Università degli studi di Roma

"La sapienza"

Dipartimento di Scienze Matematiche, Fisiche e Naturali

Tesi di Laurea in Fisica

ANALISI ED IMPLEMENTAZIONE ELETTRONICA DEL SIMULATORE DEL SISTEMA DI READ-OUT DELL'ESPERIMENTO NEMO

Relatore interno:

Prof. Antonio Capone

Relatore esterno:

Dott. Piero Vicini

Candidato:

Ottorino Frezza

matricola: 11110147

ANNO ACCADEMICO 2004-2005

TESINE

Prof. P. Rapagnani

SVILUPPO DI REFRIGERATORI PER ANTENNE GRAVITAZIONALI INTERFEROMETRICHE

Prof. M. Capizzi

MISURE DI MASSE EFFICACI E DEL GRADO DI LOCALIZZAZIONE DEI PORTATORI DI CARICA IN COMPOSTI DI $In_xGa_{1-x}As_{1-y}N_y$

RINGRAZIAMENTI

Grazie a mio padre e mia madre, ai miei fratelli e ad Elena per il costante supporto. Grazie ai miei relatori e ai miei compagni di laboratorio per la loro gentilezza e disponibilità. Grazie a Monica, Monia, Sara, Nicola, Marco, Luca, Valerio e a tutti gli amici con cui ho vissuto tante esperienze durante questi anni. Un pensiero a chi non c'è più e mi ha voluto bene...

Indice

Capitolo 1

Introduzione

L'argomento affrontato in questo lavoro è l'analisi e lo sviluppo di un emulatore del sistema di "readout" dei dati provenienti dall'apparato sottomarino nell'esperimento NEMO.

NEMO è un esperimento di astrofisica delle particelle che ha per scopo la rivelazione e la ricostruzione delle traccie dei neutrini astrofisici di alta energia, così da poterne individuare le sorgenti e svelare nuove informazioni sul cosmo. La scelta di osservare neutrini è dovuta alla bassissima interazione che queste particelle hanno con la materia, proprietà che consente loro di attraversare grandi distanze e zone ad alta intensità di materia ed energia. L'elemento centrale dell'esperimento è il telescopio sottomarino, costituito da un reticolo ordinato di fotomoltiplicatori posti in mare alla profondità di più di 3000m. Attualmente la previsione del numero di fotomoltiplicatori utilizzati è di 4096, posizionati su 64 torri verticali a formare un cubo il cui volume è di$\sim 1Km^3$. L'acqua che circonda il rivelatore, oltre a servire da schermo per le particelle di origine atmosferica, svolge il ruolo di materiale su cui i neutrini possono interagire e di mezzo trasparente in cui le particelle cariche relativistiche prodotte

dall'interazione dei neutrini generano luce Cherenkov che può quindi propagarsi fino ai fotomoltiplicatori. I neutrini, interagendo con l'acqua tramite interazione di corrente carica, producono muoni che conservano la stessa direzione di propagazione dei neutrini. I muoni, viaggiando ad una velocità maggiore a quella della luce nell'acqua, producono per effetto Cherenkov una radiazione elettromagnetica il cui fronte d'onda è disposto sulla superficie di un cono con angolo di apertura e velocità noti. Rivelando i fotoni che costituiscono tale radiazione e registrandone il tempo e la posizione è possibile ricostruire la traiettoria dei muoni e quindi dei neutrini di partenza. Per poter ricostruire la traiettoria dei neutrini è necessaria una analisi dei dati ottenuti dal sistema sottomarino, analisi che per ovvi motivi viene fatta a terra, in un laboratorio posto sulla costa, a cui i dati vengono trasmessi dall'apparato sottomarino attraverso un cavo elettro-ottico. I fotoni prodotti per effetto Cherenkov sono però solo una piccola parte dei fotoni che investono un fotomoltiplicatore posizionato in acqua marina, la maggior parte di tali fotoni è prodotta in seguito all'attività di atomi radioattivi e da organismi presenti sott'acqua. La rivelazione di questi fotoni costituisce una fonte cospicua di rumore rispetto ai segnali cercati. La maggior parte dell'informazione trasportata a terra riguarda quindi eventi non desiderati e rende laboriosa l'individuazione e la ricostruzione del segnale cercato attraverso un apposito algoritmo.

Per ridurre la quantità di dati si effettua una selezione sulla base di principi fisici noti. Una prima selezione viene effettuata dall'elettronica di acquisizione dei segnali presente sott'acqua, che elimina tutti quei segnali uscenti dai fotomoltiplicatori con ampiezza inferiore ad una soglia prefissata (se il segnale tipico generato da un singolo fotone ha ampiezza 120mV la soglia è posta a 30mV) . Un

secondo livello di selezione può essere applicato cercando una correlazione spazio-temporale tra i segnali provenienti da fototubi posizionati sulla stessa torre. L'elettronica dedicata a questo compito deve estrarre i segnali che presentino caratteristiche fisiche tali da essere individuati come candidati ad eventi fisici ricercati. In pratica deve generare un "segnale di Trigger" che corrisponde alla rivelazione di un evento candidato. A tale scopo una possibile architettura di ricezione dei dati a terra potrebbe essere costituita da 64 moduli, ognuno dedito all'analisi dei dati provenienti da una singola torre e costituito da schede di "concentrazione" dei segnali collegate tramite un Bus-PCI ad alcune CPU. Questi moduli avrebbero il compito di effettuare il Trigger di torre. In linea di principio questo Trigger può essere generato in due modi:

- Modalità "puro Hardware", lasciando alle CPU il solo compito di controllo delle schede di concentrazione e di interfaccia con gli altri moduli che, tramite una rete interna, conservano i dati acquisiti in un determinato intervallo temporale nell'intorno del segnale di Trigger.
- Modalità "Software", in cui le CPU analizzano il flusso di dati e generano "via Software" il segnale di Trigger.

Per verificare la fattibilità di tale architettura di ricezione, scegliere un'opzione tra le due sopra elencate e per fornire un supporto allo sviluppo dell'algoritmo di Trigger è stato realizzato un emulatore del sistema di "readout" dei dati.

Tale Emulatore è costituito da una scheda PCI su cui è installata una FPGA e una memoria. La FPGA, opportunamente configurata, può simulare il funzionamento della scheda di concentrazione; nella memoria vengono scritti dati rappresentativi dei segnali che

tipicamente vengono acquisiti da fotomoltiplicatori (PMT) messi in acqua marina. Tali dati sono stati ottenuti elaborando misure effettuate con PMT immersi per tempi lunghi in mare. I dati residenti in memoria sono letti dalla scheda di concentrazione con il giusto "timing", inviati dalla scheda sul Bus-PCI e indirizzati in dei buffer di memoria a disposizione dell'utente. L'accesso e l'utilizzo della scheda, montata su un PC dotato di sistema operativo Linux Open Source, è reso possibile tramite l'utilizzo di un driver appositamente scritto. Il lavoro svolto per la mia tesi è stato quello di realizzare il codice di programmazione della FPGA attraverso l'uso del linguaggio VHDL. La successiva applicazione del driver alla scheda ha permesso di effettuare dei primi test sull'efficienza del sistema scheda-CPU, dimostrando la fattibilità dell'architettura proposta considerando che la tecnologia delle piattaforme di calcolo è in rapido sviluppo e permetterà, quindi, in futuro, una sempre maggiore velocità di trasferimento e analisi dei dati.

Il primo capitolo della tesi è dedicato ad illustrare i concetti fondamentali alla base dell'astrofisica dei neutrini con una breve introduzione ai principali esperimenti in corso. Nel secondo capitolo sono descritti i componenti essenziali che costituiscono il telescopio NEMO. Una breve introduzione all'elettronica utilizzata nell'apparato sottomarino, necessaria per meglio comprendere il motivo della realizzazione dell'emulatore, è affrontata nel terzo capitolo. Il quarto è dedicato alla descrizione della proposta di architettura e del "data rate" aspettato nella ricezione a terra. Nel quinto si entra nel dettaglio della composizione e del funzionamento dell'emulatore. Nell'ultimo capitolo, infine, è descritta la prova di simulazione effettuata e i relativi risultati.

Capitolo 2

Rivelazione di Neutrini

2.1 Astrofisica delle particelle

I primi mezzi con cui l'uomo ha esplorato l'universo sono stati i propri occhi. Un grande passo avanti fu fatto con l'introduzione del telescopio, che permise già a Galileo, che fu il primo ad usarlo, l'osservazione di importanti fenomeni del nostro sistema solare come ad esempio i crateri lunari, le macchie solari, i satelliti di Giove e Saturno. Una rivoluzione altrettanto ampia è avvenuta nella seconda metà del secolo scorso, con l'idea di analizzare la radiazione elettromagnetica proveniente dallo spazio non più solo nel range del visibile, ma in tutto lo spettro. Quest'idea ha permesso la scoperta di nuovi fenomeni nell'osservazione del cosmo: l'esistenza di un fondo di radiazione a microonde dovuto, secondo le attuali teorie, al Big Bang, ne è forse l'esempio più rappresentativo.

Ma per quanto sia importante ancora oggi lo studio della radiazione elettromagnetica, i fotoni che compongono tale radiazione non sono in grado di giungere fino a noi da zone remote dell'universo. Questi infatti sono attenuati dall'interazione con la radiazione infra-

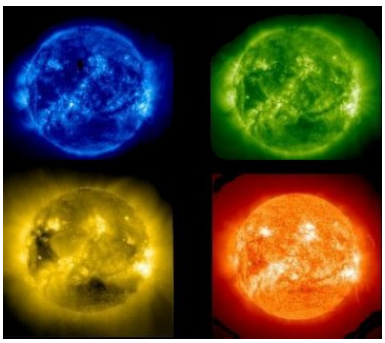


Figura 2.1: Sole osservato a frequenze che vanno dal lontano infrarosso ($\lambda = 10^{-}3m$) all'ultravioletto ($\lambda = 10^{-}10m$)

rossa e con la radiazione cosmica di fondo (effetto Greisen Zatsepin Kuz'min)(ref.(1),(2)) che ne limita il cammino a $\sim 30Mpc$ per fotoni con un energia $\sim 10TeV$, ed a $\sim 10Kpc$ per fotoni con energia $\sim 1000TeV$. Inoltre le regioni più dense e più calde del nostro universo come i nuclei galattici attivi (AGN), i quasar, i sistemi binari di buchi neri sono opache a tali particelle (ref.(3)).

I fotoni non sono comunque l'unica fonte di informazione che arriva dallo spazio. La Terra è "bombardata" da tutta una serie di particelle, alcune provenienti dal sole e dall'atmosfera terrestre, altre invece provenienti dal di fuori del sistema solare. Tutte queste particelle hanno però un loro cammino libero medio caratteristico, oltre il quale sono deviate o assorbite.

La propagazione di protoni nello spazio ad esempio, è limitata debolmente dall'interazione con i campi magnetici galattici e intergalattici. Ma come per i fotoni, l'effetto GZK ne limita il cammino libero medio: protoni con energie $E_p \gtrsim 10^{20}\ eV$ hanno una propagazione che si limita a 30 Mpc.

I neutroni hanno una vita media breve ($\sim 900s$ a riposo), che costituisce un evidente limite alla loro propagazione su lunghe di-

stanze.

I neutrini invece, particelle di massa trascurabile e privi di carica elettrica, interagiscono solo debolmente e per tale motivo possono provenire da zone extragalattiche.

La rivelazione di neutrini potrebbe quindi fornire informazioni sull'origine dei raggi cosmici, sulla natura dei Gamma Ray Burst (GRB) (ref.(4),(5)) e sui motori che alimentano i nuclei Galattici Attivi (AGN). L'esistenza di sorgenti astrofisiche di neutrini di alta energia, ipotizzata teoricamente e confermata da alcune osservazioni (ref.(6)), è legata all'esistenza di siti astrofisici che provocano l'accelerazione dei protoni, i quali, interagendo tra loro e con i fotoni, producono neutrini attraverso la relazione:

$$p + (\gamma, p) => n + \pi + \ldots$$

$$\pi^+ => \mu^+ + \nu_\mu$$

$$\text{con } \mu^+ => e^+ + \overline{\nu_\mu} + \nu_e$$

$$\pi^- => \mu^- + \overline{\nu_\mu}$$

$$\text{con } \mu^- => e^- + \nu_\mu + \overline{\nu_e}$$

dunque il flusso di neutrini è paragonabile, alla sorgente, a quello dei fotoni e degli elettroni, ed ogni neutrino trasporta una frazione dell'energia del protone inizialmente accelerato (ref.(7)). In seguito alla propagazione nello spazio i fotoni possono interagire, il loro spettro si modifica arricchendo le energie più basse. Il flusso di neutrini invece, pur diluendosi nella propagazione dalla sorgente alla Terra, conserva la distribuzione in energia determinata dalla dinamica della sorgente.

2.2 Telescopi per neutrini

La funzione principale dei telescopi per neutrini è la ricostruzione della traiettoria e dell'energia dei neutrini rivelati. Come abbiamo visto, i neutrini sono particelle elusive, che riescono a percorrere spazi intergalattici praticamente illimitati senza essere deviati. Se questo da un lato rende interessante il loro studio, dall'altro ne rende molto difficile la rivelazione.
L'idea alla base degli attuali telescopi per neutrini fu proposta per la prima volta da Markov nel 1960 (ref. (8)). Consiste nell'usare, come materiale sensibile al passaggio dei neutrini, una grande massa di acqua, in modo da avere un numero di interazioni significative tra questa e i neutrini che la attraversano. Il neutrino interagendo con un nucleone produce, tramite una interazione di corrente carica, un leptone:

$$\nu_l + N \Rightarrow l^- + X$$

I leptoni uscenti, rivelati da questo tipo di telescopio, sono i muoni. Questi vengono rivelati indirettamente attraverso la radiazione Cherenkov prodotta.

2.2.1 Radiazione Cherenkov

La produzione di radiazione Cherenkov avviene ogni volta che una particella attraversa un mezzo ad una velocità superiore alla velocità della luce nel mezzo stesso. Questa radiazione viene emessa su un cono con vertice sulla particella e asse lungo la sua traiettoria. L'angolo di apertura del cono dipende dal rapporto tra la velocità della particella e quello della velocità della luce nel mezzo considerato.

Detta v la velocità della luce nel mezzo si ha:

$$v = c/n$$

e l'angolo di emissione della radiazione è:

$$\cos\theta = 1/n\beta \text{ con } \beta = v/c$$

Quindi il muone prodotto dal neutrino crea un cono di luce, che contiene informazioni circa l'energia e la direzione del neutrino che lo ha prodotto.
Per muoni con energia inferiore a $\sim 100GeV$ la lunghezza della traccia è calcolabile considerando che la particella perde in energia $\sim 250Mev/m$ per ionizzazione. Per energie superiori la perdita di energia per Bremsstrahlung diventa dominante e la lunghezza della traccia del muone cresce solo logaritmicamente con la sua energia. In ogni caso ci aspettiamo che il rivelatore sia capace di distinguere neutrini astrofisici, dal fondo dei raggi cosmici di origine astrofisica, per $E_\nu > 1 \div 10Tev$. Per tali energie il muone prodotto si propaga mediamente per distanze di centinaia di metri rendendone possibile la ricostruzione dalla sua direzione di propagazione con grande accuratezza ($\sigma(\vartheta) < 0.3°$).

2.2.2 Progetti avviati

Il primo telescopio di questo tipo fu DUMAND (Deep Underwater And Neutrino Detector) (ref.(9)) che, nato da un progetto statunitense, si prefiggeva di posizionare l'apparato sul fondo dell'oceano Pacifico vicino alle isole Hawaii, alla profondità 4,6Km. Nel 1987 fu posizionata la prima stringa prototipo, contenente i moduli ottici e tutta l'elettronica necessaria, che fu in grado di misurare il flusso di muoni atmosferici solo per poche ore. Questa esperienza fu definiti-

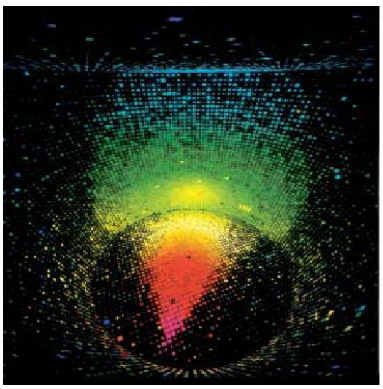


Figura 2.2: Cono di luce Cherenkov prodotto dal passaggio di un muone nell'esperimento Kamiokande

vamente conclusa nel 1996 senza mai riuscire a divenire operativa. Attualmente sono in funzione, o in fase di progettazione, diversi telescopi per neutrini. Tra i più importanti ricordiamo ANTARES, NESTOR, BAIKAL, AMANDA, ICECUBE e NEMO.
(ref.(10),(11),(12),(13))

ANTARES (Astronomy with a Neutrino Telescope and Abyss environmental RESearch) (ref.(14)) e NESTOR (NEutrino Supernova and TeV sources Ocean Range) sono entrambe costruiti nel Mediterraneo, il primo al largo delle coste francesi ed il secondo in acque greche.

BAIKAL è il telescopio che sfrutta invece l'acqua di un lago siberiano, il lago Baikal appunto. I fotorivelatori che devono osservare la radiazione Cherenkov sono posti a $2000m$ di profondità sul fondo del lago.

Il progetto AMANDA(ICECUBE nella futura versione da $1Km^3$), essenzialmente americano, opera(opererà) sotto i ghiacci del polo Sud, rivelando neutrini provenienti dall'emisfero boreale, in questo senso si può pensare come complementare al progetto NEMO che

cercherà invece di rivelare neutrini provenienti dall'emisfero australe, così da completare l'intero spazio osservabile.

NEMO (ref.(15)), infine, è il telescopio di cui mi occuperò in maggior dettaglio nel capitolo seguente, essendo il progetto in cui si inserisce il mio lavoro di tesi.

Capitolo 3

NEMO

3.1 Introduzione

NEMO, acronimo di NEutrinos Mediterranean Observatory, è un esperimento di astrofisica delle particelle che ha come scopo l'individuazione della traiettoria di neutrini di alta energia ($\gtrsim 10^{12} eV$) provenienti da sorgenti extra-galattiche come AGN, Quasar, GRB e Supernovae.

Il telescopio sottomarino NEMO è formato da una serie di fotorivelatori, posizionati nel mare a più di $3000m$ di profondità, secondo una precisa geometria. Il volume totale dell'apparato sarà di circa $1Km^3$. In un apparato sottomarino l'acqua svolge molteplici ruoli: è il bersaglio su cui i neutrini interagiscono, il radiatore in cui i muoni, prodotti dall'interazione di neutrino, producono luce Cherenkov, il mezzo che permette la propagazione del segnale luminoso e lo schermo capace di ridurre il flusso della radiazione di fondo di origine atmosferica. Riducendo tale fondo si riduce la probabilità che eventi di muoni atmosferici, provenienti dall'alto del rivelatore, siano mal ricostruiti e identificati come tracce di neutri-

ni extra-galattici. L'identificazione univoca dei muoni prodotti dalle interazioni dei neutrini cercati è quindi possibile selezionando i muoni provenienti da sotto l'apparato, poichè questi sono sicuramente il prodotto di una interazione tra neutrini e materia avvenuta in prossimità del rivelatore, in virtù delle proprietà di schermo offerte dalla Terra contro tutte le altre particelle. Dunque i neutrini rivelati sono quelli provenienti dall'emisfero australe. I rivelatori colpiti dai fotoni prodotti per effetto Cherenkov, raccolgono così una informazione sull'energia, la posizione e la direzione di volo dei muoni. La sequenza dei tempi di arrivo dei fotoni Cherenkov sui vari fotorivelatori è infatti funzione univoca della direzione di propagazione dei muoni. I muoni, a loro volta, conservano l'informazione della direzione di provenienza dei neutrini: ricostruire la loro traccia nello spazio è lo strumento per realizzare l'astronomia con neutrini. L'informazione è tradotta in segnale elettronico dall'elettronica di lettura dei fotomoltiplicatori, segnale che poi è inviato ad un laboratorio posto sulla costa. Tramite queste informazioni sarà possibile ricostruire a terra, tramite un hardware e un software specifico, la traiettoria e l'energia dei muoni che hanno prodotto il cono di luce.

Per l'esperienza NEMO è stata appena completata la prima fase, denominata R&D, durante la quale sono state effettuate misure riguardanti le proprietà del posto e studi riguardanti la fattibilità del progetto. Attualmente è in corso NEMO FASE 1, che prevede la progettazione e l'installazione, di fronte al porto di Catania a 2000 m di profondità, di un prototipo composto da due delle 64 torri di NEMO che saranno utilizzate nel progetto finale. Inoltre NEMO FASE 1, prevede il posizionamento di una stazione sottomarina per studi geologici il cui nome è GEOSTAR. É infatti interessante sottolineare la collaborazione con tutti i principali enti di ricerca italiani del settore

marino: INFN (Rm, Ct), OGS, SZN, CNR (IAMC-Me, ISMAR-Ts, ISMAR-Ve, IBF-Pi), CONISMA (IUN, DBAEM), ENEA (CRAM). Tale collaborazione si propone di usare NEMO come stazione per lo studio della variabilità di base e le tendenze evolutive, l'analisi e la sintesi dei processi ecologici nel bacino Occidentale dello Ionio, essendo il sito di NEMO, come vedremo in seguito, una postazione ottimale anche per tali scopi.

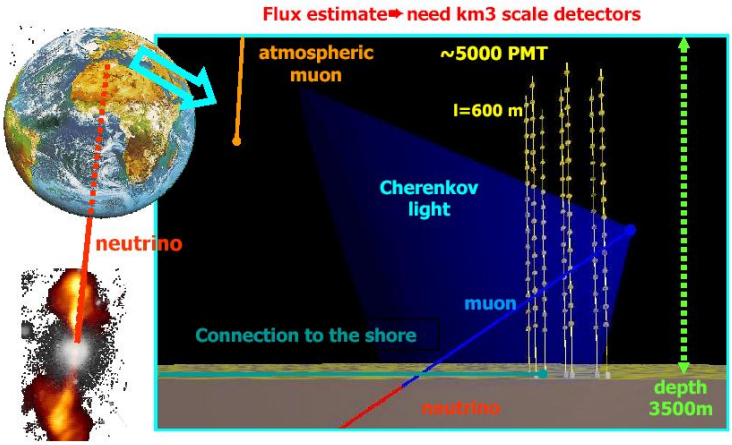


Figura 3.1: Telescopio NEMO

3.2 Sito e sue proprietà

La scelta del sito di NEMO dipende da vari parametri.

Il primo criterio di scelta è la profondità del sito. Questa condizione è essenziale affinchè gli scintillatori siano schermati dalle particelle atmosferiche, che nell'interazione con l'acqua produrrebbero un rumore ottico molto più grande del segnale cercato. Alla profondità di $3000m$, dove NEMO sarà collocato, il rumore prodotto da queste particelle viene ridotto fino a cinque ordini di grandezza.

