlu 28'
enddo

```

```

c      call cfsa(16,mlu28_a_2,1,q) ! mode 1
c      Load ds 7,17,18 with random words
c      sum of words for each subdetector <= 200 GeV
c
x = 1
y = 13
100  call cfsa(9,ds7_a_0,dummy,q)
      call cfsa(9,ds17_a_0,dummy,q)
      call cfsa(9,ds18_a_0,dummy,q)
      sumhada_rfi = 0
      sumhadb_rfi = 0
      sumelm_rfi = 0
      start = jnint(ran(y)*32)
      start1 = jnint(ran(y)*32)
      start2 = jnint(ran(y)*32)

      do j=0, 31
        if (j+start.gt.31) then
          i = j - 32 + start
        else
          i = j + start
        endif
        elmrfi(i)=jnint(ran(x)*(2048))
        sumelm_rfi = sumelm_rfi + elmrfi(i)
        if (sumelm_rfi.ge.8192) then
          sumelm_rfi = sumelm_rfi - elmrfi(i)
          elmrfi(i) = 0
        endif
      enddo
      do j=0, 15
        if (j+start1.gt.15) then
          i = j - 16 + start1
        else
          i = j + start1
        endif
        hadarfi(i)=jnint(ran(x)*(2048))
        sumhada_rfi = sumhada_rfi + hadarfi(i)
        if (sumhada_rfi.ge.8192) then
          sumhada_rfi = sumhada_rfi - hadarfi(i)
          hadarfi(i) = 0
        endif
      enddo
      do j=0, 15
        if (j+start2.gt.15) then
          i = j - 16 + start2
        else
          i = j + start2
        endif
        hadbrfi(i)=jnint(ran(x)*(2048))
        sumhadb_rfi = sumhadb_rfi + hadbrfi(i)
        if (sumhadb_rfi.ge.8192) then
          sumhadb_rfi = sumhadb_rfi - hadbrfi(i)
          hadbrfi(i) = 0
        endif
      enddo
      do i=0,31
        call cfsa(16,ds7_a_0,elmrfi(i),q)
        if(.not.q)write(*,*)'No Q on write DS 7'
      enddo
      do i=0,15
        call cfsa(16,ds17_a_0,hadarfi(i),q)
        if(.not.q)write(*,*)'No Q on write DS 17'
        call cfsa(16,ds18_a_0,hadbrfi(i),q)
        if(.not.q)write(*,*)'No Q on write DS 18'
      enddo
      elmrfi(32) = elmrfi(0)
      hadarfi(16) = hadarfi(0)
      hadbrfi(16) = hadbrfi(0)

c
c      load op code in ALU
c
      call cfsa(26,alu35_a_0,dummy,q)
      call cfsa(16,alu35_a_0,1,q) ! op code is A+B
      if (.not.q) write(*,*)' No Q on write alu's op code'
      call cfsa(26,alu36_a_0,dummy,q)
      call cfsa(16,alu36_a_0,1,q) ! op code is A+B
      if (.not.q) write(*,*)' No Q on write alu's op code'
      call cfsa(26,alu37_a_0,dummy,q)

```

```

call cfsa(16,alu37_a_0,1,q) 1 op code is A+B
if (.not.q) write(*,*) 'No Q on write alu's op code'

c
c
c      start to clock

call cfsa(25,ed_a_0,dummy,q)
call cfsa(9,ds_a_0,dummy,q)
call cfsa(26,ck_a_0,dummy,q)

c
c
c      Now reads results

call cfsa(16,ds_a_1,0,q) 1 reset rp
do j=1, 32
  i = j
  k = (j-1)/2
  call cfsa(0,ds_a_0,result,q) 1 read result
  if(.not.q) write(*,*) 'No Q on read test ds'
  result = jibits(result,0,2)

c
c
c      computes elm results start from fi segment 2

      if (i.gt.31) i=0
      i1 = i
      i2 = i
      if (i.lt.31) then
        elm_fi_sum = elmrfi(i) + elmrfi(i+1)
      endif
      if (i.eq.31) then
        i1 = 31
        i2 = -1
        elm_fi_sum = elmrfi(31) + elmrfi(0)
      endif

c
c
c      computes had A,B results start from fi segment 1

      if (k.gt.15) k=0
      k1 = k
      k2 = k
      if (k.lt.15) then
        had_a_fi_sum = hadarfi(k) + hadarfi(k+1)
        had_b_fi_sum = hadbrfi(k) + hadbrfi(k+1)
      endif
      if (k.eq.15) then
        k1 = 15
        k2 = -1
        had_a_fi_sum = hadarfi(15) + hadarfi(0)
        had_b_fi_sum = hadbrfi(15) + hadbrfi(0)
      endif

c
c
c      makes the approximation

elm_fi_sum = jnint(elm_fi_sum/256.)
had_a_fi_sum = jnint(had_a_fi_sum/512.)
had_b_fi_sum = jnint(had_b_fi_sum/512.)

      if (elm_fi_sum.ge.8) then
        elm_clust = 1
      else
        elm_clust = 0
      endif

      if (had_a_fi_sum.ge.4.and.had_b_fi_sum.ge.4) then
        had_clust = 2
      else
        had_clust = 0
      endif

      clust = had_clust + elm_clust
      goto (1,2,3,4) clust + 1
1      status = 'nothing'
      goto 5
2      status = 'ELM CLUSTER'
      goto 5
3      status = 'HAD CLUSTER'
      goto 5
4      status = 'HAD + ELM cluster'

5      if (result.eq.clust) then
        response = ' o.k.'

```



```

else
    response = ' BAD '
endif
if ((error.eq.1.and.response.eq.'BAD').or.error.eq.0) then
+   write(*,'(1x,a4,i2,a2,i2,a2,i5,4x,a4,i2,a2,i2,a2,
+       2(1x,a2,i5))')
+       'ELM(' ,i1+1,' +',i2+2,'):',elm_fi_sum,
+       'HAD(' ,k1+1,' +',k2+2,'):', 'A:',had_a_fi_sum,'B:',
+       had_b_fi_sum
+       write(*,'(1x,a15,4x,a19,i2,a9,a5)')
+       status,'Hardware result is:',result,' and it's ',
+       response
endif
write(*,*)' '
enddo
call dlyeno('again',yes,yes)
if (yes.eq.1) goto 100
end

```