Le proprietà dell'acqua nel sito di installazione devono inoltre

mostrare valori della lunghezza di attenuazione della luce (nell'intervallo di lunghezze d'onda $320 \div 500nm$) prossimi a quelli dell'acqua pura, così da facilitare la propagazione dei fotoni. I valori ottenuti, documento di una campagna di misure in diversi siti sottomarini, sono visualizzati in figura (2.2) (ref.(16)).

Un altro criterio importante per la scelta del luogo è l'intensità della corrente sottomarina, che deve essere ridotta così da evitare danni all'apparato e spostamenti dei fotorivelatori tali da provocare errori sistematici nell'individuazione della loro posizione; una corrente marina elevata, inoltre, aumenta l'attività di bioluminescenza, anch'essa capace di produrre rumore ottico.

Il sito non deve essere inoltre lontano dalla costa, questo perchè il telescopio è collegato a terra tramite un cavo elettro-ottico il cui costo aumenta in maniera proporzionale alla sua lunghezza. Inoltre il segnale per essere trasportato su lunghezze superiori ai $120Km$ dovrebbe essere amplificato.

Un fattore da considerare è anche la bioluminescenza, prodotta da organismi viventi (batteri, molluschi) e dal decadimento di atomi radioattivi (ad esempio ^{40}K). La bioluminescenza diventa trascurabile al di sotto di 2000 m di profondità, dove la fonte di rumore predominante è il decadimento del ^{40}K.

Il luogo scelto per il posizionamento del futuro rivelatore è situato in Sicilia, al largo delle coste di Capo Passero, 36°16'N;16°06'E, dove sono state effettuate misure di salinità, di temperatura e proprietà ottiche dell'acqua, al variare della profondità e delle stagioni.

Le misure effettuate mostrano, nel sito scelto, che:

- La lunghezza di assorbimento della luce ($\sim 70m@440nm$) è compatibile con i valori dell'acqua pura.

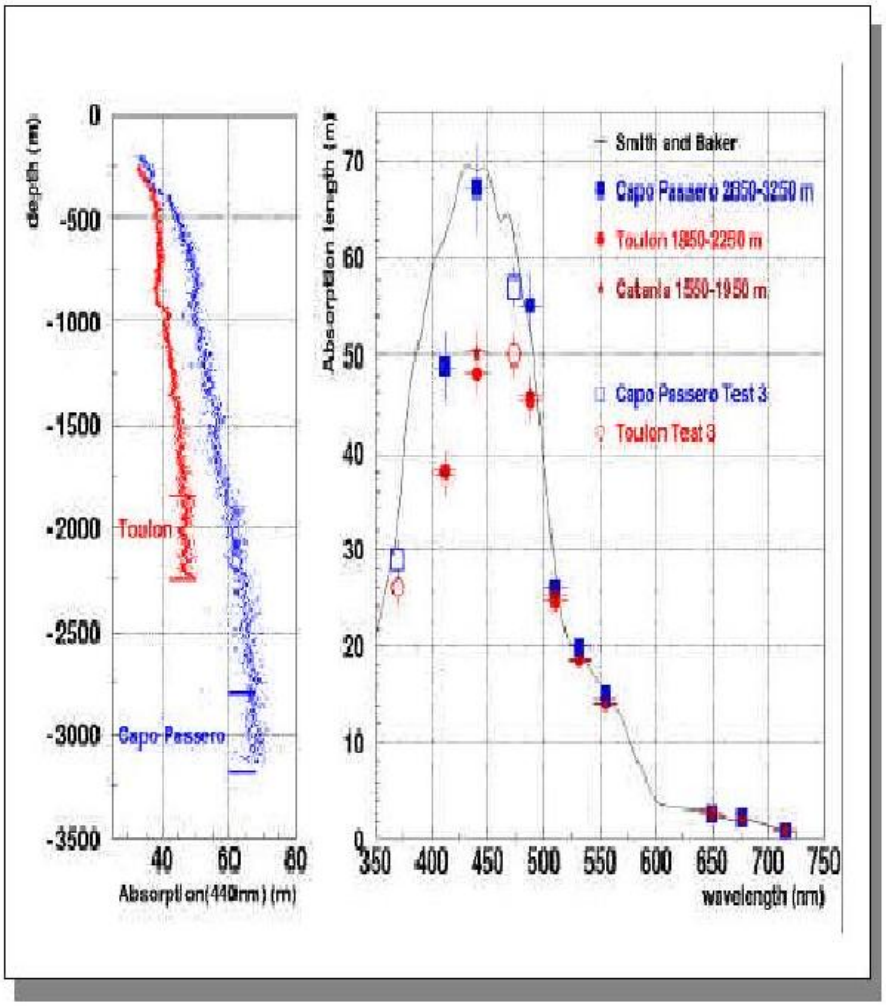


Figura 3.2: Lunghezza di attenuazione e assorbimento della luce misurate nel sito di NEMO (Capopassero) e nel sito di ANTARES e nel sito di Catania.

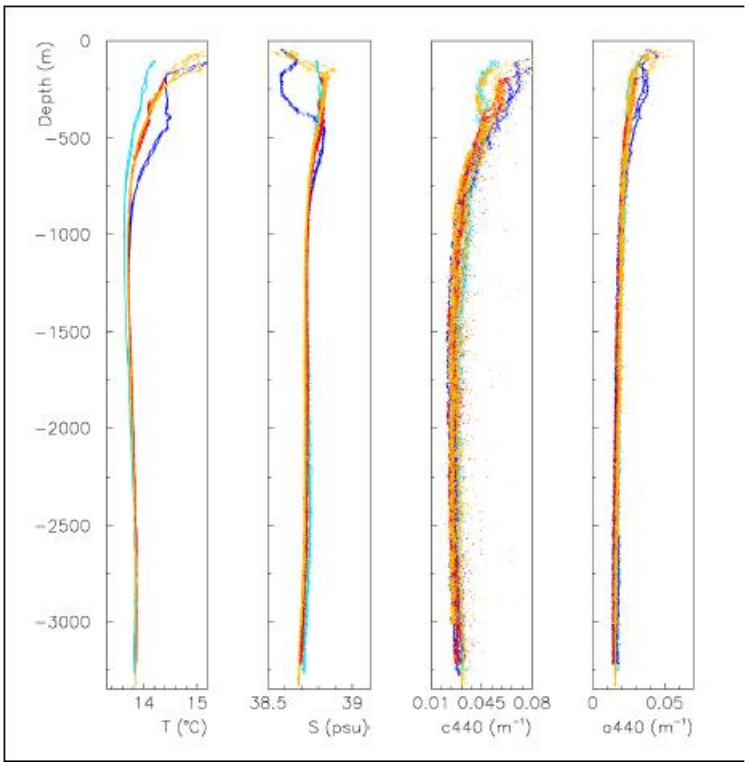


Figura 3.3: Dipendenza stagionale delle proprietà oceanografiche e ottiche al variare della profondità nel sito di Capo Passero. Agosto 02;marzo 02; maggio 02; dicembre 99;

- I valori delle misure effettuate sono stabili nel corso degli anni, una condizione essenziale, essendo dieci anni la durata prevista dell'esperienza.
- Il fondo di rumore ottico è basso, consistente essenzialmente nel decadimento del ^{40}K con rari segnali di bioluminescenza.
- Il sito è vicino alla costa, il fondale è piatto e lontano da canyon e il luogo è lontano da fiumi importanti.
- La corrente marina è bassa ($2-3cm/s$ corrente media; sempre inferiore a $12cm/s$).
- Il tasso di sedimentazione è basso ($\sim 60mg/m^2day$).

3.3 Descrizione dell'apparato

NEMO sarà essenzialmente costituito da una stazione situata sulla costa collegata tramite un cavo elettro-ottico all'apparato sottomarino.

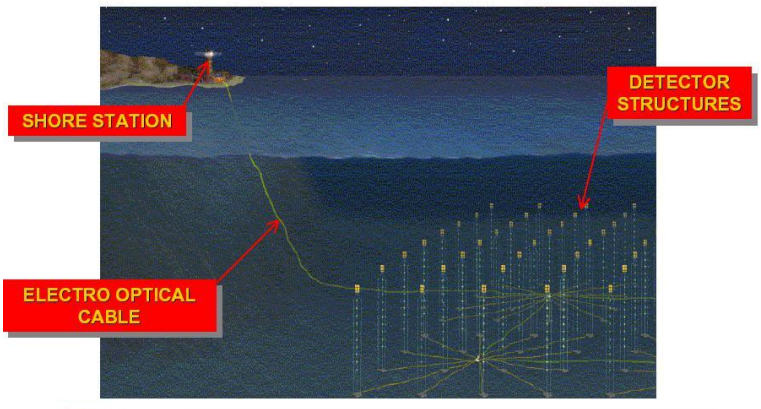


Figura 3.4: Schema dall'apparato

Nella stazione a terra dove avviene l'immagazzinamento e l'elabo-

razione dei dati, sarà presente un'elettronica adatta a questo scopo, oltre ad una serie di calcolatori che analizzeranno i dati provenienti dal mare.

Il cavo elettro-ottico ha più funzioni:

1. Deve fornire la potenza necessaria al funzionamento di tutta la strumentazione sottomarina.

2. Deve trasportare i risultati dai rivelatori al laboratorio a terra.

3. Deve inviare e ricevere i segnali di controllo di tutto il sistema.

Infine la parte subacquea formata da 64 torri disposte a formare un reticolo quadrato di 1 km per ogni lato. La loro altezza dal fondale marino è di $750m$ e sono distanziate una dall'altra di circa 200 m. Ogni torre possiede 64 fotorivelatori, per un insieme di 4096 fotorivelatori.

3.3.1 Le torri

Le torri vengono ancorate sul fondale e tenute verticali, in tensione, tramite una boa di galleggiamento, che nel momento dell'installazione ne consente anche la corretta apertura (vedi fig.2.5). Ad ogni torre sono fissati 16 bracci in posizione orizzontale e lunghi 20 m. Ognuno di questi è costituito da una struttura metallica rigida e dotato di 4 moduli ottici (OM) disposti a due a due alle estremità. Al centro di ogni braccio è presente una scatola con l'elettronica necessaria al controllo dei quattro OM.

La distanza tra il fondale e il primo braccio è di 150 m, gli altri bracci sono distanziati verticalmente tra loro di 40 m. Ogni torre alla sua base ha una "Tower Box"(TB) che raccoglie i dati provenienti dalle 16 scatole elettroniche di ciascun piano.

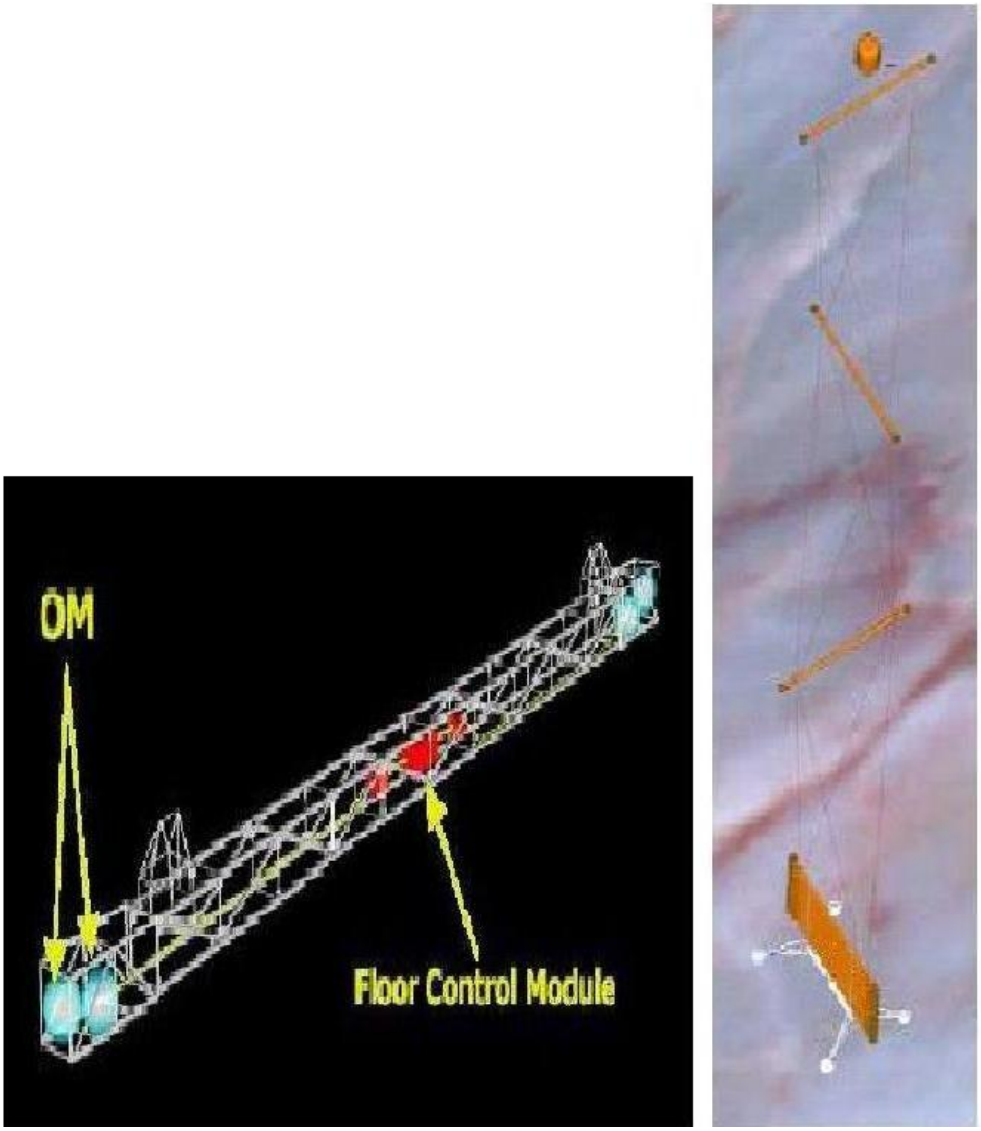


Figura 3.5: Braccio della torre e torre durante la fase di apertura spinta dalla boa

3.3.2 Moduli ottici

I moduli ottici di NEMO sono alloggiati in opportune sfere, dette BENTHOS sfere, che possono resistere a pressioni fino a $\sim 600bar$. All'interno di ognuna di queste sfere è presente un fotomoltiplicatore e l'elettronica necessaria per l'alimentazione, per la lettura dei dati e la loro conversione analogica-digitale.

La sfera è costruita in borosilicato, ha un diametro di circa 40 cm e uno spessore di circa 15 mm. Ad una metà della sfera viene fissato tramite del silicone trasparente il fotomoltiplicatore, l'altra metà contiene l'elettronica. La parte della sfera contenente l'elettronica è schermata per far incidere sul fotomoltiplicatore solo i fotoni provenienti dal fondale marino, così da rendere rivelabili solo i muoni generati dai neutrini provenienti dall'emisfero australe, per le ragioni riportate in precedenza.

3.3.3 Layout delle torri

Le torri sono disposte a formare un reticolo quadrato di $1Km^3$ di lato, la distanza tra una torre e la sua più vicina è di $\sim 180m$. Ogni torre ha alla sua base una TB in cui vengono raccolti tutti i dati provenienti dai 16 piani della torre stessa.

L'esperimento prevede che tutti i collegamenti vengano fatti sul fondale, sia quelli per la distribuzione di potenza che quelli per lo slow control e il trasporto dati. É presente una Main Junction Box (MJB) collegata alla stazione a terra tramite un cavo elettro ottico. La potenza è trasmessa attraverso il cavo in corrente alternata, ad un voltaggio più alto del necessario, al fine di limitare le dispersioni.

Nella MJB avviene un prima riduzione di tale potenza($\sim 30kW$) che poi viene trasferita alle otto Secondary Junction Box (SJB).

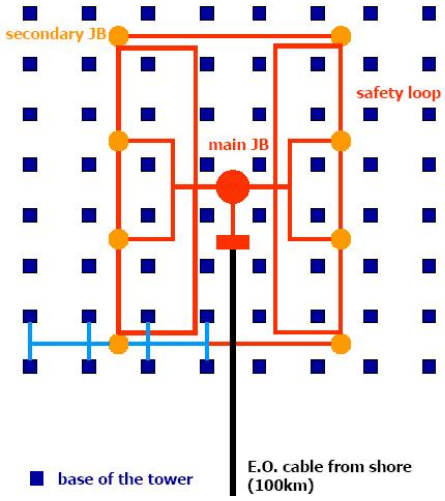


Figura 3.6: Disposizione torri, JB e cavi di collegamento

Ogni SJB è collegata sia con la MJB che con otto torri. Inoltre la SJB è collegata con la SJB successiva e precedente, così da formare un anello che potrebbe essere usato come linea di emergenza nel caso in cui si guastasse un componente.

3.3.4 Junction Box

Le Junction Box sono probabilmente il componente più critico dell'apparato. Esse contengono tutti i collegamenti elettrici tra le varie torri, e tra le torri e la stazione a terra. Una loro rottura, infatti, potrebbe implicare il black-out di grande parte o addirittura di tutto il telescopio. Devono inoltre essere costruite in modo opportuno, in modo tale da poter operare per $\sim$ 10 anni senza interruzioni, la loro manutenzione infatti potrebbe essere molto costosa. Queste considerazioni hanno fatto sì che negli altri esperimenti simili (es. ANTARES) fosse fatto un largo impiego di materiali molto resistenti, anche se molto costosi, come il titanio. Le grandi dimensioni del

telescopio NEMO e quindi il suo alto numero di JB, rendono troppo costosa questa scelta.

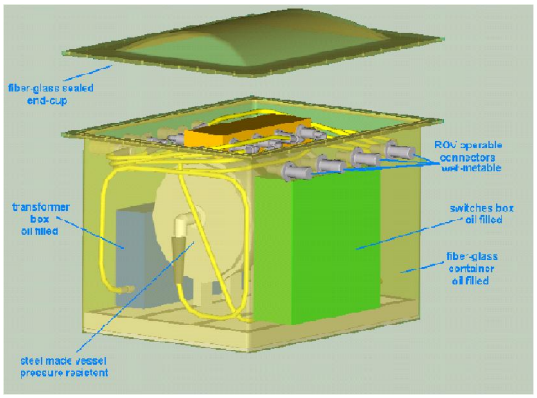


Figura 3.7: Ricostruzione di una Junction Box

Per NEMO, quindi, è stata progettata una JB costituita da una scatola interna fatta di acciaio, posta in un contenitore di fibra di vetro pressurizzato riempito con dell'olio in modo da impedire qualsiasi contatto tra l'acqua marina e l'acciaio. Il contenitore in acciaio garantisce la tenuta meccanica (la resistenza alla pressione esterna trasmessagli dall'olio) mentre il contenitore in fibra di vetro garantisce la resistenza agli effetti corrosivi dell'acqua marina. Questo schema è equivalente per tutte le JB presenti sottacqua, essendo queste simili tra loro.

Capitolo 4

Cenni sull'elettronica di NEMO

4.1 Introduzione

I componenti elettronici utilizzati in NEMO sono molti e vanno dai sistemi per l'acquisizione dei dati a quelli di controllo e alimentazione dell'apparato. Cercherò di descrivere i componenti essenziali per rendere più comprensibile l'ambito della mia attività, dividendo la descrizione tra l'elettronica presente sott'acqua e quella sulla costa che tratterò nel capitolo successivo.

4.2 Sistema sincrono

Una preliminare osservazione è che NEMO è un sistema sincrono, ogni componente elettronico funziona sulla base di un clock comune che, generato a terra, è poi inviato all'apparato sottomarino e distribuito insieme ai segnali di "slow control" con le modalità mostrate nel resto del capitolo. Dato che questo clock, tra l'altro, regola il funzionamento dei fotomoltiplicatori e scandisce il tempo nell'apparato

sottomarino, è indispensabile che giunga ad ogni modulo ottico sincronizzato così da evitare errori sistematici nella misura del tempo di avvenimento di ogni evento. Gli "offset" per il clock derivano dai diversi tempi di propagazione nelle Junction Box del segnale inviato da terra e dalle eventuali differenze di cammino che tale segnale deve affrontare. La calibrazione temporale del telescopio è dunque un problema fondamentale per il funzionamento del sistema che però non è affrontato in questo scritto, comunque, per maggiori dettagli rimandiamo a ref.(17)

4.3 Flusso di dati

Prima di descrivere i componenti elettronici sono necessarie alcune parole sulla quantità di dati che questi componenti trattano. Il telescopio sarà composto di 4096 fotorivelatori e per ognuno di questi si prevede un numero medio di eventi, ossia rivelazione di fotoni, non superiore a $50 Keventi/sec$. Ogni evento (hit) è campionato ad una frequenza di $200 MHz$, e mediamente è composto da $\sim$100 bit di acquisizione. Ogni piano della torre produce quindi:

$$4 * 50 Kevent/s * 100 bit = 20 Mbit/s.$$

Considerando che ogni torre è composta da 16 piani, il numero di dati che arriva alla Tower Junction Box è di $320 Mbit/sec$. Alla Secondary Junction Box arrivano $2.56 Gbit/sec$ e alla MainJB, da cui i dati sono poi inviati a terra, arrivano $20.48 Gbit/sec$.

Il campionamento e la digitalizzazione avvengono all'interno di ogni singola sfera BENTHOS. Quando i dati giungono alla TJB vengono trasformati tramite un Transceiver in segnali ottici, così da poter essere trasportati alle SJB tramite delle fibre ottiche. I segna-

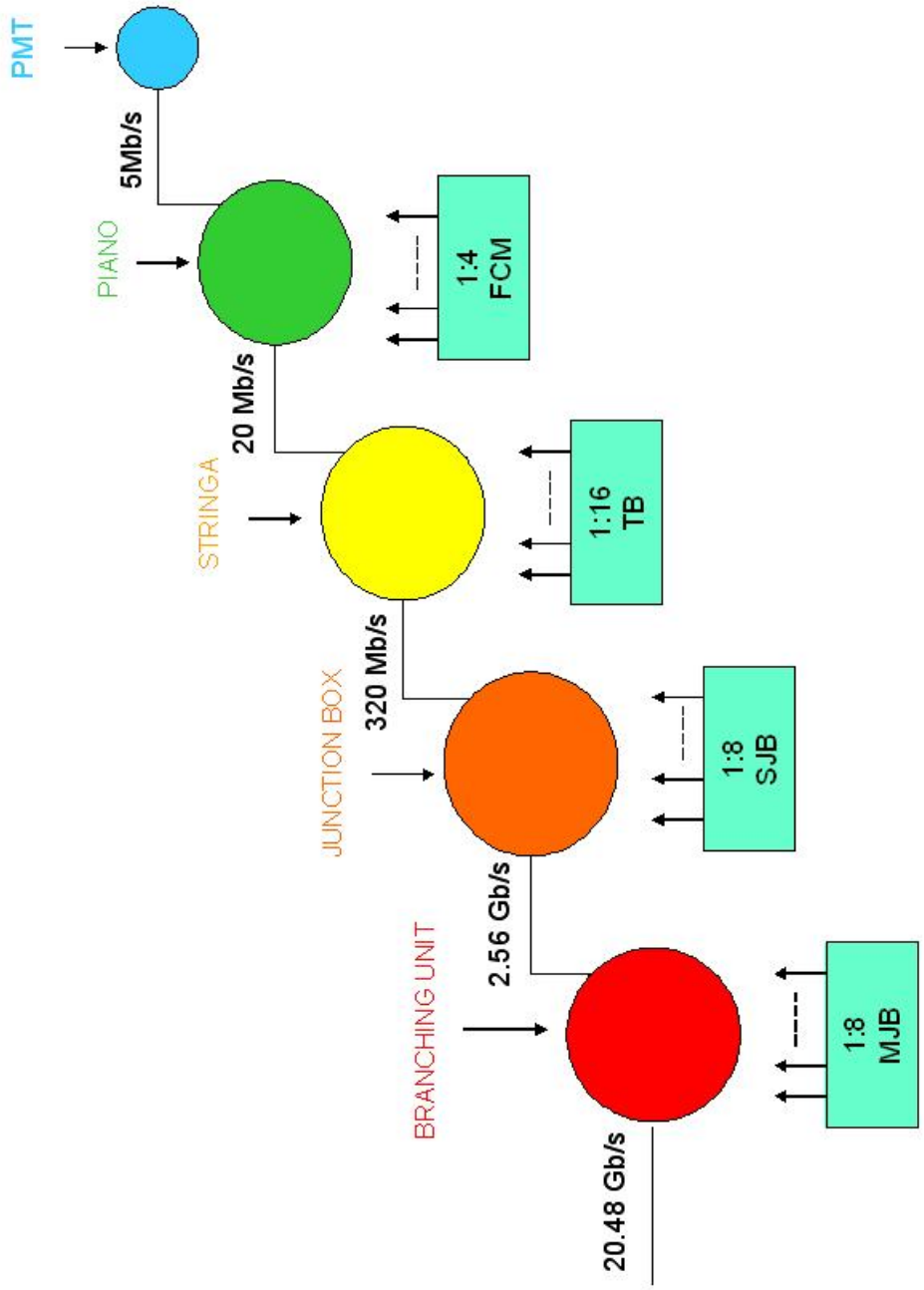


Figura 4.1: Data flow

li ottici sono "formattati" secondo il protocollo SDH (Synchronous Digital Hierarchy), che utilizza moduli STM-1 (Synchronous Transport Module) distanziati tra loro di $125\mu s$, per un "rate" complessivo di dati pari a $155 Mbit/sec$ che escono da ogni torre diretti al laboratorio presente sulla costa.

A questa quantità di dati bisogna aggiungere quelli necessari al controllo del sistema, che viaggiano da e verso le torri e hanno un ordine di grandezza di alcune centinaia di $Kbit/sec$.

4.4 Elettronica di torre

4.4.1 DAQ

All'interno dei moduli ottici è contenuta una scheda di acquisizione dati (DAQ) che ha lo scopo di digitalizzare il segnale analogico uscente dal fotomoltiplicatore. Questa scheda è costituita da più componenti:

1. AFE (Analog Front End). È il componente che raccoglie il segnale analogico uscente dal fotomoltiplicatore. Il suo compito è di adattare l'impedenza in uscita dal fotomoltiplicatore con quella presente in ingresso alla scheda, di comprimere il segnale prima di inviarlo al convertitore analogico digitale. La compressione è logaritmica e avviene tramite l'uso di un diodo, le cui caratteristiche ci aspettiamo che dipendano dalla temperatura; per rimuovere l'incertezza sulla caratteristica del circuito è presente sulla scheda un termometro che permette la calibrazione automatica del compressore di segnale al variare della temperatura.

2. ADC (Analog Digital Converter). É costituito da due converti-

tori di tipo "Flash" a 8 Bit, che operano con clock di frequenza 100MHz. Mentre un convertitore opera sul fronte di clock l'altro convertitore opera sul fronte negato (not(clock)) così da rendere la velocità di campionamento pari a 200MHz.

3. Analog, Digital I/O. Sono 8 canali di ingresso di tipo ADC, e otto canali di uscita di tipo DAC, oltre ad otto canali di ingresso digitali. Servono per la calibrazione iniziale di alcuni componenti interni alla scheda.

4. DSP (Digital Signal Processing). All'attivazione della scheda oltre ad agire come "configuration device" per la FPGA il DSP genera il clock a 100MHz utilizzato dal convertitore ADC, grazie alla presenza di un PLL (Phase Locked Loop) interno. Tramite il DSP, inoltre, è possibile fare un debug della scheda, attraverso opportune porte di comunicazione (JTAG,RS232).

5. MODEM (MODulatore DEModulatore). Connette la scheda ai cavi elettrici che collegano i moduli ottici al modulo di controllo di piano. Oltre a ricevere i dati dalla FPGA fornisce alla scheda i dati di "slow control" e la potenza necessaria al funzionamento sia della scheda che dei fotomoltiplicatori (5VDC).

6. FPGA (Field Programmable Gate Array). Questo è il componente centrale della scheda. Riceve 200MBps dall'ADC, scambia con il DSP 1MBps e dal MODEM riceve circa 20Mbps inviandogli 432 Kbps di segnale di controllo. I dati provenienti dal convertitore vengono prima posti in una FIFO interna alla FPGA, poi insieme ai segnali di "slow control" provenienti dal DSP e a quelli del tempo di rivelazione, sono impacchettati secondo un preciso formato e spediti al MODEM. Il tempo di

rivelazione, chiamato anche "tempo assoluto", è scandito da un orologio interno alla FPGA che funziona con lo stesso clock con cui funziona il convertitore. Un'altra importante funzione di questo componente è quella di "trigger" di livello zero: tutti i segnali rivelati dal fotomoltiplicatore che in ampiezza non superano una certa soglia (30 mV) sono automaticamente eliminati, ossia non vengono riportati nei dati in uscita.

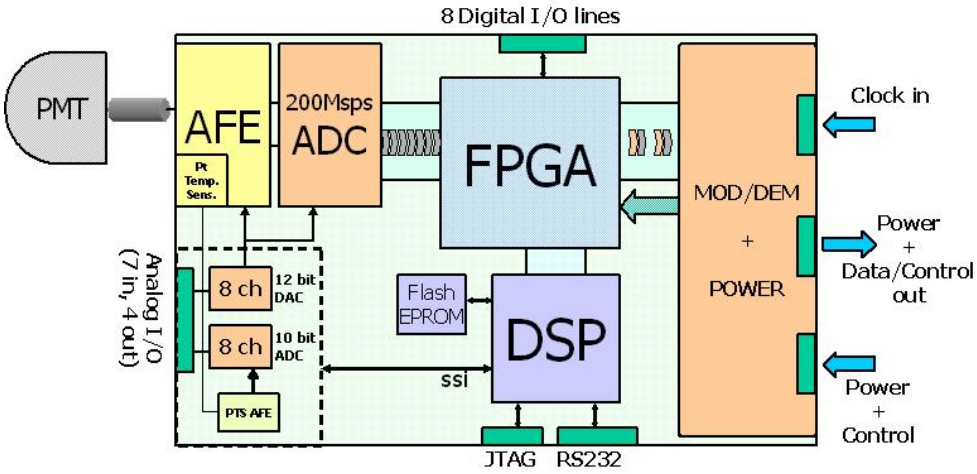


Figura 4.2: Diagramma della scheda DAQ

4.4.2 Fluttuazioni nelle misure di tempo

Le fluttuazioni in questo tipo di misure dipendono da diverse cause. Una causa ineliminabile è rappresentata dai diversi tempi caratteristici dei fotorivelatori, per il modello scelto in NEMO questa incertezza è quantificabile con una $\sigma \sim 1.3ns$. Un altro fattore ineliminabile è rappresentato dalla dispersione cromatica della luce nell'acqua di mare, che dipende sia dalle proprietà ottiche dell'acqua ma anche dalla disposizione geometrica dei rivelatori. A questi fattori và aggiunto il contributo dell'elettronica di "front-end", il

"jitter" sui segnali di clock e l'indeterminazione sulla posizione dei fotorivelatori, dato che ad uno spostamento di 10 cm corrisponde una incertezza sulle misure di tempo di $\sim$0.5 ns. Per risolvere quest'ultimo problema è prevista l'installazione di un sistema di rilevamento della posizione di tipo acustico che prevede dei trasmettitori posizionati sul fondale e dei ricevitori posizionati sulle torri così da poter ricostruire istante per istante la posizione ed eliminare tale fonte di incertezza.

4.4.3 Dati attraverso la torre

I dati uscenti dal modulo ottico sono trasportati attraverso tre coppie di cavi al modulo di controllo del piano. Questo modulo ha i seguenti compiti:

1. Comunicare con il modulo ottico a cui trasmette il clock a 1.215MHz, che estrae dal segnale in protocollo STM-1, e i segnali di "slow control" oltre alla potenza necessaria al suo funzionamento, e da cui riceve le informazioni degli eventi e i segnali di "slow control"(19.44Mbps).

2. Convertire i segnali da elettrici a ottici, tale funzione avviene tramite un Transceiver (FINISAR FTR-1631).

3. Assemblare i dati secondo il protocollo STM-1.

4. Comunicare con la Tower Junction Box; la comunicazione avviene tramite una fibra ottica ed è bidirezionale.

Lo schema del modulo di piano, definito Floor Control Module (FCM), è riportato in figura(4.3). Ai piedi della torre è presente la Tower Junction Box. Comunica con i 16 piani della torre anche

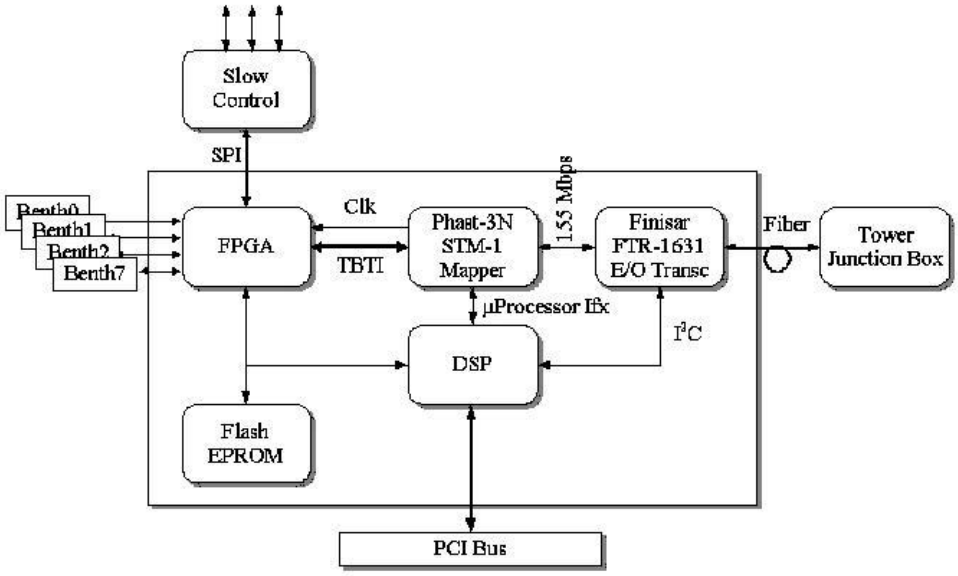


Figura 4.3: Floor Control Module

questa volta in maniera bidirezionale. Una funzione importante della TJB, che caratterizza tutti i trasferimenti di dati che avvengono tramite le fibre ottiche usate in NEMO, è la tecnica di campionamento di segnali su fibra ottica: DWDM (Dense Wavelength Division Multiplexing) che sarà illustrata nel prossimo paragrafo.

4.5 Sistema DWDM

Questa tecnica consiste nell'implementare su di una unica fibra ottica più segnali luminosi, ognuno con un diverso "colore". Il numero di segnali sovrapponibili è dato dalla larghezza di banda della fibra ottica attraverso la quale si vuol trasportare il segnale: attualmente è possibile trasportare fino a 128 canali differenti. La lunghezza d'onda fondamentale della luce che viaggia nei cavi è di 1550nm per i segnali provenienti dal mare e di 1300nm per quelli diretti a terra. Con la tecnica DWDM si è in grado dunque di fare uscire dalla TJB una unica fibra ottica contenente i flussi di dati provenienti da ciascuno dei sedici piani della Torre. Anche il segnale che arriva al modulo di controllo di piano è diviso in due parti, sono così separati i dati di ingresso da quelli di uscita.

Come abbiamo visto il sistema di trasferimento dati di NEMO è ramificato: la JB principale è connessa a otto JB secondarie attraverso cavi elettro ottici costituiti da quattro fibre ottiche e quattro conduttori elettrici. Le JB secondarie sono a loro volta connesse con otto Tower JB.

La prima "multiplazione" avviene nel concentratore di torre dove i 16 canali provenienti dai piani di una torre sono riuniti su di un'unica fibra, distanziati ognuno di loro di 200GHz. Il secondo stadio di "multiplazione" avviene nella Junction Box secondaria dove

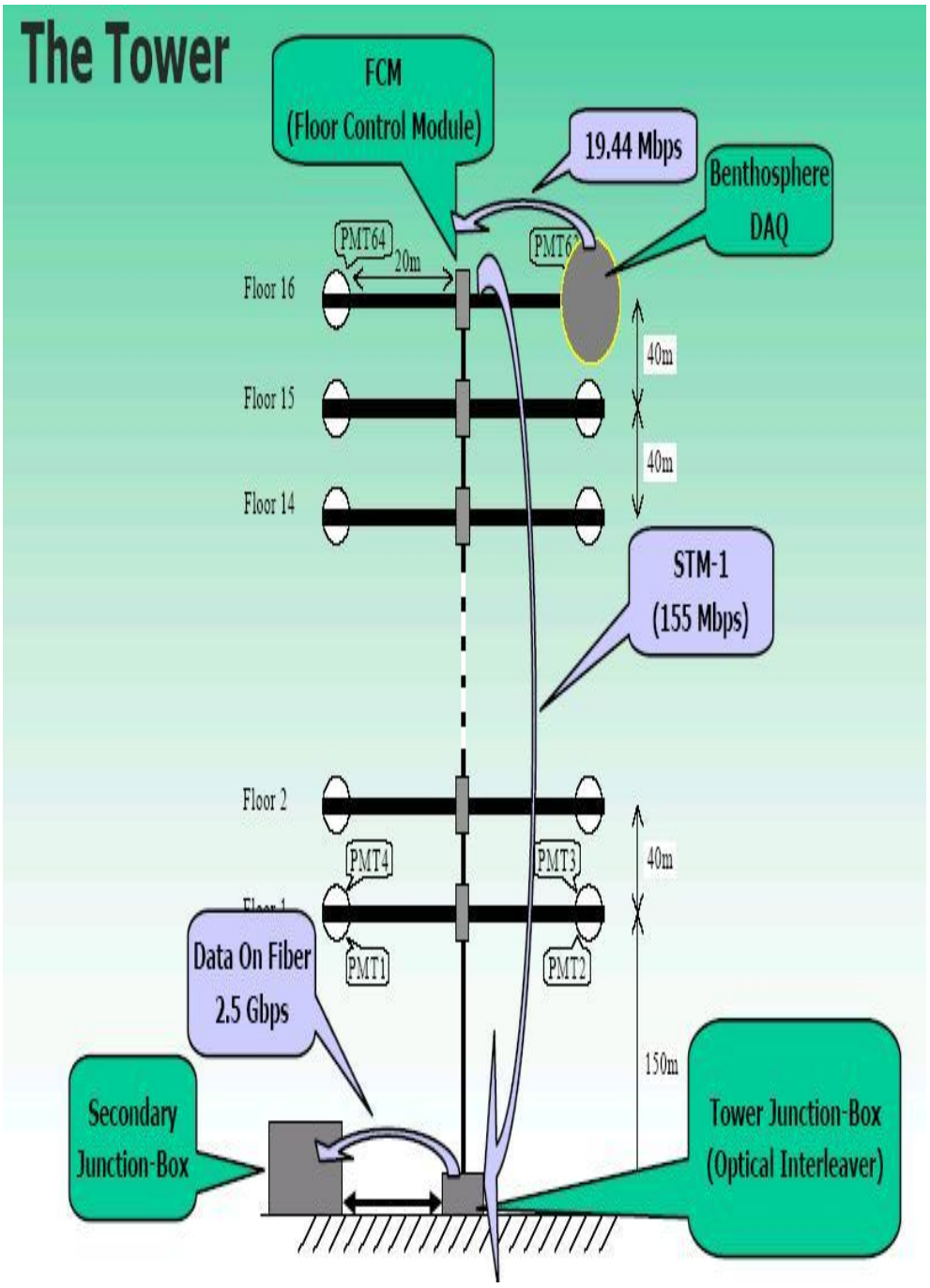


Figura 4.4: Dati attraverso la torre

i 32 canali provenienti da due torri sono "multiplati" su di un'unica fibra. All'uscita di questa JB abbiamo dunque un cavo elettro-ottico, costituito da 4 fibre ognuna contenente 32 canali distanziati di 100GHz ciascuno (0.8nm), diretto alla JB principale. Dalla JB principale parte, tramite un unico cavo, la comunicazione tra il sistema sottomarino e quello sulla costa. Lo schema di trasmissione dati è riportato in figura e per ulteriori informazioni si rimanda a ref.(18) e ref.(19).

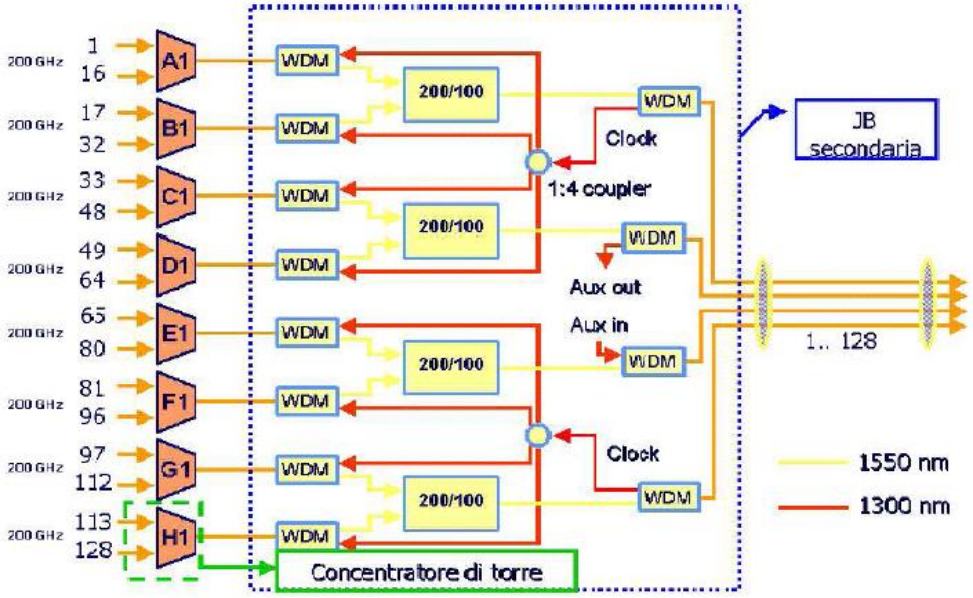


Figura 4.5: Sistema di trasmissione dati

4.6 Data Format

Come già accennato il protocollo scelto per la trasmissione dei dati dai diversi piani alla Tower Junction Box in NEMO è un protocollo Telecom standard sincrono: 155Mbps STM-1/SONET(Synchronous Over NETwork). I vantaggi di questo tipo di protocollo sono:

- Protocollo sincrono.
- "Data rate range" da 52 Mbps fino a 10 Gbps.
- Telecom Standard (sicuro,durevole,supportato..).
- Transceivers Elettro-Ottico disponibile
- Elettronica relativamente semplice.

Il Frame STM-1 è il "format" di base per la trasmissione SDH. É presente un Frame STM-1 ogni 125μs e quindi ce ne sono 8000 al secondo. Il modulo STM-1 è formato da una "Section Overhead" più un Virtual Container (VC). Le prime 9 colonne del Frame compongono il Transport Overhead, le restanti 261 compongono la capacità del VC. Il Frame è dunque composto da 270 colonne, la sua "altezza" è di 9 righe.

In figura (4.6) è mostrata una rappresentazione di come dovrebbero essere caricati i dati secondo il protocollo SDH nell'esperimento NEMO. Ogni "treno" di dati è composto da un Frame di testa seguito da quattro moduli STM-1, distanziati come detto di 125μs con una capacità di trasporto di informazione complessiva pari a 150Mb/s.

Come si vede anche nella figura è importante che i treni provenienti dalle diverse torri viaggino allineati tra loro (sincronizzazione ideale). Per maggiori dettagli sulle modalità di trasporto dei dati e i relativi problemi si rimanda a ref.(20)

4.7 Cavo elettro-ottico

Il progetto NEMO necessita di un cavo sottomarino per la distribuzione della potenza elettrica necessaria al funzionamento delle

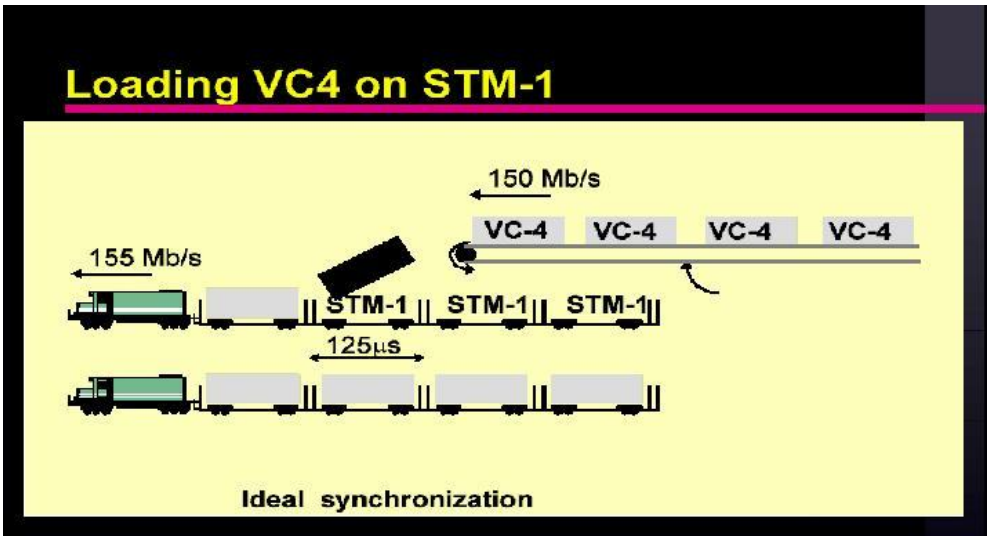


Figura 4.6: "Caricare" VC-4 su STM-1

strutture sottomarine e, come illustrato, per la trasmissione dei dati dei sensori ottici. La soluzione proposta è quella di utilizzare un unico cavo dotato di una doppia armatura d'acciaio ed adatto all'alloggiamento di fibre ottiche e conduttori metallici, simile a quelli utilizzati nel campo delle telecomunicazioni. Questi cavi permettono la trasmissione di segnali ottici fino a distanza dell'ordine di 100 km, senza alcuna necessità di amplificatori di segnale. Il progetto prevede l'utilizzo di un cavo composto da 48 fibre ottiche standard ITUT G-655, e 4 o più conduttori elettrici. La potenza al carico è circa 30/40 kW, con una tensione massima di 6 kV in corrente alternata trifase. La sezione del cavo è riportata in figura. L'operazione di interramento del cavo richiederà l'utilizzo di navi posa cavi idonee a tali scopi dotate di sistema di posizionamento di tipo Full Duplex e di sistemi di controllo computerizzati sia per la navigazione sia per il posizionamento del cavo sottomarino. Per motivi legati alla sicurezza del cavo, che potrebbe essere compromessa da attività di

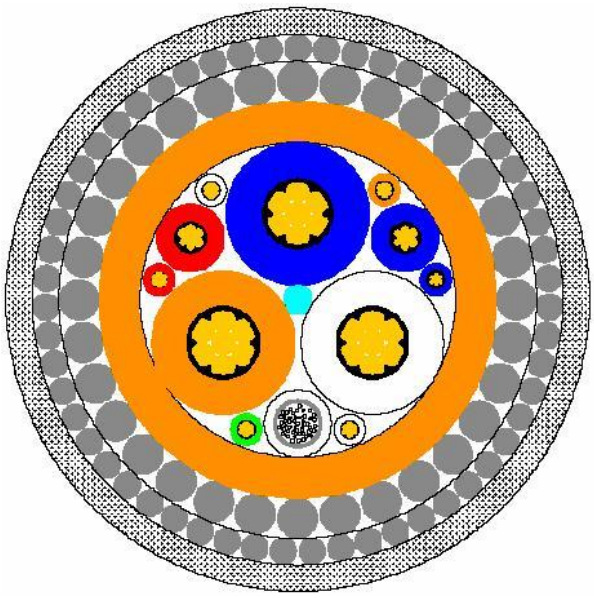


Figura 4.7: Sezione del cavo

pesca e dallo stazionamento di imbarcazioni, si provvederà all'interramento del cavo ad un profondità di 1m sotto il fondo marino. L'operazione di interramento sarà effettuata fino alla profondità di -1000m.

4.8 Sistema di distribuzione dell'energia

Le prime considerazioni che sono state fatte riguardo alla trasmissione dell'energia elettrica al sistema sottomarino riguardano le modalità con cui tale energia andava trasmessa. Le opzioni possibili sono: corrente continua monopolare o bipolare oppure corrente alternata monofase o trifase. In seguito a considerazioni di carattere economico e di perdita di potenza si è concluso che il modo migliore di trasferire energia è tramite corrente alternata trifase. La corrente trifase non è usata solo per la trasmissione ma anche per la succes-

siva distribuzione tra le varie parti del rivelatore. Ogni componente del "detector" vuole essere alimentato in corrente continua, ciò comporta la necessità di convertitori AC/DC. Per maggiori informazioni si rimanda a ref.(21).

Per fare un calcolo dell'energia di cui necessita l'apparato sottomarino bisogna considerare quanta energia serve a ciascun componente per funzionare e quali sono le relative distanze tra i componenti, considerata la sezione e il tipo di cavi elettrici. A questo punto, conoscendo i componenti, è possibile fare un calcolo di quanta potenza è necessaria:

- Per ogni piano vanno alimentati 4 fotomoltiplicatori, il modulo di trasmissione dati, gli strumenti per la misurazione dei parametri fisici e il sistema acustico di controllo della posizione che in totale richiedono 18W ad una tensione continua di 48V.
- Ogni JB di torre richiede energia per il modulo di concentrazione dei dati, sistema di controllo elettronico e sensori per la misura dei parametri fisici per un totale di 100W e anche questi componenti devono essere alimentati alla tensione continua di 48V.
- Le JB primarie e secondarie hanno gli stessi scopi delle JB di torre con lo stesso tipo di alimentazione ma necessitano di 200W.

Moltiplicando tali cifre per il numero di componenti otteniamo

$$18W * 1024 + 100W * 64 + 200W * 9 = 26.632kW$$

necessarie al funzionamento dell'apparato sottomarino.

Il sistema elettrico di distribuzione di tale potenza può essere diviso nei seguenti quattro tratti:

- Sistema di trasmissione dell'energia dalla costa al sito sottomarino: avviene in corrente alternata trifase tramite 4 conduttori elettrici.

- Sistema di distribuzione primario: dalla JB primaria a quelle secondarie e sempre in corrente alternata trifase con 4 conduttori elettrici.

- Sistema di distribuzione secondario: dalle JB secondarie ai moduli di torre avviene tramite corrente alternata trifase e 4 conduttori più un canale con corrente monofasica.

- Dal Box di torre ai piani e avviene in corrente alternata monofase.

Il sistema appena descritto è mostrato in figura 4.8:

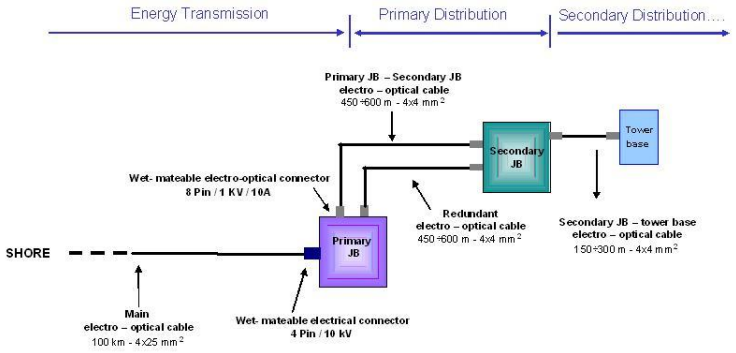


Figura 4.8: Sistema di distribuzione dell'energia

La potenza iniziale trasmessa di 36kW, ridondante rispetto alle reali necessità, viene in parte persa nell'attraversamento del cavo di trasmissione e la restante è poi distribuita lungo i componenti. Anche la tensione trasmessa ha una caduta lungo il cavo e durante le successive distribuzioni. Nella figura seguente sono riportati i vari

livelli di potenziale elettrico e la distribuzione della potenza erogata.

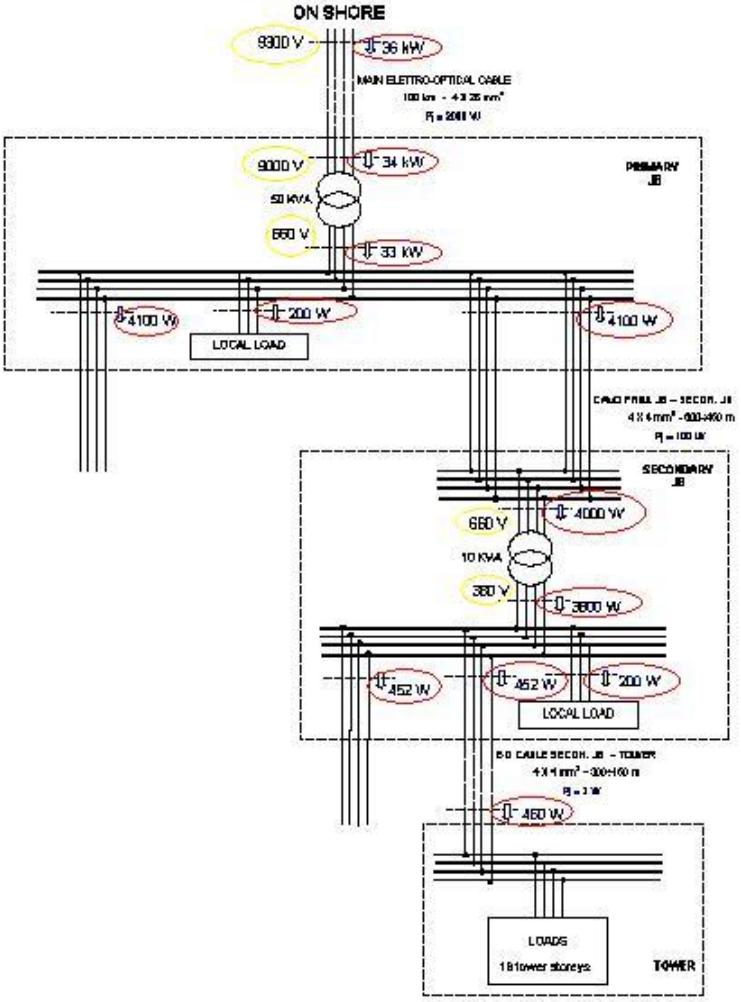


Figura 4.9: Distribuzione della potenza e livelli di potenziale

Capitolo 5

Ricezione dati

5.1 Trigger di primo livello

La funzione di un trigger è quella di selezionare gli eventi quando determinate condizioni sono soddisfatte e eliminare, invece, tutti quegli eventi che non sono significativi. I segnali prodotti dai fotomoltiplicatori, come abbiamo visto, sono in larga misura dovuti a processi di fondo: decadimento del ^{40}K, bioluminescenza. Se si dovessero analizzare tutti i dati che arrivano a terra la mole di calcolo sarebbe eccessiva. Una ricostruzione degli eventi significativi sarebbe allora impossibile.

La condizione per cui un dato deve essere preso in considerazione, ed un altro no, deve essere ricavata da considerazioni di tipo fisico sul fenomeno cercato. L'effetto Cherenkov, prodotto indirettamente dal neutrino, produce un cono di luce con un angolo noto e con un fronte d'onda che viaggia ad un velocità nota anch'essa. É quindi possibile cercare una correlazione tra le posizione dei fotorivelatori e l'istante in cui vengono colpiti dai fotoni. Questa correlazione deve essere cercata per tutti quei rivelatori che hanno una distanza,

dal punto in cui un evento è stato rivelato, confrontabile con la propagazione della luce nell'acqua. Questo lavoro richiede, oltre ad un alta velocità di calcolo, che siano accessibili i dati provenienti da tutti i moduli ottici. Un trigger di questo genere è possibile solo disponendo di una buona potenza di calcolo per l'esecuzione di un software specifico.

Quello che è possibile fare, per facilitare questa analisi tramite software, è cercare una prima correlazione tra fotorivelatori presenti sullo stesso piano e tra quelli dei primi piani vicini di una stessa torre. Questo lavoro potrebbe essere direttamente fatto dalla scheda di concentrazione che riceve i dati provenienti dal mare di una stessa torre, eliminando i dati che non soddisfano il trigger di torre.

Prima di entrare nei dettagli della scheda e del suo funzionamento vediamo alcune proposte e caratteristiche dell'architettura di ricezione.

5.2 Frequenza dei dati

La caratteristica del trigger di torre è dunque la correlazione temporale tra eventi registrati dai PMT del piano e dei piani primi vicini. Le opzioni per accendere il trigger sono due:

1. Trigger 1) Rivelazione di un evento tra tre PMT dello stesso piano.

2. Trigger 2) Rivelazione di un evento tra due PMT di un piano e due PMT del piano vicino.

Da considerazioni di tipo statistico e considerando che il fondo ottico presente è di $50/100kHz$, la frequenza con cui i due tipi di trigger di torre dovrebbero attivarsi è di:

- Trigger 1) $0.24/1.96Hz$
- Trigger 2) $0.11/1.74Hz$

La frequenza dei dati che soddisfano il trigger in tutto l'apparato dovrebbe dunque essere di $355/3678Hz$, con un flusso di dati dopo il primo trigger di $0.0581/2.41Gbit/sec$.

Da previsioni di questo genere può essere fatta una stima sulla quantità dei dati che arriverà a terra e soprattutto una stima su quei dati che andranno conservati per l'elaborazione. I risultati sono riportati nella tabella seguente:

	Torre/s	Torre/day	NEMO/day
Raw Data	40MB	3.5TB	200TB
Trigger 1	0.1MB	10GB	640GB
Trigger 2	...	20GB	1.25GB

Tabella 5.1: Data storage

5.3 Tecnologia

Prima di descrivere i diversi modelli proposti per l'architettura di ricezione, poichè comunque questa si basa su l'utilizzo di PC commerciali che immagazzinano i dati da analizzare, vediamo alcuni numeri caratteristici su tali PC per avere una stima sulle disponibilità di calcolo che ci offre l'attuale tecnologia.

I PC commerciali attualmente disponibili, basati su bus PCI standard e "low-end", possono raccogliere un flusso pari a:

$$33(66)MHz * 32(64)bit = 132(528)MB/s$$

e per ogni PC esistono

$$4(2)slots + hostcpu$$

Nel futuro prossimo è prevista l'introduzione di PC con bus PCI-X di nuova generazione che sono caratterizzati da banda passante pari a:

$$133MHz * 64bit = 1GB/s$$

e numero ridotto di slot. I collegamenti possibili potrebbero essere basati su

- Ethernet 10/100, Gigabit Ethernet.
- Myrinet (2-5 Gb/s).

Il software previsto per i PC è Linux Open Source.

Per la tecnologia attuale, dunque, la trattazione della mole di dati prevista dell'apparato NEMO non è impressionante, ed inoltre, dato il largo impiego di componenti commerciali il continuo sviluppo della tecnologia potrà fornire al momento dell'installazione di NEMO componenti migliori di quelli ora disponibili.

5.4 Architettura DAQ

Una possibile architettura di ricezione, stoccaggio e analisi dati prevista è riportata in figura(5.1) (ref.(24)).

Il blocco indicato con Elettro-Ottico è il sistema di conversione ottico-elettrico. Le schede indicate con C_n sono i concentratori, mentre quelle indicate con SC sono le schede di ricezione ed invio dei segnali di Slow Control. Le CPU elaborano le informazioni del singolo piano e, in caso di attivazione del trigger, le inviano al cluster di PC per la ricostruzione degli eventi.

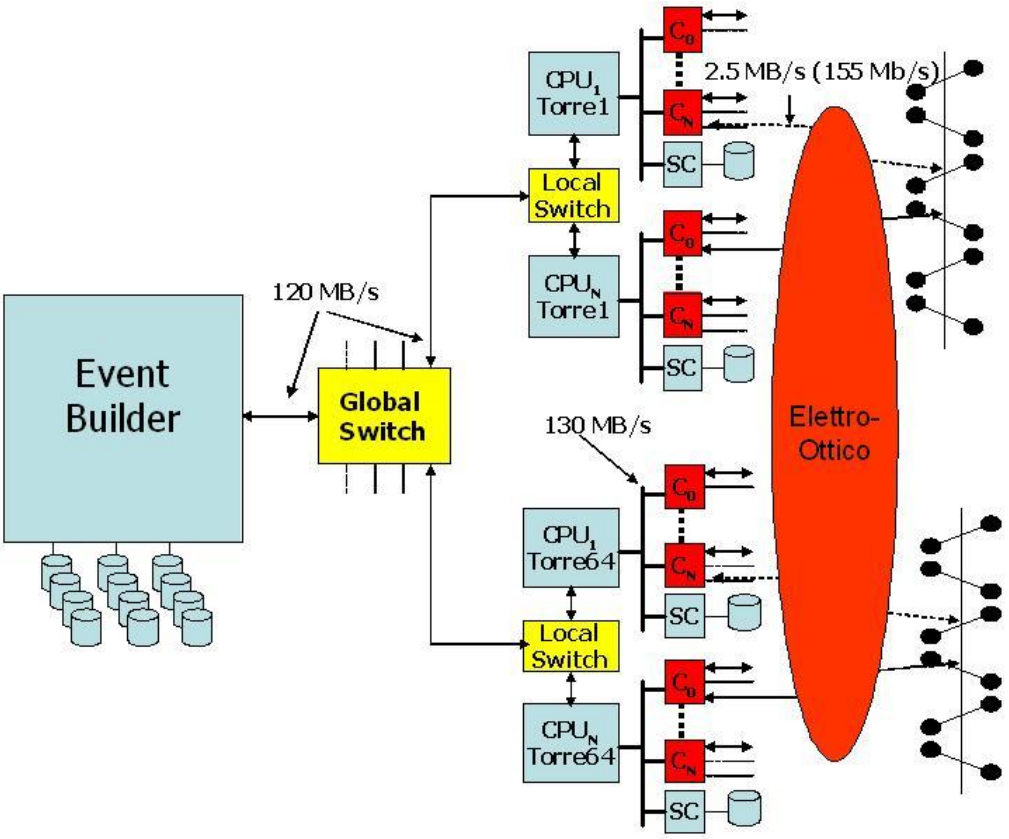


Figura 5.1: Architettura Daq

Il trigger di torre potrebbe essere fatto interamente dalle CPU presenti prima dei concentratori attraverso l'uso di un Software, oppure in Hardware dai concentratori od un misto tra le due opzioni. Vediamole con maggior dettaglio:

Opzione 1:Puro Software La ricostruzione temporale e il riconoscimento dell'evento di torre vengono fatti dalla CPU, la scheda di concentrazione è un'interfaccia tra protocollo SDH e il Bus PCI. Per ogni concentratore non è necessario avere informazioni sui piani primi vicini e quindi ad ogni porta dei concentratori di una CPU corrisponde un piano diverso. Detti S il numero di concentratori per CPU, M il numero di porte per schede di concentazione, e P il numero di CPU per torre il prodotto di questi tre numeri, a meno delle ridondanze necessarie, deve essere uguale al numero di piani di una torre:

$$PSM - (P - 1) = 16(piani/torre)$$

Da cui ricaviamo la seguente tabella con il numero di componenti necessari:

S	M	P
3	1	8
3	2	3
3	3	2
3	4	2
4	1	5
4	2	3
4	3	2
4	4	1

Opzione 2:Misto Software e Hardware La scheda di concentrazione riallinea temporalmente i dati dei canali provenienti dai singoli piani, lasciando però alla CPU il compito di riconoscere l'evento.

Opzione 3:Puro Hardware La scheda ha sia il compito di riallineare i dati che quello di riconoscere una coincidenza tra dati del piano e dei primi piani vicini. Quindi ogni concentratore deve avere le informazioni di torre provenienti da un piano e dai piani primi vicini. Anche in questo caso a meno del numero di canali ridondanti necessari per il trigger deve essere:

$$PSM - (PS - 1) = 16(piani/torre)$$

Da cui ricaviamo:

S	M	P
3	3	3
3	4	2
4	3	2
4	4	2

Per capire quale delle tre opzioni mostrate è la migliore è necessario simulare l'apparato di ricezione. La simulazione andrà fatta tentando di riprodurre le condizioni dell'apparato previste nell'esperimento NEMO (condizioni quasi reali).

5.5 Simulazione Hardware

Attraverso delle simulazioni dell'apparato di ricezione si può dunque arrivare a scegliere l'architettura migliore tra quelle proposte.

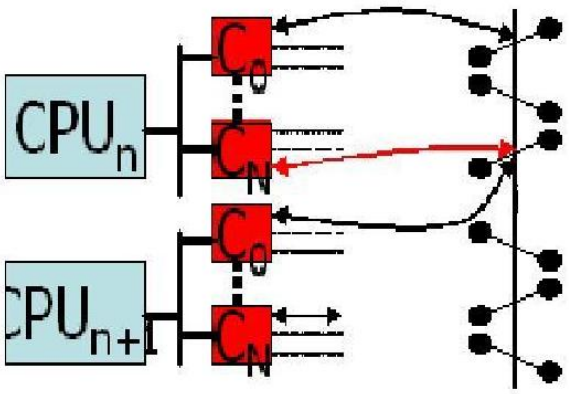


Figura 5.2: Architettura Software. Prevede un canale aggiuntivo ogni due schede CPU.

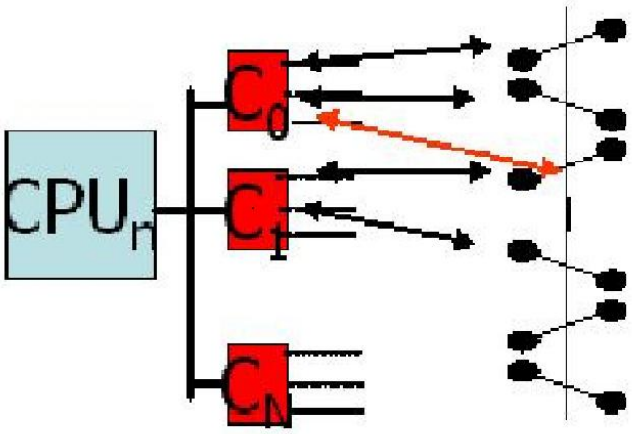


Figura 5.3: Architettura Hardware. Prevede un canale aggiuntivo ogni due schede di concentrazione.

A tale scopo si costruisce un apparato costituito da un banco di memoria, una FPGA (Field Programmable Gate Array) e un bus PCI con cui si interfaccia la FPGA, come in figura (5.4). Queste simulazioni possono essere fatte in condizioni quasi reali dell'apparato, caricando nel banco di memoria dei dati ottenuti da simulazioni di Montecarlo oppure da dati fisici ricavati ad esempio negli altri esperimenti di telescopi sottomarini (ANTARES) o infine da campagne di acquisizione specifica come quelle realizzate dalla collaborazione NEMO per la caratterizzazione del sito.

Una volta caricati in memoria, i dati sono letti dalla FPGA alla frequenza aspettata e inviati poi al bus.

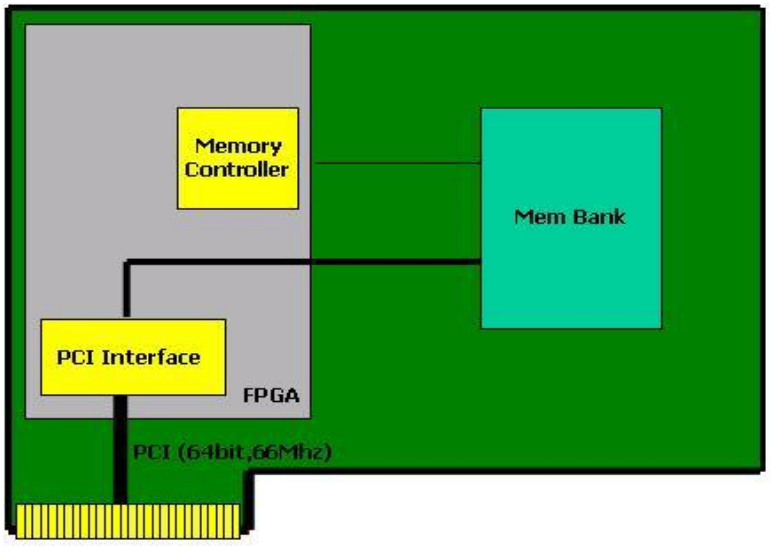


Figura 5.4: Schema di emulazione

5.6 FPGA

La FPGA usata nella simulazione svolge il compito della scheda di concentrazione prevista nel DAQ ed ha il compito di comunicare con il bus PCI. Questa si interfaccia con il bus PCI attraverso un blocco specifico denominato "PCI-CORE" che semplifica la realizzazione del sistema. L'FPGA, il cui nome è ACEX EP1K00EFC484-1 ed è prodotta dall'azienda ALTERA, è contenuta nella scheda che si chiama PCISYS (prodotta dalla PLDA). Quest'ultima può essere alimentata sia a 5 che a $3.3V$, tratta dati a $32/64bit$, e la frequenza di scambio dati con il bus è di $33/66MHz$.

Lo schema a blocchi della PCISYS è riportato in figura5.5.

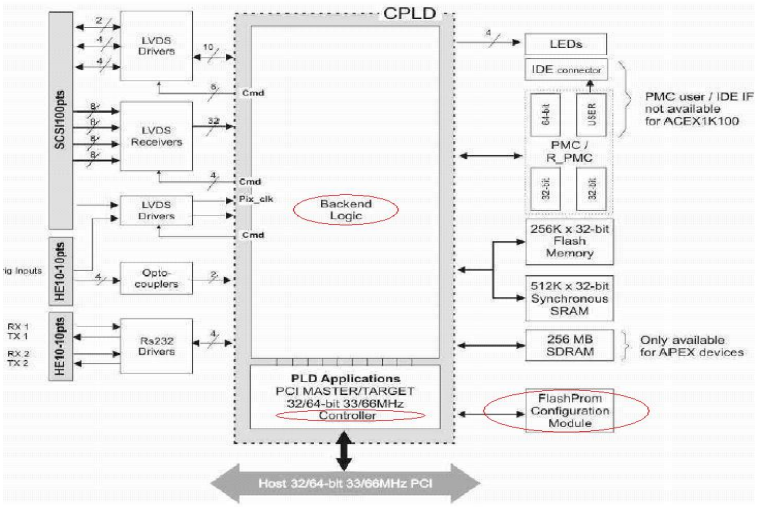


Figura 5.5: Diagramma a blocchi della PCISYS

L'insieme del Logic Backend con il PCI-CORE costituisce la CPLD (Complex Programmable Logic Device). L'utente può configurare il Logic Backend tramite un FlashProm Configuration Module, e il Backend è interfacciato con il bus tramite il Controller. Questa

scheda può essere configurata come Master o come Target.
Dato che la scheda di concentrazione ha il compito di inviare i dati al bus con un Timing specificato, questa dovrà essere configurata in modalità MASTER.

Dovendo leggere le informazioni temporali e spaziali contenute sui dati, questi devono avere un formato specificato che mostrerò in seguito. Inoltre la CPLD deve contenere un blocco di controllo della memoria e un blocco dedicato alla comunicazione col bus.

La memoria necessaria per raccogliere i dati è presente sulla PCISYS ed è di tipo SRAM (Synchronous Random Access Memory).

5.7 Struttura dei dati

I dati che dalla stazione sottomarina arrivano alle schede di concentrazione sono raggruppati in una struttura determinata. Nel capitolo precedente è stato mostrato come, secondo i progetti attuali, i dati vengono sistemati secondo il protocollo SDH e quindi come dovrebbe essere tale struttura: un "burst" di dati è composto da quattro Frame, ed ogni Frame è un modulo STM-1. Per rispecchiare le condizioni previste per l'apparato di ricezione nell'esperimento NEMO è necessario che i dati caricati in memoria abbiano tale formato.

Anche nella simulazione quindi un burst di dati è composto da quattro Frame. Ogni Frame di dati è composto da un numero variabile di Word, con "in testa" una Word di identificazione chiamata Header. Ogni Word è formata da 32bit.

Esistono tre differenti tipi di Frame:

1. SLOW FRAME

   È composto da due sole Word, l'Header e una seconda Word, contenente l'informazione sul tempo assoluto di avvenimento

dell'evento registrato nel DATA FRAME successivo. Il tempo viene registrato da opportuni contatori posti all'interno dei moduli ottici. Il clock di tali contatori, che caratterizza anche la frequenza di campionamento, è di 200MHz. Quindi 5ns è l'unità di tempo. Ogni Slow Frame copre una finestra temporale di 500μs, composta da quattro Frame di 125μs contenute nei quattro Data Frame seguenti. L'Header è così strutturata:
Bits(31;30)=(0;1),
Bits(29;23)Spare,
Bit(22)Data Section Overflow,
Bits(15;0)Absolute PMT identifier.

2. DATA FRAME

   È composto da un Header e un numero variabile di Word da 1 a 64. L'Header del Data Frame è così strutturata:
   Bits(31;30)=(1;0),
   Bits(29;26)PMT identifier,
   Bit(22)Data Section Overflow,
   Bits(21;16)Frame Size,
   Bits(15;0)Time Record.
   Le Word che contengono le informazioni dell'evento sono composte da 4 campioni di 1Byte ciascuno.

3. SPECIAL FRAME.

   È composto solo dall'Header che ha due '1' nei bit 31 e 30 e serve solo nella simulazione per comunicare al concentratore che la simulazione è finita. Non sarà dunque presente nell'esperimento vero e proprio.

   La sequenza con cui tali Frame si susseguono non è casuale, ad

ogni Slow Frame, che apre il Data Stream, segue un numero variabile di Data Frame, la struttura si ripete fino a quando si presenta uno Special Frame che termina la simulazione.

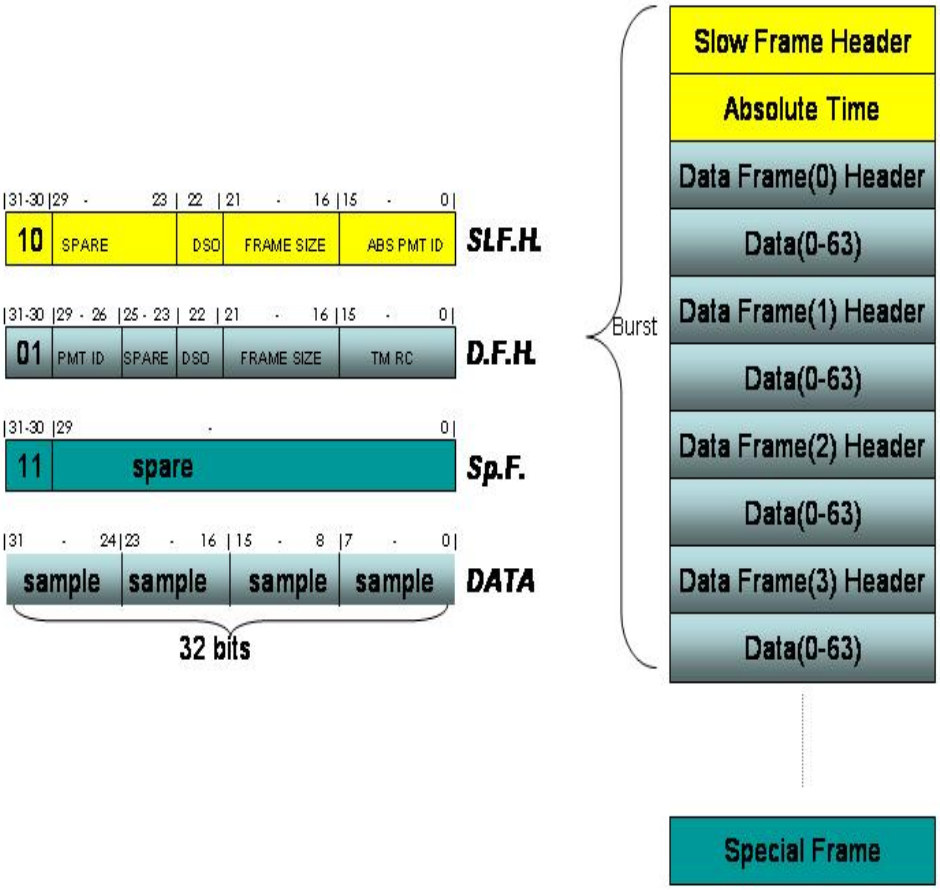


Figura 5.6: Struttura dei dati nella memoria dell'emulatore.

5.8 Scheda di emulazione

Il CPLD, che chiameremo per semplicità emulatore, è costituito dal Logic Backend e dal "PCI Core", entrambe configurabili dall'utente.

Il Logic Backend è costituito essenzialmente da una DMA (Direct Memory Access), due FIFO (First In First Out), un Memory Control e un orologio interno. Sulla scheda è presente anche la memoria dove immagazzinare i dati per l'emulazione. Lo schema a blocchi è riportato in figura.

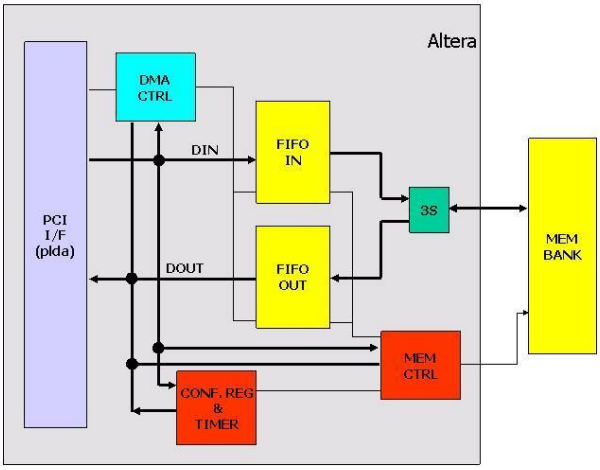


Figura 5.7: Schema a blocchi del circuito.

L'emulatore si interfaccia da un lato con il bus PCI e dall'altro con la memoria ed ha, durante l'emulazione, due diverse fasi ognuna con compiti diversi: in una prima fase deve prendere i dati dal bus PCI e trasferirli tutti alla memoria; in una seconda fase deve prenderli dalla memoria e trasferirli al Bus, leggendo le informazioni presenti sui dati ed effettuando eventualmente il trigger oppure un riordinamento temporale a seconda dell'opzione scelta per l'architettura di ricezione.

Attualmente si è scelto di affidare al software il trigger di primo livello, lasciando alla scheda di concentrazione il compito di riallineare temporalmente i diversi dati.

L'emulatore è stato dunque programmato con due modalità di funzionamento:

- SYSTEM MODE: Viene caricata la memoria sulla scheda con la sequenza di dati precostituita. Il caricamento avviene a partire dall'indirizzo '0' aumentando il puntatore alla memoria ad ogni Word scritta. La struttura di tale sequenza è una ripetizione ad indirizzi adiacenti di Slow Frame e Data Frame.
- RUN MODE: Modalità emulatore. I dati vengono letti dalla memoria con un timing specificato e scritti nella FIFO di lettura per essere poi inviati dal DMA controller del core PCI nella memoria del PC.

Vediamo con più attenzione i singoli componenti del circuito di emulazione.

5.8.1 Bus PCI

Il Bus è il percorso lungo il quale i dati passano dal microprocessore alla memoria e ai dispositivi periferici , come ad esempio le schede periferiche. La necessità di poter aggiungere schede standard nei PC, ha portato al formarsi di diversi tipi di Bus con standard ben definiti, tra i più usati tipi di Bus ricordiamo Bus tipo ISA, EISA, AGP, e il Bus PCI (Peripheral Component Interconnect).

Il Bus PCI è un tipo di Bus a 32 bit, estendibile a 64, con una frequenza di clock di 33 o 66 MHz. Normalmente una Mainboard dispone di slot PCI in numero variabile da 2 a 5 o piú. Il vantaggio principale dell'utilizzo del Bus PCI rispetto agli altri tipi è la velocità di trasferimento dei dati che è di 132MB/s, un altro vantaggio è quello della presenza di tale bus sulla quasi totalità delle piattaforme commerciali.

Ci sono tre diversi tipi di componenti che utilizzano il Bus PCI:

- PCI Bus/System Bridge. Interfaccia il Bus PCI al processore di sistema e alla memoria principale. Questo tipo di componente può agire sia da Master che da arbitro:
  Master è quel componente che può richiedere l'utilizzo del Bus per l'invio o la ricezione di dati dagli altri componenti collegati con il Bus. L'arbitro è il componente che decide secondo una determinata scala di priorietà chi ha momentaneamente accesso al Bus.

- Schede PCI Master. Componenti aggiuntivi che possono utilizzare il Bus e richiedere l'accesso alle altre schede aggiuntive o alla memoria principale del sistema.

- Schede PCI Target. Componenti aggiuntivi che possono agire solo come "bersaglio". Questi componenti possono rispondere a richieste del Master ma non possono cominciare un ciclo di Bus.

In figura è mostrato un tipico sistema PCI:

5.8.2 Core PCI

Come già detto la scheda si interfaccia con il Bus PCI attraverso un Core, che rende semplice il disegno della logica "custom" al bus-PCI. Quasi tutte le proprietà del Core sono programmabili e configurabili per adattarsi alle diverse funzionalità della scheda. Il Core è costruito attorno ad una macchina a stati centrale che controlla tutte le operazioni e che assicura la sincronizzazione con le operazioni del Bus. In figura è mostrato lo schema della scheda sul Bus PCI con

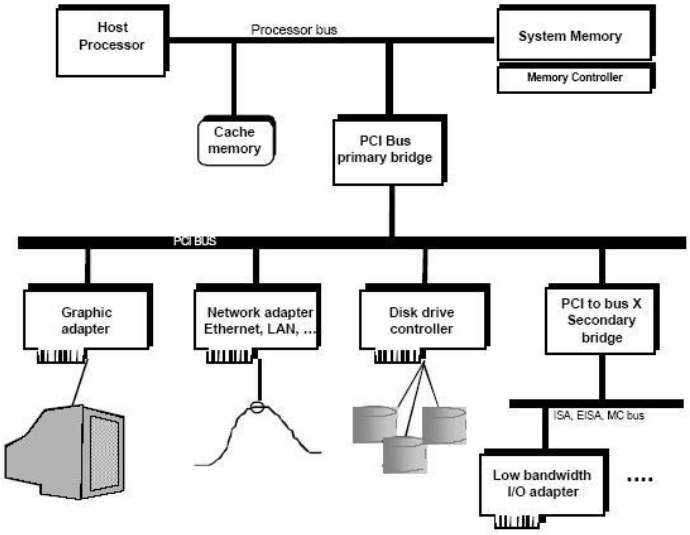


Figura 5.8: Sistema PCI

in evidenza i componenti del Core.
Il Core PCI contiene i seguenti blocchi funzionali:

- Core State Machine. Controlla tutte le operazioni di interfaccia PCI. Il suo stato riporta la sequenza delle operazioni sul Bus PCI.

- Configuration Space. Controlla gli accessi di scrittura e lettura nei registri di configurazione del Bus PCI. Contiene anche la decodifica degli indirizzi che permettono al Core di rispondere alle transazioni quando questi appaiono nella sua memoria o nello spazio degli indirizzi di I/O.

- Parity Control. Può essere abilitato o meno, riporta i calcoli di data parity e gli errori al sistema.

- Interrupt Support. Implementa un controllo per i processi di

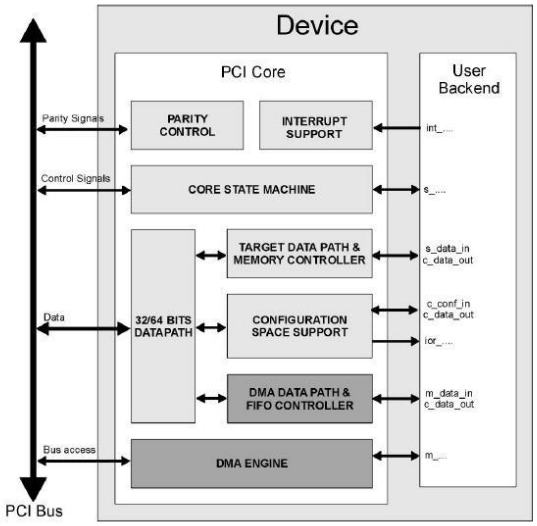


Figura 5.9: Architettura del Core

interrupt richiesti sul bus PCI.

- 32/64 Data Path. Interfaccia il Bus Data/Address del PCI con i Bus interni.
- Target Data Path. Assicura la fattibilità di un trasferimento dati in modalità Target e può eventualmente controllare l'accesso diretto alla memoria della scheda.
- DMA Engine. É una macchina a stati secondaria che controlla tutte le operazioni in modalità Master.
- DMA Data Path. Assicura il trasferimento coerente di dati in modalità Master e può opzionalmente controllare in maniera diretta le FIFO.

Quindi il Backend direttamente programmato dall'utente può comunicare con il Bus solo attraverso il Core e tutte le transazioni dirette sono proibite. Il Backend và quindi programmato per interfacciarsi

con il Core PCI e non con il Bus. Nel prossimo capitolo descriverò come questo è stato programmato per l'emulazione.

5.9 Memorie On-board

La scheda PCI utilizzata per l'emulazione è dotata di due memorie:

- SRAM Module

- FLASHPROM Module

La prima è la memoria in cui vengono posizionati i dati durante la fase System. É una memoria RAM (Random Access Memory) sincrona e il nome del modello è CY7C1383B.

La memoria di tipo FLASH, è una ROM (Read Only Memory) programmabile, che serve appunto a programmare il Backend nella CPLD. Programmata attraverso Bus PCI all'accensione della scheda trasferisce le informazioni alla CPLD.

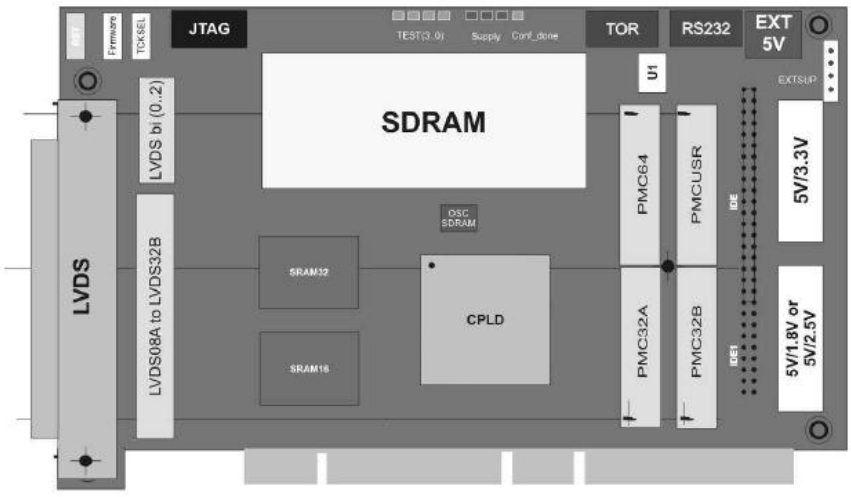


Figura 5.10: Disposizione della PCISYS (lato componenti)

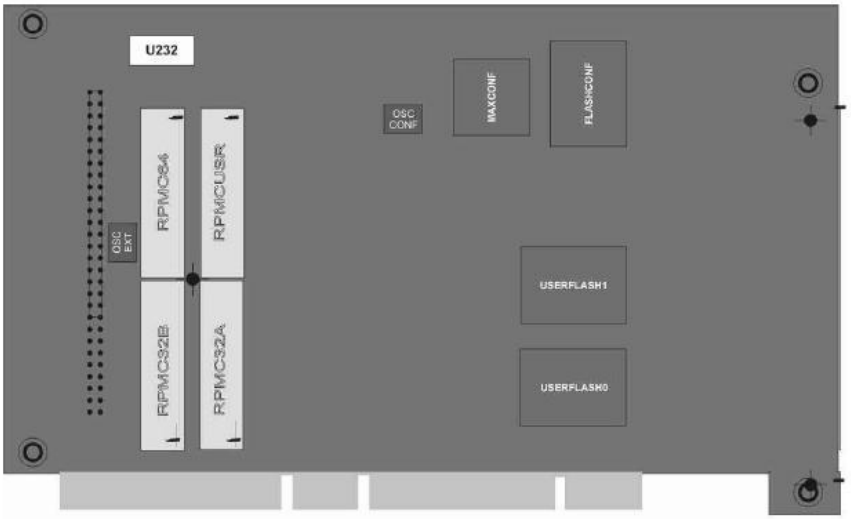


Figura 5.11: Disposizione della PCISYS (lato saldature)

Capitolo 6

Logic Backend

6.1 Introduzione

In questo capitolo sono descritte le modalità di programmazione del Logic Backend e le caratteristiche del circuito in esso contenuto. Partendo da una breve descrizione del linguaggio usato per il codice di programmazione (VHDL) e del suo uso nel processo di sviluppo della scheda, il capitolo si conclude con la descrizione del funzionamento dei singoli blocchi del circuito, dediti sia al controllo dei dati contenuti nella memoria "on-board" che alla comunicazione con il PCI-Core e, dunque, con il bus-Pci.

6.2 Standard VHDL

Il VHDL è un linguaggio di descrizione Hardware. L'abbreviazione VHDL deriva da VHSIC (Very High Speed Integrated Circuit) e Hardware Description Language (linguaggio di descrizione hardware per sistemi ad alta velocità di integrazione).

All'inizio degli anni ottanta, il dipartimento della difesa americana (DOD) desiderava creare un linguaggio standard, e ad un più alto

livello di astrazione, per descrivere e documentare i sistemi hardware anzichè utilizzare un linguaggio logico proprietario, ed avere così indipendenza dai loro fornitori. É per questo che DOD creerà due linguaggi che si assomiglieranno fortemente: ADA (logico) e VHDL (hardware-fisico).

Lo standard VHDL prenderà piede nel 1987 e verrà normalizzato dall'IEEE (Institute of Electrical and Electronics Engineers). Nel 1993 una nuova revisione dell'IEEE ha permesso di sfruttare a pieno le capacità del linguaggio VHDL, in particolare per:

- La sintesi automatica dei circuiti a partire dalla descrizione.
- La verifica delle descrizioni temporanee (prototipi).
- La verifica formale dell'equivalenza dei circuiti.

Il principale vantaggio nell'usare il VHDL è dato dal fatto che è un linguaggio standard riconosciuto da tutti i fornitori di strumenti CAE (Computers Aided Electronics) e quindi, come già detto, indipendente dal fornitore.

6.3 Configurazione della CPLD

Una volta scritto il codice, con il linguaggio appena descritto, verificata la sua correttezza e sintetizzato in un codice di programmazione, con i metodi che descriverò nel prossimo capitolo, questo viene fisicamente trasferito sulla scheda attraverso l'uso di un opportuno cavo oppure tramite il bus PCI a cui la scheda è connessa. Il codice, tradotto in linguaggio binario, viene registrato nella memoria FLASH-EPROM presente sulla scheda e, ad ogni successivo avvio della scheda, la memoria è letta da un opportuno CPLD (MAX3128) che provvede poi alla configurazione della FPGA.

6.4 Funzione del Logic Backend

Come già detto nel capitolo precedente il compito dell'emulatore è quello di "caricare" i dati di simulazione nella memoria (SYSTEM MODE) e, avviata l'emulazione (RUN MODE), spedirli con un preciso "timing" sul Bus PCI. Il "timing" è determinato dal confronto tra il "tempo assoluto"(T_a) presente sui dati e quello locale della scheda. Poichè T_a occupa la seconda Word dello Slow Frame, deve essere fatta una lettura del tempo per ogni burst di dati (dove con burst si intende uno Slow Frame con i successivi quattro Data Frame). Nel caso in cui il tempo assoluto sia maggiore di quello locale, il MEMORY CONTROLLER invia i dati nella FIFO di uscita e avvisa il driver che un burst è presente in questa FIFO. Il segnale di avviso è codificato in un registro interno al Backend. Il registro in questione si chiama NSFP (Number Slow Frame Present) ed è scrivibile dal MEMORY CONTROLLER e leggibile dal driver applicato alla scheda in modalità Target. Il registro contiene un contatore che viene incrementato dal MEMORY CONTROLLER ogni volta che un nuovo burst è presente nella FIFO di uscita ed è anche fornito di un bit che avvisa il driver nel caso in cui al MEMORY CONTROLLER si presenti uno Special Frame e, quindi, l'informazione di fine emulazione. Il registro deve invece essere decrementato di uno quando il driver ha finito di leggere un intero burst. Il significato di ciascuno dei 32 bit che compongono tale registro è riportato in appendice A.

La scheda in modalità RUN deve dunque interfacciarsi sia con la memoria, dalla quale deve leggere i dati e spedirli nella FIFO di uscita, che con il bus PCI in cui deve inviare i dati presenti nella FIFO di uscita.

6.4.1 Interfaccia memoria locale-FIFO

Il MEMORY CONTROLLER è dotato di un "orologio" che, quando viene attivato il RUN MODE, è azzerato e comincia a contare fornendo così la misura del tempo locale (T_i). Il MEMORY CONTROLLER confronta poi il tempo assoluto con il tempo locale e quando

$$T_a \geq T_i$$

spedisce i dati presenti in memoria nella FIFO di uscita. Questo processo finisce non appena al MEMORY CONTROLLER si presenta uno Special Frame che determina il passaggio della scheda in modalità SYSTEM, completando prima però la scrittura del Frame che era in quel momento eventualmente in corso. La scheda può essere portata in modalità SYSTEM anche dal driver e il passaggio avviene come nel caso precedente.

In caso di sovrascrittura della FIFO già piena, il MEMORY CONTROLLER non interrompe la scrittura della FIFO ma segnala l'evento al driver mediante un opportuno segnale di Interrupt.

6.4.2 Interfaccia FIFO-PCI

Il passaggio dei dati dalla FIFO di uscita al Bus è completamente gestito dal driver applicato alla scheda. Questo, dopo aver portato la scheda in modalità RUN, attende il segnale di Interrupt che lo avvisa quando un burst è presente nella FIFO di uscita. A questo punto il driver legge le Header dei Frame e conseguentemente attiva il blocco DMA del Backend, affinchè cominci un trasferimento di dati sul bus pari al numeri di Word che costituiscono il burst contenuto nella FIFO. Al termine dell'operazione il driver decrementa il registro

NSFP. A questo punto legge nuovamente il registro e, se ci sono nuovi dati attiva ancora il DMA, altrimenti aspetta un nuovo segnale di Interrupt. Il driver deve anche occuparsi di indirizzare i dati in dei buffer di memoria, ognuno pari alla dimensione massima di un burst completo, ossia $2Word + 4 * 64Word$.

6.5 Circuito interno

Il circuito è formato essenzialmente da due parti:

- Il DMA che si occupa del passaggio di dati tra FIFO e Bus PCI.
- Il MEMORY CONTROLLER che controlla il passaggio di dati tra memoria e FIFO.

Nel circuito sono presenti due FIFO:
La FIFO di ingresso è coinvolta nella fase SYSTEM. Sono qui inseriti i dati provenienti dal Bus e diretti alla memoria. La sua dimensione è di 32 bit di lunghezza per ogni dato e può contenere un numero di Word pari a 256, per una dimensione totale di 8192 bit.
La FIFO di uscita è coinvolta invece nella fase RUN. Raccoglie i dati provenienti dalla memoria in attesa che siano inviati al Bus. Rispetto alla precedente può contenere un numero maggiore di Word pari a 1024 per una dimensione totale di 32768 bit.
Le FIFO sono dotate di un Reset di tipo asincrono, attivando questa funzione in qualsiasi istante vengono posti a zero tutti i bits in esse contenute.

Gli altri componenti che costituiscono il circuito sono:

- Address Select.

- Reset.
- Interrupt.
- Registro.
- Switch.
- Tristate.

Tutti questi componenti sono sincroni. Il clock utilizzato per il loro funzionamento è quello fornito dal Bus PCI pari a 33MHz. Ciascun componente è descritto nei successivi paragrafi.

6.5.1 DMA

Il DMA è il componente che si occupa direttamente del passaggio dei dati tra il Bus e le FIFO. Questo componente del circuito è a sua volta costituito da due blocchi:

- Configuration DMA
- DMA Controller.

Il primo serve a configurare il funzionamento del DMA attraverso la scrittura in modalità Target di opportuni registri. I registri in questione sono il DMA Register, il TAR (Target Address Register) e il DCR (Direct Control Register). Attraverso la scrittura del DMA Register si può impostare il DMA come Target o come Master e se ne può interrompere l'eventuale attività. Un'altra funzione di tale registro è quella di Reset, scrivendo infatti '1' nel trentaduesimo bit del registro si attiva il componente RESET del circuito la cui descrizione è riportata in seguito. Leggendo il DMA Register si può verificare quale è lo stato attuale della macchina a stati contenuta

nel DMA Controller, quante parole sono contenute nelle due FIFO e se la DMA è configurata in modalità Master o Target. Il TAR è un registro a 32 bits in cui và specificato l'indirizzo iniziale di memoria del PC-Host da cui parte la transazione PCI PC-scheda. Sempre a 32 bits è il registro DCR. Quest'ultimo riporta lo stato e controlla l'attività della macchina a stati contenuta nel DMA del PCI-Core. Per un maggior chiarimento sulle funzioni dei registri TAR e DCR rimandiamo a (22).

Il funzionamento del DMA Controller è determinato da una macchina a stati interna che, una volta configurata, controlla tutte le transizioni. Il diagramma della macchina a stati è riportato in figura (6.1).

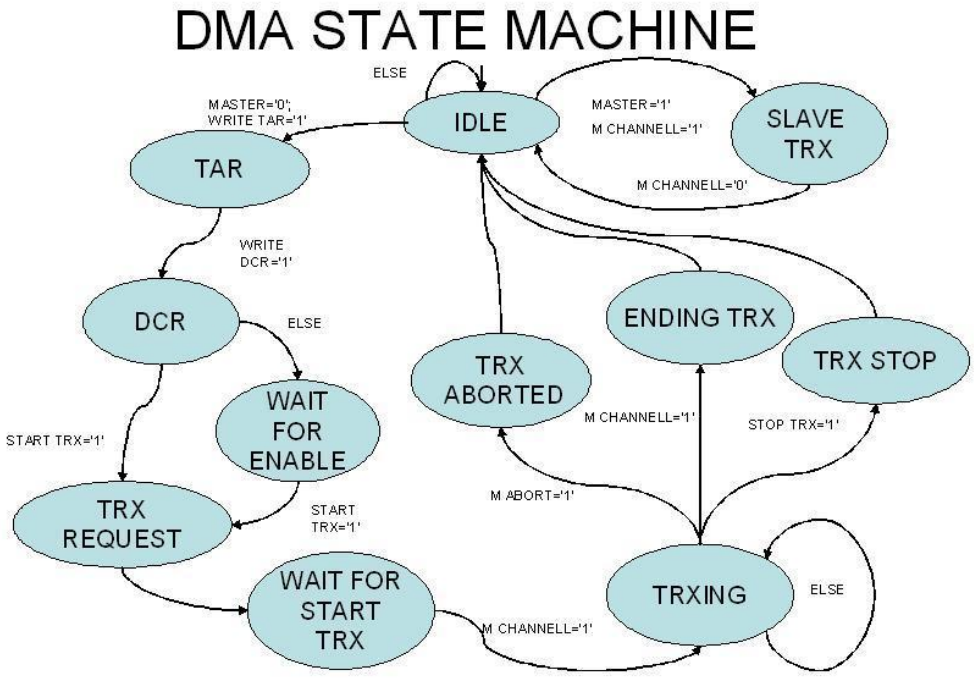


Figura 6.1: Diagramma DMA state machine

Partendo dallo stato iniziale di attesa ("idle"), la macchina a stati si comporta in modo differente a seconda della modalità di configurazione Target o Master del DMA. Nel primo caso se è richiesta una transizione di dati nel Bus la macchina si porta nello stato "slave trxing". Nel secondo caso invece se si sta configurando il registro TAR la macchina si porta in "write tar" e, una volta configurato il TAR, se si configura il registro DCR si passa in "write dcr". In questi stati, "slave trxing" o "write dcr", la macchina aspetta il comando di attivazione del passaggio START-TRX. Avvenuto questo, se è concesso l'utilizzo del Bus PCI ossia se è attivo il segnale M-CHANNELL, la macchina passa in "trxing" e attiva il passaggio dei dati da Bus a FIFO. In questo stato la macchina può essere arrestata dall'utente attraverso il segnale STOP-TRX o dall"'arbitro" del Bus attraverso il segnale M-ABORT, oppure la macchina termina il passaggio programmato attraverso il registro DCR e ritorna nello stato "idle".

6.5.2 MEMORY CONTROLLER

Il Memory Controller controlla il passaggio dei dati tra FIFO e memoria. Anche questo blocco, come il DMA, è costituito da più componenti:

- Memory Configuration
- Memory Control
- Fetcher.

Il Memory Configuration serve a programmare il funzionamento del Memory Controller attraverso la scrittura di due registri a 32 bits: MEMREG1 e MEMREG2.

Scrivendo il primo registro in modalità Target si può impostare la macchina in modalità RUN o SYSTEM, di cui ho spiegato il significato nel capitolo precedente. Attraverso questo registro, inoltre, si configura l'indirizzo della memoria "on-board" a partire dal quale saranno scritti o letti i dati.
Attraverso il registro MEMREG2 può essere controllato lo stato della macchina del Memory Controller e la quantità di dati trasferiti nell'ultimo passaggio di dati tra memoria e FIFO. Scrivendo un numero in codifica binaria nei primi sette bits di questo registro si imposta la quantità di dati da trasferire dalla FIFO di ingresso alla RAM in modalità SYSTEM.

Il memory Control è il blocco contenente la macchina a stati che, attraverso l'analisi dei dati, attiva o meno ogni trasferimento. La macchina a stati contenuta in esso è riportata in figura (6.2). Partendo dalla posizione iniziale "idle" segue percorsi diversi dipendentemente dalla modalità configurata.

Se la macchina è impostata in SYSTEM alla attivazione del segnale START TRX si porta in "init trx". A questo punto esistono due possibilità che dipendono dalla direzione che si vuole per i dati: "write" se si vuole scrivere la memoria, "read" se questa deve essere letta. In entrambe i casi viene scritto o letto un numero di dati pari a quello impostato nel MEMREG2. Terminato tale passaggio la macchina a stati torna in "idle".

Se invece si è impostato il sistema in RUN dallo stato "idle" la macchina passa nello stato "read header". Qui vengono letti gli ultimi due bit della Word proveniente dalla memoria che rendono possibile l'identificazione del tipo di Frame presente. Quindi abbiamo tre possibilità, rappresentate da tre diversi stati e determinate

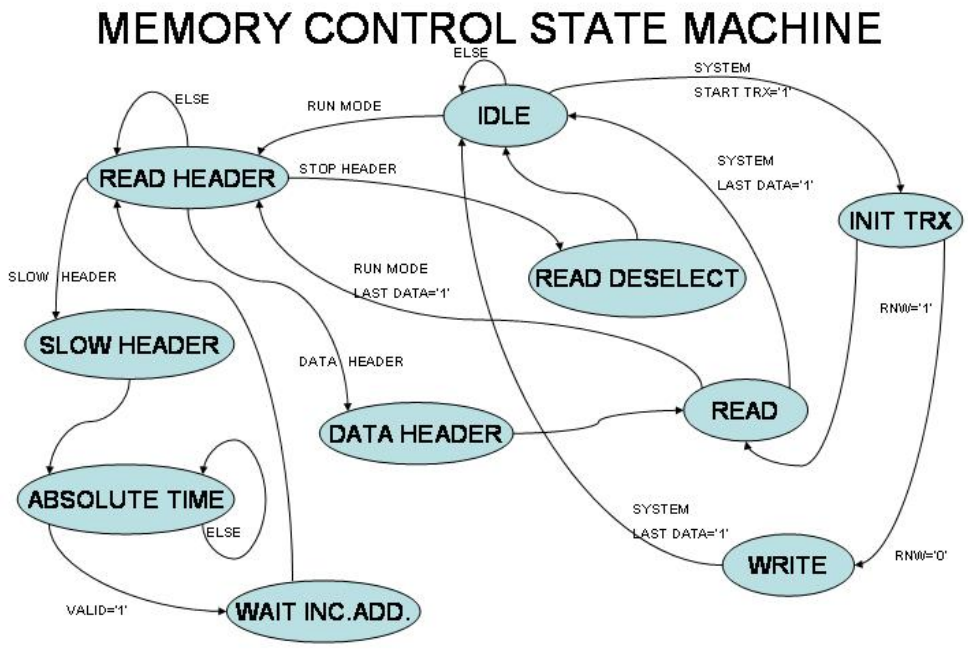


Figura 6.2: Macchina a stati del Memory Control

dai tre diversi tipi di Frame esistenti: "slow header", "data header" e "stop header". Nel primo caso si passa alla lettura del tempo assoluto scritto nella seconda Word ("absolute time") e, se il blocco Fetcher convalida attraverso il segnale VALID='1', si passa ("wait incadd") alla lettura di un nuovo dato in "read header". Se si presenta invece un Data Frame la macchina passa in "data header" e incrementando il puntatore alla memoria si porta in "read". In questo stato avviene la lettura di tutto il Data Frame, dopodiché si torna alla lettura del nuovo Header che è presente in memoria all'indirizzo successivo. Il meccanismo viene fermato quando si presenta uno Special Frame, in quel caso il sistema passa automaticamente in modalità SYSTEM, completando però regolarmente la lettura dell'intero Frame di dati. La macchina può essere fermata anche dall'utente che la può portare in modalità SYSTEM attraverso la scrittura del registro MEMREG1.

Il blocco Fetcher infine è quello che si occupa di fare il controllo sul tempo. É costituito da un contatore interno che misura il tempo locale. Il tempo assoluto viene prelevato dal Memory Control al quale spedisce poi il segnale di convalida a effettuare lo scambio dati tra RAM e FIFO.

6.5.3 Interrupt

La scheda prevede la possibilità di attivare o meno dei segnali di Interrupt. Questa attivazione e il controllo di tali segnali può essere fatto tramite un registro a 32 Bits appositamente dedicato e chiamato INTREG. I diversi tipi di Interrupt che possono essere inviati dalla scheda alla CPU sono otto:

1. Eccezione di scrittura della FIFO in ingresso già piena, l'operazione viene comunque regolarmente completata ma i dati sono ovviamente corrotti.

2. Eccezione di lettura della FIFO in ingresso vuota, anche quì la lettura viene comunque eseguita e i dati, formati da Word contenenti degli zero, vengono inviati in memoria.

3. Eccezione di scrittura della FIFO di uscita già piena, come al punto 1.

4. Eccezione di lettura dalla FIFO di uscita vuota, come al punto 2.

5. Eccezione di Slow Frame in FIFO di uscita; si alza ogni volta che il numero di Slow Frame presente nel registro NSFP passa dal valore '0' al valore '1'.

6. Eccezione di fine scrittura; si alza ogni volta che il Memory Controller ha terminato di trasferire i dati dalla FIFO di ingresso alla memoria in modalità SYSTEM.

7. Eccezione di TRX ABORT; si alza quando l'arbitro del Bus decide di interrompere lo scambio di dati della scheda sul Bus PCI.

8. Eccezione di TRX STOP; si alza ogni volta che l'utente decide di fermare il funzionamento della scheda in modalità RUN portandola in modalità SYSTEM.

Tutte le richieste di Interrupt vengono automaticamente annullate ogni volta che viene effettuata una lettura del registri INTREG.

6.5.4 Altri componenti

- REGISTRO. Questo è il componente del circuito in cui vengono convogliate tutte le informazioni del circuito, che sono organizzate in sette registri. Il significato di ogni singolo bit dei registri è riportato in appendice(B) insieme ai relativi indirizzi e valori di Default.

- ADDRESS SELECT.
  Questo componente è un Multiplexer, che traduce i diversi indirizzi che provengono dal segnale del Bus PCI-ADDR in altrettante uscite, ognuna delle quali si porta al valore '1' quando il corrispondente indirizzo è presente sul segnale.

- RESET.
  Questo blocco trasmette il segnale di Reset a tutti gli altri componenti. Il segnale di Reset può provenire dal bus PCI oppure essere attivato attraverso il "settaggio" ad '1' del trentunesimo bit del registro DMAREG.

- SWITCH.
  È il blocco preposto al controllo della modalità di funzionamento del Memory Control(SYSTEM o RUN).

- TRISTATE.
  É un interruttore che consente la lettura o la scrittura della memoria, oppure ne impedisce l'accesso.

Capitolo 7

Emulazione

7.1 Test dell'apparato

Questo paragrafo è dedicato alla descrizione dei test effettuati per verificare la correttezza del codice VHDL e del comportamento della scheda una volta programmata. Nel prossimo paragrafo invece è descritta la modalità con cui è stata eseguita la prova di emulazione dell'apparato di ricezione NEMO. Nel terzo paragrafo infine sono riportati i risultati della simulazione con alcune considerazioni riguardanti le prestazioni del sistema scheda-Bus-CPU.

La correttezza del codice è inizialmente controllata tramite un programma di compilazione e simulazione. La compilazione evidenzia eventuali errori di sintassi del codice, mentre attraverso la simulazione si possono emulare le risposte del sistema composto dalla particolare FPGA configurata con il codice di programmazione e del bus PCI, grazie all'utilizzo di una serie di "librerie e package" specifici. Il programma in questione è fornito dall'azienda Active e il suo nome è A-HDL. É possibile così simulare sia l'interfaccia scheda bus-PCI, sia il comportamento del PCI-Core e del Logic Backend

contenuti nella FPGA così come configurati dall'utente. Il valore aspettato di ogni segnale e variabile interna del concentratore e del bus PCI in funzione del tempo può essere visualizzato in una opportuna "finestra" grafica. Tramite un file in cui vengono specificati i segnali di "input" è anche possibile effettuare trasferimenti dati tra bus e scheda, rendendo così possibile la verifica funzionale del codice e la sua eventuale correzione nel caso in cui le risposte ottenute dall'emulatore non siano quelle aspettate.

Terminata la fase di simulazione, in caso di risposte soddisfacenti, si programma la scheda come descritto nei capitoli precedenti. Una volta programmata la scheda viene sistemata in un PC dotato di sistema operativo Linux Open Source. La scelta di utilizzare il sistema Linux è dovuta a più ragioni. Uno dei maggiori vantaggi che derivano dall'utilizzo di un sistema operativo Open Source è quello di avere a disposizione una libreria estremamente vasta di "pacchetti" Software, adatti a soddisfare le varie necessità che un utente può incontrare, tra i quali ambienti di sviluppo per il codice del driver. Il fatto che il sistema operativo sia Open Source rende inoltre possibile l'accesso al codice sorgente del sistema rendendolo così ottimizzabile per i propri scopi.

La scheda viene controllata da un "character device driver" appositamente scritto, che implementa le principali funzioni di I/O scheda-CPU permettendo la corretta integrazione della nostra scheda all'interno del sistema.

7.2 Emulazione

I dati utilizzati per le prove di emulazione sono stati ricavati da misure effettuate in sito durante la fase denominata NEMO R&D,

tramite dei moduli ottici simili a quelli previsti per la fase finale di NEMO. Questi dati sono stati poi elaborati così da assumere il formato richiesto dalla scheda di concentrazione, dopo la conversione ottico-elettrica. Una volta caricati i dati nella memoria interna alla scheda, questa viene portata in modalità RUN dal driver. I dati sono letti dal MEMORY CONTROLLER e spediti con il "timing" aspettato sul bus PCI e scritti poi in opportuni buffer a disposizione dell'utente. Tramite alcune funzioni di controllo fornite dal driver e visualizzando i dati ricevuti dalla scheda è stato possibile osservare che la scheda di concentrazione non commette errori nella lettura del tempo assoluto riportato nello Slow Frame e quindi nel tempo di invio dei dati sul Bus; inoltre è stata verificata la correttezza del trasferimento dati dalla memoria interna al "buffer utente". I risultati delle prove di simulazione e le conseguenti considerazioni sono riportate di seguito.

7.3 Risultati dei test e conclusioni

I test per la misura delle prestazioni della scheda sono stati effettuati inviando burst di dati composti da uno Slow Frame seguito da 100 Data Frame, ognuno contenente 5 word di dati, per un "data rate" complessivo di 602 word per burst corrispondente al data-rate medio aspettato da un piano della torre. I risultati ottenuti sono riportati di seguito.

7.3.1 Dati ottenuti

Prova 1	chiamate	tot.time	sing. time	min.,max.
Durata del trasf. di burst	1000	$33142\mu s$	$< 33\mu s >$	$32\mu s$, $45\mu s$
Tempo tra due S. F.in FIFO	999	$508273\mu s$	$< 508\mu s >$	$221\mu s$,$796\mu s$
Tempo di routine di Interrupt	2000	$15380\mu s$	$< 7\mu s >$	$6\mu s$,$18\mu s$

Prova 2	chiamate	tot.time	sing. time	min.,max.
Durata del trasf. di burst	1000	$33219\mu s$	$< 33\mu s >$	$32\mu s$, $45\mu s$
Tempo tra due S. F.in FIFO	999	$508275\mu s$	$< 508\mu s >$	$501\mu s$,$515\mu s$
Tempo di routine di Interrupt	2000	$15481\mu s$	$< 7\mu s >$	$6\mu s$,$16\mu s$

Prova 3	chiamate	tot.time	sing. time	min.,max.
Durata del trasf. di burst	1000	$33198\mu s$	$< 33\mu s >$	$32\mu s$, $46\mu s$
Tempo tra due S. F.in FIFO	999	$508268\mu s$	$< 508\mu s >$	$502\mu s$,$515\mu s$
Tempo di routine di Interrupt	2000	$15481\mu s$	$< 7\mu s >$	$6\mu s$,$15\mu s$

Nelle tabelle sono riportati i risultati di tre diverse prove, effettuate per verificare la riproducibilità dei risultati. La prima riga mostra il tempo impiegato dal sistema per trasferire l'intero contenuto di un burst composto, come già detto, da uno Slow Frame e i relativi Data Frame. La seconda riga mostra la misura del tempo che intercorre tra l'invio di due Slow Frame consecutivi che come aspettato è mediamente uguale alla durata di un burst. La terza riga riporta il tempo medio di gestione della scheda da parte della CPU. Le colonne indicano il numero di chiamate ricevute dalla CPU da parte della scheda, il tempo totale impiegato per effettuare il trasferimento e il tempo medio di un singolo trasferimento (ricavato dal tempo totale per effettuare N trasferimenti diviso il numero N dei trasferimenti effettuati).

Dai dati mostrati si nota che la CPU è occupata in media per $2*7\mu s = 14\mu s$ (start dma+end dma), se si eliminano però i Monitor relativi alle due prime righe, riducendo quindi i compiti della CPU, il tempo medio di impiego passa da $14\mu s$ a $8\mu s$, come risulta da successive misure non riportate. Il tempo di impiego della CPU è però sottostimato in quanto non si tiene conto del tempo necessario alla CPU per attivare la routine di interrupt da parte del Kernel. Il calcolo totale, sovrastimato, può essere ottenuto sottraendo al tempo di durata di un trasferimento completo di un burst, pari a $33\mu s$, il tempo impiegato dalla scheda per trasferire 602 word, che avviene alla frequenza di clock 33MHz pari a $18\mu s$:

$$(33 - 18)\mu s = 15\mu s$$

. La CPU, quindi, lavora fino ad un massimo di $15\mu s$ ogni $512\mu s$ (intervallo di tempo fra l'arriva di uno Slow Frame e il successivo) pari a $\sim 2.9\%$ del tempo totale. Aggiungendo anche il tempo necessario alla CPU per trasferire i dati dal Kernel Space allo User Space pari circa (sovrastimato) a $2\mu s$ per un Bus a $400MHz$, otteniamo $17\mu s$ di impiego su $512\mu s$ pari a $\sim 3.3\%$ del tempo a disposizione.

Da queste misure risulta chiaramente che, con l'attuale modalità di trasferimento dei dati, il carico computazionale dovuto alla gestione della scheda è minimo e quindi è possibile integrare in una singola CPU l'elettronica di "read-out" di più piani. Una importante osservazione da fare è che il tempo necessario per l'invio dei dati e la successiva scrittura nello User Space può essere ridotto, considerando che i componenti (commerciali) utilizzati per le misure riportate non sono di ultima generazione (P III 800MHz, FSB 133MHz). Il tempo totale di trasferimento è dato dalla somma del tempo necessario ai dati per attraversare il Bus-PCI (Trx time) e

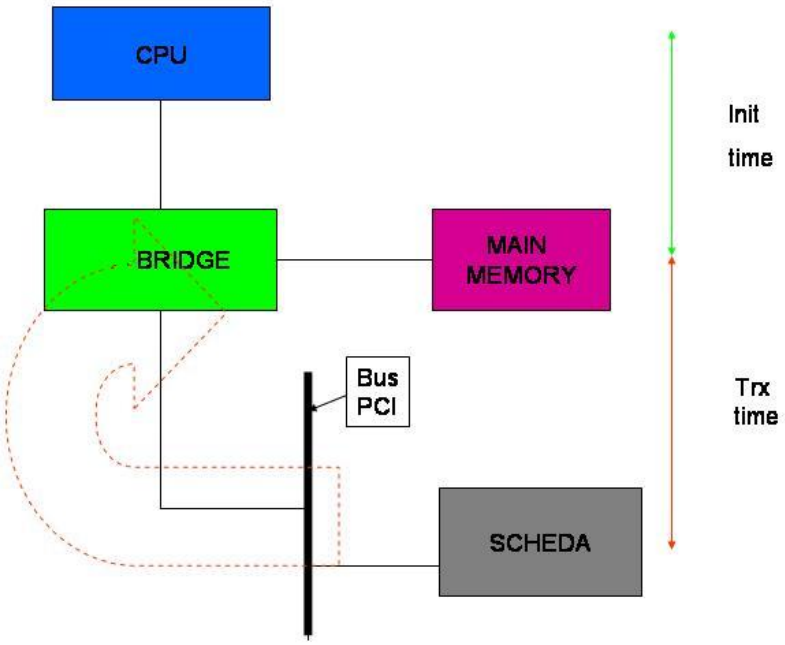


Figura 7.1: Architettura di comunicazione CPU-scheda

dalla latenza inserita dalla CPU per inizializzare il trasferimento (Init Time), come riportato in figura7.1. Il tempo di trasferimento dei dati può essere quasi dimezzato raddoppiando la dimensione dei dati trasferiti, passando dal trasferimento di Word a 32 bit a Word a 64 bit, funzione implementabile anche sulla piattaforma utilizzata. Un altro modo per ridurre il Trx time è quello di aumentare la frequenza con cui lavora il Bus-PCI: il bus utilizzato nei test funziona con un clock a 33MHz, mentre esistono in commercio Bus-PCI che lavorano a frequenze doppie o anche maggiori (66-133Mhz). Per ridurre il tempo di inizializzazione invece si possono utilizzare CPU più veloci: sono disponibili infatti CPU che lavorano a frequenze maggiori di 3GHz, oppure è possibile utilizzare chipset più veloci che aumentano la velocità di comunicazione tra i vari componenti. In conclusione, tramite adozione di piattaforme di calcolo di ultima generazione, potremmo avere rendimenti molto migliori di quelli attualmente misurati. Ciononostante, e come già detto, i dati ottenuti sono incoraggianti e permettono di affermare che anche con gli attuali componenti è possibile l'integrazione sulla singola CPU di canali provenienti da più piani e quindi la realizzazione di un sistema completo ed efficiente di emulazione del sistema di read-out.

Bibliografia

[1] K.Greisen(1966) *Phys.Rev.Lett.* 16,748.

[2] G.T. Zatsepin & V.A.Kuz'min(1966) JEPT *Lett.* 4,78.

[3] T.K. Gaisser, F. Halzen & T. Stanev (1994)*astro-ph/9410384*

[4] M. Vietri (1998) *Phys. Rev. Lett. 80, 3690*

[5] A. Dar & A. De Rújula (2000) *astro-ph/0012227*

[6] Learned JG e Mannheim K. *High Energy Neutrino Astrophysics.* 2000, Annu.Rev.Nucl.Part.

[7] E. Waxman & J. Bachall (1997) *Phys. Rev. Lett. 78, 2292*

[8] Markov MA. *Annual Int. Conf. on High Energy Physics.* Proc. (1960 Rochester NY), e EGS Judarshan et al.

[9] *http://www.phys.hawaii.edu/dmnd/dumand.html*

[10] *http://www.ith.de/baikal/baikalhome.html*

[11] *http://amanda.physics.wisc.edu*

[12] *http://nemowen.lns.infn.it*

[13] *http://www.nestor.org.gr/intro/main.htm*

[14] The ANTARES Collaboration (1999) *A Deep Sea Telescope for High Energy Neutrinos astro-ph/9907432*

[15] E Migneco. *The NEMO Project.* Catania June 2004, INFN LNS.

[16] A. Capone et al. (2000) *Measurement of deep seawater optical proprties with AC9 trasmissometer* N.I.M.-A487, 423-434.

[17] Meeting NEMO LNS, 2/12/2003 *Calibrazione temporale in NEMO PHASE-1* M.Circella.

[18] M. Bonori. *Meeting NEMO.* LNS, 17-11-2004.

[19] M. Bonori. *I.N.F.N. sez. Roma1.* Catania 1/3-12-2003.

[20] F.Ameli. *Synchronous Data Trasmission Protocol for NEMO experiment* VLVνT Amsterdam 5-8 October 2003.

[21] R. Concimano *VLVν* T Workshop, NIKHEF Amsterdam, 5-8 October 2003

[22] PCI Core Getting Started Guide. *PLD Applications.*

[23] PCISYS User's Guide. *PLD Applications.*

[24] Piero Vicini. *NEMO: una proposta per l'elettronica di DAQ e del trigger a riva.* I.N.F.N. NEMO meeting 12/2003.

[25] T. K. Gaisser*astro-ph/9707283*

Appendice A

Registri

- DMAREG.
  BIT(0) modalità del dma, che può funzionare come Master se il bit è a '0', oppure, settando questo bit a '1' è impostata come Target.
  BIT(1) se a '1' attiva la macchina a stati della DMA.
  BIT(2) disattiva la macchina a stati anche se il passaggio è in atto, che però viene regolarmente terminato
  BIT(3) va ad '1' ogni volta che il dma ha terminato il passaggio (bit solo di lettura)
  BITS(4-25) DMA STATUS: (solo lettura)
  (4) nuovo passaggio richiesto.
  (5) passaggio in corso.
  (6) passaggio finito.
  (7) sempre a '0'.
  (8-15) Numero di dati, in codifica binaria,presenti nella FIFO di ingresso.
  (16-25))Numero di dati, in codifica binaria, presenti nella FIFO di uscita.

BITS(26) (solo lettura) Se a '1' vuol dire che l'arbitro ha appena interrotto il passaggio dei dati.
BITS(27-30) tutti zero.
BIT(31) RESET, Portando questo bit ad '1' si attiva appunto la funzione di reset.

- TAR, Target Address Register.

- DCR, Direct control Memory.

- MEMREG1.
  BIT(0) se è a '0' il Memory Control è impostato in modalità SYSTEM, altrimenti in RUN.
  BIT(1) se si è in SYSTEM, settato a '1' attiva il passaggio di dati tra FIFO e memoria.
  BIT(2) sempre in modalità SYSTEM specifica se il passaggio deve avvenire da FIFO a memoria ('0') oppure da memoria a FIFO('1').
  BIT(3) se portato ad '1' ferma l'attività della macchina del Memory Control portando la scheda da modalità RUN a SYSTEM.
  BITS(4-23) specificano l'indirizzo della memoria RAM alla quale i dati vanno letti o scritti.
  BITS(24-31) tutti a zero.

- MEMREG2
  BITS(7-0) Qui và scritto il numero di Word che si vuole tra-

sferire dalla FIFO di ingresso alla RAM.
BITS(13-8) riporta il numero di Word trasferite null'ultimo passaggio di dati tra RAM e FIFO(solo lettura).
BITS(15-14)stato Memory Control(solo lettura): se il bit(14) è a '1' c'è la richiesta di un passaggio, se invece è a '1' il bit(15) indica che il passaggio stà avvenendo altrimenti il passaggio è finito.
BITS(31-16) tutti a zero.

- NUMSLOWFRAME.
  BITS(0-4) riporta il numero di Slow Frame presenti nella FIFO di uscita, se viene scritto un '1' nel bit zero di questo registro, tale numero viene decrementato di uno.
  BITS(30-5) tutti a zero.
  BITS(31) il Memory Control ha individuato uno Stop Frame(solo lettura).

- INTERRUPT REGISTER.
  BITS(8-0) indicano se sono attivi o meno i nove tipi di Interrupt.
  BITS(17-9) INTERRUPT VECTOR, indica quale tipo di interrupt è attivo.

I valori di Default di questi registri sono riporati nella tabella seguente:

TrEmu memory map.

Register	Address (BAR0)	Size	Access Type	Reset Value
DMAREG	0000(0h)	32	R\W	0000000
TAR	0001(4h)	32	R\W	0000000
DCR	0010(8h)	32	R\W	0000000
MEMREG1	0011(ch)	32	R\W	0000000
MEMREG1	0100(10h)	32	R\W	0000000
NSFP	0101(14h)	32	R\W	0000000
INTREG	0110(18h)	32	R\W	0000000

Figura A.1: Registri

Appendice B

Schema a blocchi del Logic Backend

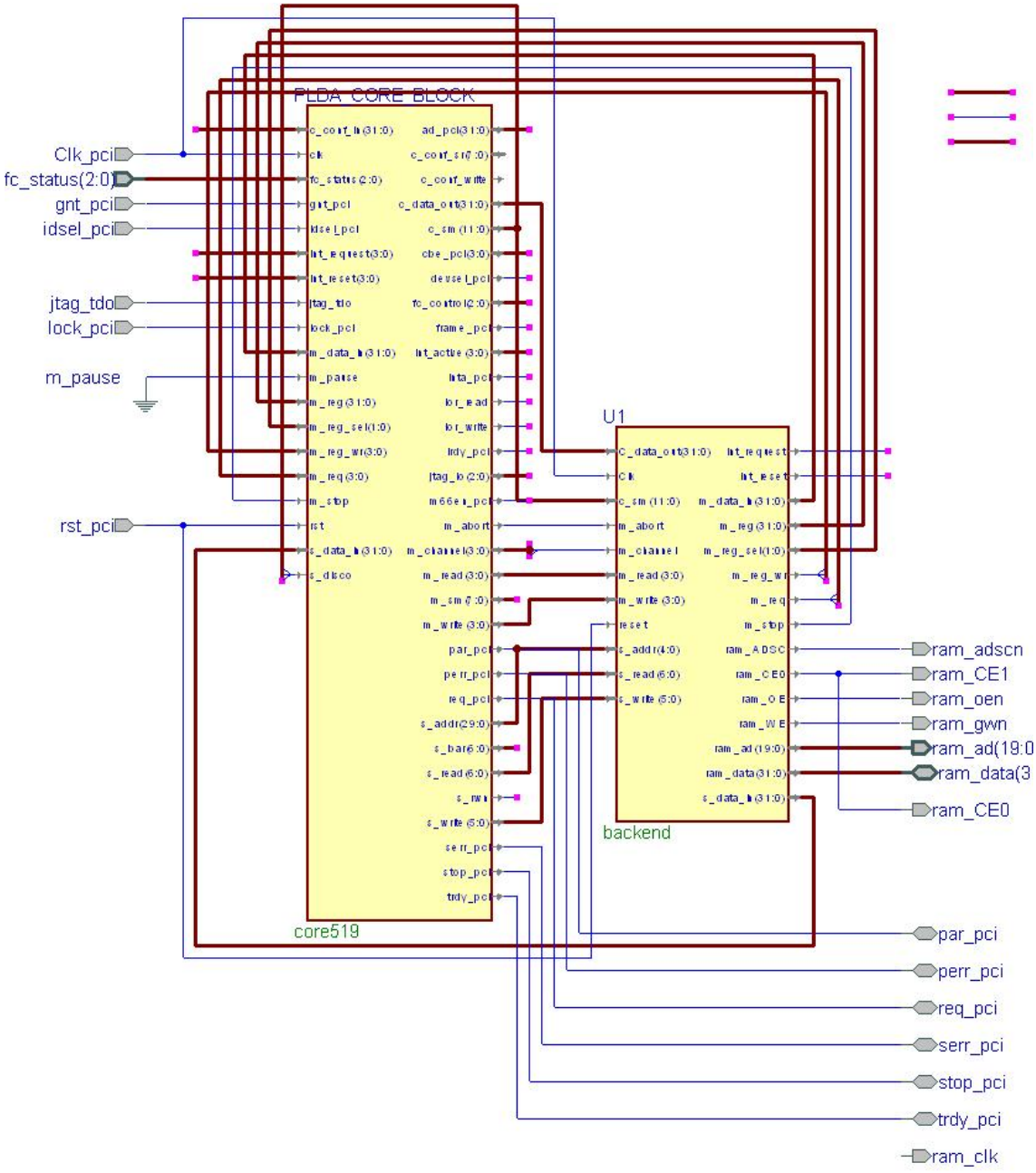
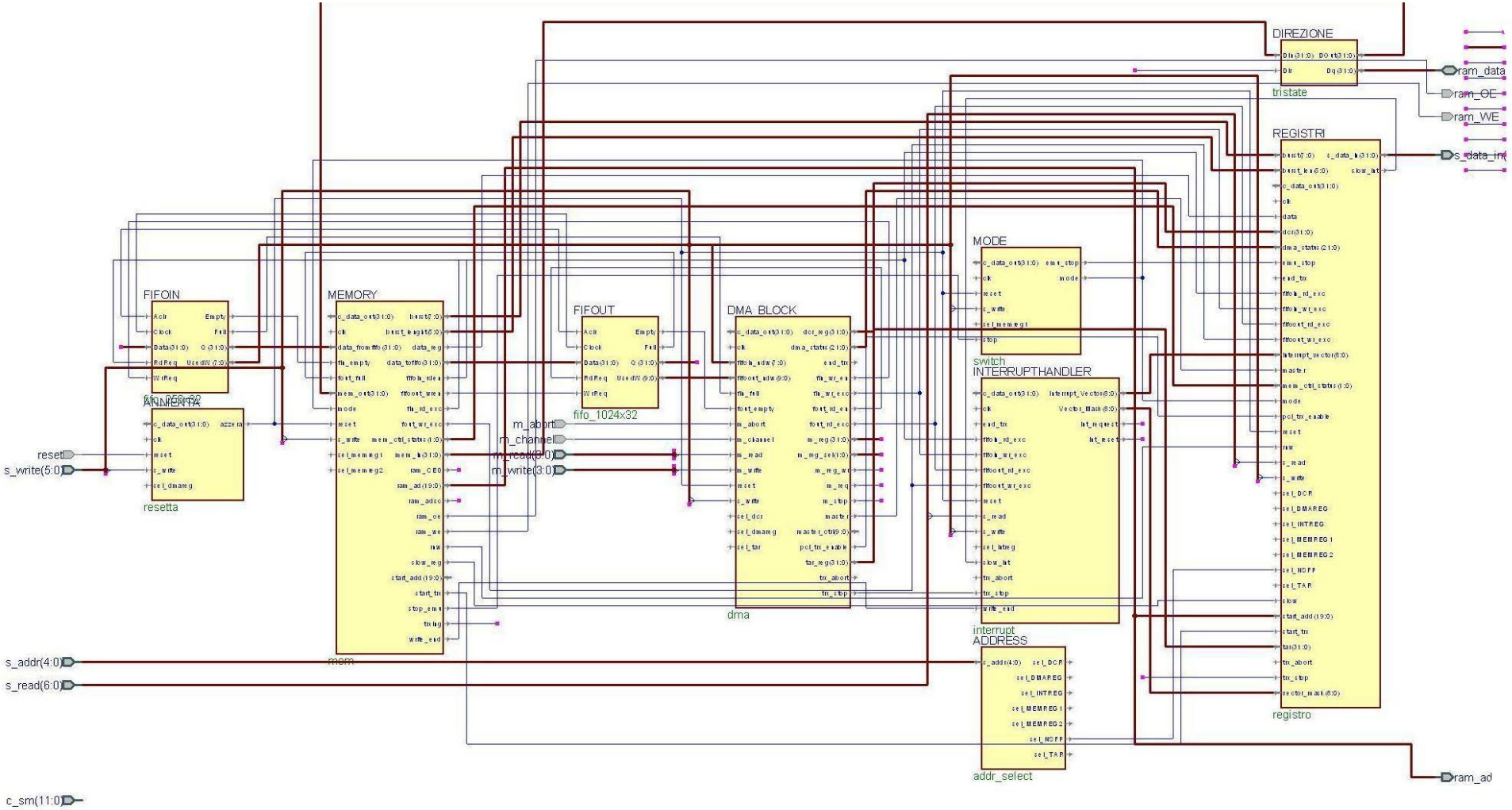


Figura B.1: Top level


DIREZIONE
tristate
ram_data
ram_OE
ram_WE
REGISTRI
registro
MODE
switch
INTERRUPTHANDLER
interrupt
ADDRESS
addr_select
FIFOIN
MEMORY
FIFOUT
fifo_1024x32
DMA_BLOCK
dma
resetta
reset
s_write(5:0)
s_addr(4:0)
s_read(6:0)
c_sm(11:0)
m_abort
m_channel
m_read(3:0)
m_write(3:0)
ram_ad

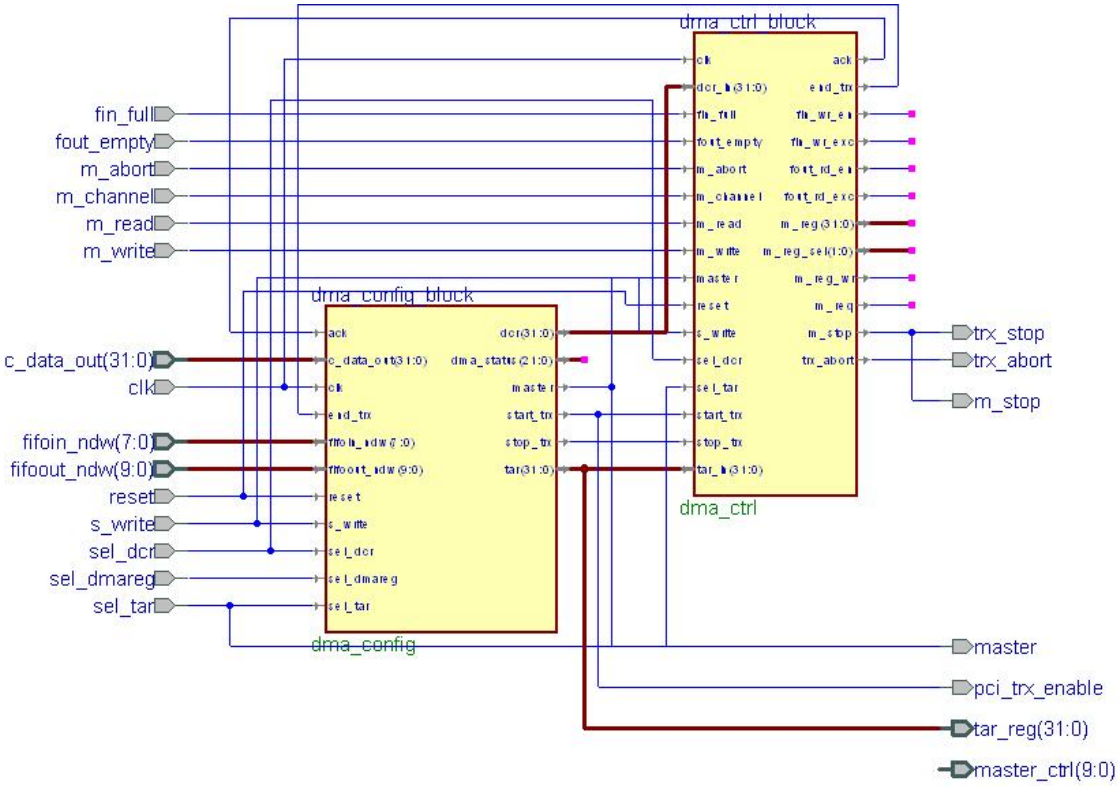


Figura B.3: DMA

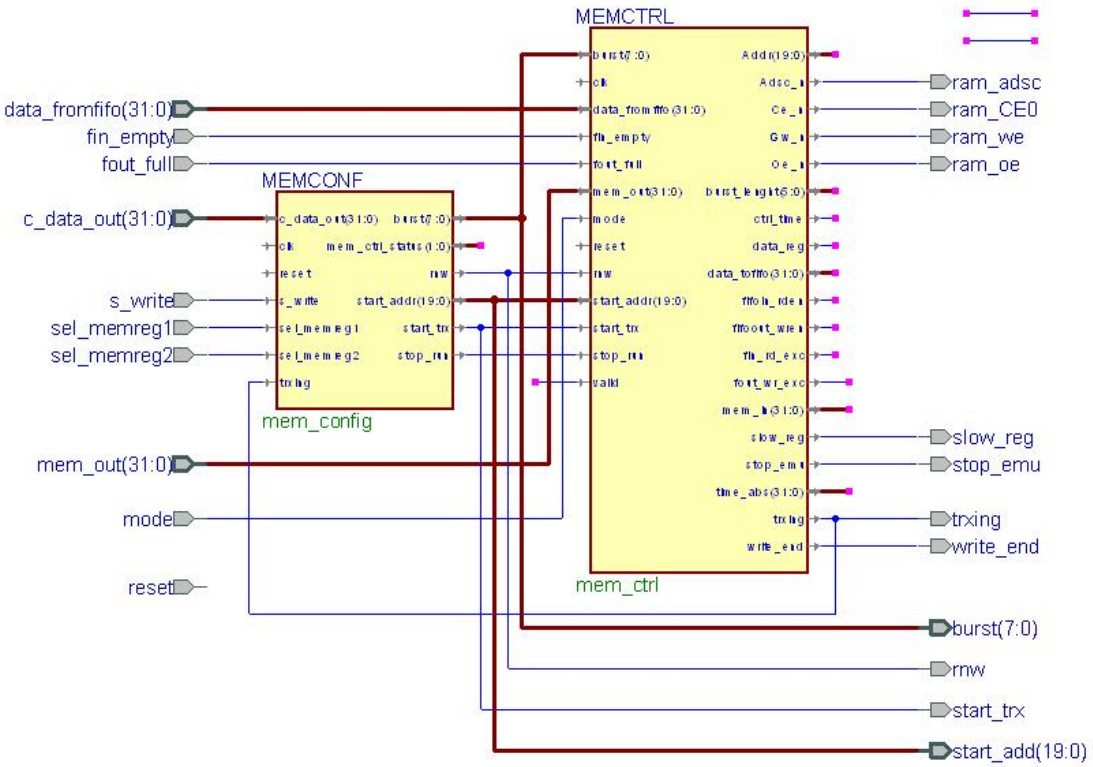


Figura B.4: MEMORY CONTROLLER

Appendice C

Il Codice

Trigger.

```
PACKAGE trigger_package IS
---dma_config
constant addr_DMAREG                : std_logic_vector(4 downto 0):="00000"; -- h0000000 constant addr_TAR
: std_logic_vector(4 downto 0):="00001"; -- h0000004 constant addr_DCR                  : std_logic_vector(4
downto 0):="00010"; -- h0000008
----memory controller
constant addr_MEMREG1               : std_logic_vector(4 downto 0):="00011"; -- h000000c constant addr_MEMREG2
: std_logic_vector(4 downto 0):="00100"; -- h0000010
constant addr_NSFP                  : std_logic_vector(4 downto 0):="00101"; -- h0000014 constant addr_INTREG
: std_logic_vector(4 downto 0):="00110"; -- h0000018 end package trigger_package;
```

Address select.

```
entity addr_select is
    port(
    s_addr      :               in std_logic_vector(4 downto 0);
    sel_DMAREG:                 out std_logic;
    sel_TAR:                    out std_logic;
    sel_DCR:                    out std_logic;
    sel_MEMREG1:                out std_logic;
    sel_MEMREG2:                out std_logic;
    sel_NSFP:                   out std_logic;
    sel_INTREG:                 out std_logic
        );
end entity addr_select;
architecture behavioral of addr_select is
begin
    sel_TAR          <='1' when (s_addr=addr_TAR) else '0';
    sel_DCR          <='1' when (s_addr=addr_DCR) else '0';
    sel_DMAREG       <='1' when (s_addr=addr_DMAREG) else '0';
    sel_MEMREG1      <='1'when  (s_addr=addr_MEMREG1) else '0';
    sel_MEMREG2      <='1'when   s_addr=addr_MEMREG2 else '0';
    sel_NSFP         <='1' when (s_addr=addr_NSFP) else '0';
    sel_INTREG       <='1' when  s_addr=addr_INTREG else '0';
end BEHAVIORAL;
```

Interrupt.

```
 entity interrupt is port ( clk:                    in std_logic; reset: in
std_logic; sel_intreg:             in std_logic; s_write:               in std_logic; s_read: in std_logic;
c_data_out: in std_logic_vector(31 downto 0); fifoout_rd_exc:         In std_logic; fifoout_wr_exc:         In
std_logic; fifoin_rd_exc:          In std_logic; fifoin_wr_exc:         In std_logic; end_trx:
In std_logic; trx_stop:               In std_logic; trx_abort: In std_logic; slow_int:               in
std_logic; write_end:              in std_logic; int_reset: Out std_logic; int_request:            Out
std_logic; Vector_Mask:            Out std_logic_vector(8 downto 0); Interrupt_Vector:       Out
std_logic_vector(8 downto 0)); end interrupt;
architecture behavioral of interrupt is signal MASK: std_logic_vector(8 downto 0); signal INT_REG:
std_logic_vector(8 downto 0); signal temp_int_request: std_logic; begin
```

```
MASK_REGISTER: process(clk,reset) begin
    if(reset='0') then
        MASK <= (others => '0'); -- all not masked
    elsif(clk'event and clk = '1') then
        if(s_write = '1' and sel_intreg= '1') then
            MASK <= c_data_out(8 downto 0);
        end if;
    end if;
end process;
INTERRUPT_REGISTER: process(clk,reset) begin
    if(reset = '0') then
        INT_REG <= (others => '0');
    elsif(clk'event and clk = '1') then
        if int_reg(0)='0' then
        int_reg(0)<=slow_int;
        elsif int_reg(0)='1' then
            if s_read='1' and sel_intreg='1' then
            int_reg(0)<='0';
            else
            int_reg(0)<='1';
            end if;
            end if;
        if int_reg(4)='0' then
        int_reg(4)<=fifoin_rd_exc;
        elsif int_reg(4)='1' then
            if s_read='1' and sel_intreg='1' then
            int_reg(4)<='0';
            else
            int_reg(4)<='1';
            end if;
            end if;
        if int_reg(5)='0' then
        int_reg(5)<=fifoin_wr_exc;
        elsif int_reg(5)='1' then
            if s_read='1' and sel_intreg='1' then
            int_reg(5)<='0';
            else
            int_reg(5)<='1';
            end if;
            end if;
        if int_reg(6)='0' then
        int_reg(6)<=fifoout_rd_exc;
        elsif int_reg(6)='1' then
            if s_read='1' and sel_intreg='1' then
            int_reg(6)<='0';
            else
            int_reg(6)<='1';
            end if;
            end if;
        if int_reg(7)='0' then
        int_reg(7)<=fifoout_wr_exc;
        elsif int_reg(7)='1' then
            if s_read='1' and sel_intreg='1' then
            int_reg(7)<='0';
            else
            int_reg(7)<='1';
            end if;
            end if;
        if int_reg(1)='0' then
        int_reg(1)<=end_trx;
        elsif int_reg(1)='1' then
            if s_read='1' and sel_intreg='1' then
            int_reg(1)<='0';
            else
            int_reg(1)<='1';
            end if;
            end if;
        if int_reg(8 )='0' then
        int_reg(8)<=write_end;
        elsif int_reg(8)='1' then
            if s_read='1' and sel_intreg='1' then
            int_reg(8)<='0';
            else
            int_reg(8)<='1';
            end if;
        end if;
        INT_REG(2) <= TRX_STOP;
```

```
        INT_REG(3) <= TRX_ABORT;
         end if;
end process;
INTERRUPT_REQUEST: process(clk,reset) begin
    if(reset = '0') then
        temp_int_request <= '0';
        int_reset <= '0';
    elsif(clk'event and clk = '1') then
        if(sel_intreg = '1' and s_read = '1') then -- se leggo il vector interrupt resetto la richiesta e alzo il res
            int_reset <= '1';
            temp_int_request <= '0';
        else
            int_reset <= '0';
            temp_int_request <= (INT_REG(0) and not(MASK(0))) or
                                (INT_REG(1) and not(MASK(1))) or
                                (INT_REG(2) and not(MASK(2))) or
                                (INT_REG(3) and not(MASK(3))) or
                                (INT_REG(4) and not(MASK(4))) or
                                (INT_REG(5) and not(MASK(5))) or
                                (INT_REG(6) and not(MASK(6))) or
                                (INT_REG(7) and not(MASK(7))) or
                                (INT_REG(8) and not(MASK(8)));
        end if;
    end if;
end process INTERRUPT_REQUEST;
Vector_Mask <= MASK; Interrupt_Vector <= INT_REG; int_request <= temp_int_request;
--------------------------------------------------------------------------------
```

Tristate.

```
ENTITY TRISTATE IS
    Port(
        DIn: In std_logic_vector(31 downto 0);
        DOut: Out std_logic_vector(31 downto 0);
        Dir: In std_logic;
        Dq: InOut std_logic_vector(31 downto 0));
end TRISTATE;
architecture behavioral of TRISTATE is begin
    process(DIn,Dir)
    begin
        if(Dir = '0') then
            Dq <= DIn;
        else
            Dq <= (others => 'Z');
        end if;
    end process;
    DOut <= Dq;
end behavioral;
```

Reset.

```
entity resetta is
    port (
    clk:        in std_logic;
    reset:      in std_logic;
    sel_dmareg: in std_logic;
    s_write:    in std_logic;
    c_data_out: in std_logic_vector(31 downto 0);
    azzera:     out std_logic
    );
end resetta; architecture  behavioral of resetta is signal azz: std_logic; begin
CANCELLA:   process(clk,reset) begin if reset='0' then
 azz<='1';
elsif clk'event and clk='1'then
 if s_write='1'and sel_dmareg='1'then
 azz<=c_data_out(31);
 else
 azz<='0';
 end if;
 end if;
 end process;
 azzera<=not(azz);--poichè il reset è attivo "basso"
 end behavioral;
```

Switch.

```
entity switch is
    port (
    clk:        in std_logic;
```

```
    reset:      in std_logic;
    sel_memreg1:in std_logic;
    s_write:    in std_logic;
    stop:       in std_logic;
    c_data_out: in std_logic_vector(31 downto 0);
    emu_stop:   out std_logic;
    mode:       out std_logic
    );
end switch;
architecture behavioral of switch is signal tmode:std_logic; begin process(clk,reset) begin if reset='0't
    tmode<='0';
elsif clk='1'and clk'event then
    if s_write='1'and sel_memreg1='1'then
        tmode<=c_data_out(0);
    elsif stop='1' then
        tmode<='0';
    end if;
end if; end process; mode<=tmode;
REGISTRO:   process(clk,reset) begin if reset='0'then
    emu_stop<='0';
elsif clk='1'and clk'event then if tmode='1'then if stop='1'then
    emu_stop<='1';
elsif tmode='1'then
    emu_stop<='0';
    end if;
    end if;
    end if;
    end process;
end behavioral;
```

Fifo out.

```
 entity fifo_1024x32 is
     port (
             Data : in std_logic_vector(31 downto 0);
             Clock : in std_logic;
             WrReq : in std_logic;
             RdReq : in std_logic;
             Aclr : in std_logic;
--             Sclr : in std_logic;
             Full : out std_logic;
             Empty : out std_logic;
             Q : out std_logic_vector(31 downto 0);
             UsedW : out std_logic_vector(9 downto 0)
     );
    -- pragma translate_off
    function int2bin(int : IN integer; binsize : IN integer) return std_logic_vector is
    variable result : std_logic_vector((binsize-1) downto 0);
    variable tmp : integer;
    begin
       tmp := int;
       for i in 0 to (binsize-1) loop
           if (tmp mod 2 = 1) then
              result(i) := '1';
           else
              result(i) := '0';
           end if;
           tmp := tmp/2;
       end loop;
    return result;
    end int2bin;
    -- pragma translate_on
  end fifo_1024x32;
architecture behavior of fifo_1024x32 is
   constant Width : integer := 32;
   constant WidthAd : integer := 10;
   constant NumWords   : integer := 1024;
   signal empty_tmp: std_logic := '1';
   signal full_tmp: std_logic := '0';
   signal status_ptr: integer := 0;
   signal Sclr: std_logic;
begin
   -- pragma translate_off
   inreg: process(clock, Aclr)
   TYPE t_mem_data IS ARRAY(0 to NumWords-1) of std_logic_vector(Width - 1 downto 0);
   variable mem_data: t_mem_data;
   variable mem_init: boolean := false;
   variable read_ptr, write_ptr, word_ctr : integer;
```

```
   begin
--   Aclr <= '0';
   sclr <= '0';
       if NOT(mem_init) then
         for i in mem_data'LOW to mem_data'HIGH loop
            mem_data(i) := (OTHERS => 'X');
         end loop;
         mem_init := TRUE;
      end if;
      if (Aclr = '1') then
         Q <= (OTHERS => '0');
         read_ptr := 0;
         write_ptr := 0;
         word_ctr := 0;
         status_ptr <= 0;
      elsif (clock'event and clock = '1' and clock'last_value = '0') then
         if (Sclr /= '1') then
            if ((WrReq and (not full_tmp)) = '1') then
               mem_data(write_ptr) := Data;
               write_ptr := (write_ptr + 1) mod NumWords;
               word_ctr := word_ctr + 1;
               if (empty_tmp = '1') then
                  Q <= mem_data(read_ptr);
               end if;
            end if;
            if ((RdReq and (not empty_tmp)) = '1') then
               read_ptr := (read_ptr + 1) mod NumWords;
               word_ctr := word_ctr - 1;
               if (word_ctr = 0) then
                  Q <= (OTHERS => 'X');
               else
                  Q <= mem_data(read_ptr);
               end if;
            end if;
            status_ptr <= word_ctr;
         else
            Q <= (OTHERS => '0');
            read_ptr := 0;
            write_ptr := 0;
            status_ptr <= 0;
            word_ctr := 0;
         end if;
       end if;
   end process;
   status_flag: process(status_ptr)
   begin
     if (status_ptr = 0) then
        empty_tmp <= '1';
        full_tmp <= '0';
     elsif (status_ptr = NumWords) then
        empty_tmp <= '0';
        full_tmp <= '1';
     else
       full_tmp <= '0';
       empty_tmp <= '0';
     end if;
   end process;
   empty <= empty_tmp;
   full <= full_tmp;
   UsedW <= int2bin(status_ptr, WidthAd);
end behavior;
```

Fifo in.

```
  entity fifo_256x32 is
     port (
             Data : in std_logic_vector(31 downto 0);
             Clock : in std_logic;
             WrReq : in std_logic;
             RdReq : in std_logic;
             Aclr : in std_logic;
--             Sclr : in std_logic;
             Full : out std_logic;
             Empty : out std_logic;
             Q : out std_logic_vector(31 downto 0);
             UsedW : out std_logic_vector(7 downto 0)
     );
```

```
    -- pragma translate_off
    function int2bin(int : IN integer; binsize : IN integer) return std_logic_vector is
    variable result : std_logic_vector((binsize-1) downto 0);
    variable tmp : integer;
    begin
       tmp := int;
       for i in 0 to (binsize-1) loop
           if (tmp mod 2 = 1) then
              result(i) := '1';
           else
              result(i) := '0';
           end if;
           tmp := tmp/2;
       end loop;
    return result;
    end int2bin;
    -- pragma translate_on
  end fifo_256x32;
architecture behavior of fifo_256x32 is
   constant Width : integer := 32;
   constant WidthAd : integer := 8;
   constant NumWords   : integer := 256;
   signal empty_tmp: std_logic := '1';
   signal full_tmp: std_logic := '0';
   signal status_ptr: integer := 0;
   signal Sclr: std_logic;
begin
   -- pragma translate_off
   inreg: process(clock, Aclr)
   TYPE t_mem_data IS ARRAY(0 to NumWords-1) of std_logic_vector(Width - 1 downto 0);
   variable mem_data: t_mem_data;
   variable mem_init: boolean := false;
   variable read_ptr, write_ptr, word_ctr : integer;
   begin
--   Aclr <= '0';
   sclr <= '0';
       if NOT(mem_init) then
         for i in mem_data'LOW to mem_data'HIGH loop
            mem_data(i) := (OTHERS => 'X');
         end loop;
         mem_init := TRUE;
      end if;
      if (Aclr = '1') then
         Q <= (OTHERS => '0');
         read_ptr := 0;
         write_ptr := 0;
         word_ctr := 0;
         status_ptr <= 0;
      elsif (clock'event and clock = '1' and clock'last_value = '0') then
         if (Sclr /= '1') then
            if ((WrReq and (not full_tmp)) = '1') then
               mem_data(write_ptr) := Data;
               write_ptr := (write_ptr + 1) mod NumWords;
               word_ctr := word_ctr + 1;
               if (empty_tmp = '1') then
                  Q <= mem_data(read_ptr);
               end if;
            end if;
            if ((RdReq and (not empty_tmp)) = '1') then
               read_ptr := (read_ptr + 1) mod NumWords;
               word_ctr := word_ctr - 1;
               if (word_ctr = 0) then
                  Q <= (OTHERS => 'X');
               else
                  Q <= mem_data(read_ptr);
               end if;
            end if;
            status_ptr <= word_ctr;
         else
            Q <= (OTHERS => '0');
            read_ptr := 0;
            write_ptr := 0;
            status_ptr <= 0;
            word_ctr := 0;
         end if;
      end if;
   end process;
   status_flag: process(status_ptr)
```

```
   begin
     if (status_ptr = 0) then
        empty_tmp <= '1';
        full_tmp <= '0';
     elsif (status_ptr = NumWords) then
        empty_tmp <= '0';
        full_tmp <= '1';
     else
       full_tmp <= '0';
       empty_tmp <= '0';
     end if;
   end process;
   empty <= empty_tmp;
   full <= full_tmp;
   UsedW <= int2bin(status_ptr, WidthAd);
   -- pragma translate_on
end behavior;
```

Dma control.

```
entity dma_ctrl is
    port (
    clk:            in std_logic;
    reset:          in std_logic;
    m_write:        in std_logic;
    m_read:         in std_logic;
    m_abort:        in std_logic;
    m_channel:      in std_logic;
    s_write:        in std_logic;
    sel_tar:        in std_logic;
    sel_dcr:        in std_logic;
    start_trx:      in std_logic;
    tar_in:         in std_logic_vector(31 downto 0);
    dcr_in:         in std_logic_vector(31 downto 0);
    fin_full:       in std_logic;
    fout_empty:     in std_logic;
    master:         in std_logic;
    stop_trx:       in std_logic;
    fin_wr_en:      out std_logic;
    fout_rd_en:     out std_logic;
    fin_wr_exc:     out std_logic;
    fout_rd_exc:    out std_logic;
    ack:            out std_logic;
    m_req:          out std_logic;
    m_reg:          out std_logic_vector(31 downto 0);
    m_reg_sel:      out std_logic_vector(1 downto 0);
    m_reg_wr:       out std_logic;
    end_trx:        out std_logic;
    trx_abort:      out std_logic;--per interrupt handler
    m_stop:         out std_logic);
end dma_ctrl;
architecture behavioral of dma_ctrl is type dma_state_type is
(idle,slave_trxing,write_tar,write_dcr,wait_for_enable,trx_request,
wait_for_start_trx,trxing,ending_trx,trx_stop,trx_aborted); signal msm_state : dma_state_type; signal
tfin_wr_en,tfout_rd_en: std_logic; begin
---fifo signal
    fin_wr_en<=tfin_wr_en;
    fout_rd_en<=tfout_rd_en;
    tfin_wr_en<=m_read when msm_state=trxing else '0';
    tfout_rd_en<=m_write when msm_state=trxing else '0' ;
---exception
    fin_wr_exc<=tfin_wr_en and fin_full;
    fout_rd_exc<=tfout_rd_en and fout_empty;
--tar dcr
    m_reg<= tar_in when (msm_state = write_tar) else dcr_in(31 downto 1)&'0';
    m_reg_wr<='1' when (msm_state =write_tar or msm_state=write_dcr) else '0';
    m_reg_sel<="00" when msm_state=write_tar else"01";
    m_req<='1' when msm_state=trx_request else '0';
    m_stop<='1' when (msm_state=trx_stop) else '0';
--- to register
    trx_abort <= '1' when (msm_state = trx_aborted) else '0';
    ack <= '1' when (msm_state = write_tar) else '0';
    end_trx <= '1' when (msm_state = trx_stop or msm_state = trx_aborted or msm_state = ending_trx) else '0';
master_state_machine:   process(clk,reset)
    begin
        if reset='0' then
            msm_state<=idle;
```

```
        elsif clk'event and clk='1' then
            case msm_state is
                when idle=>
                if master='1'then
                    if m_channel='1'then
                        msm_state<=slave_trxing;
                    else
                        msm_state<=idle;
                    end if;
                elsif master='0'then
                    if sel_tar='1'and s_write='1' then
                        msm_state<=write_tar;
                    else
                        msm_state<=idle;
                    end if;
                end if;
                when slave_trxing=>
                    if m_channel='0'then
                        msm_state<=idle;
                    end if;
                when write_tar=>
                    if sel_dcr='1'and s_write='1'then
                    msm_state<=write_dcr;
                    else
                    msm_state<=write_tar;
                    end if;
                when write_dcr=>
                if start_trx='1'then
                    msm_state<=trx_request;
                else
                    msm_state<=wait_for_enable;
                end if;
                when wait_for_enable=>
                if start_trx='1' then
                    msm_state<=trx_request;
                else
                    msm_state<=wait_for_enable;
                end if;
                when trx_request=>
                msm_state<=wait_for_start_trx;
                when wait_for_start_trx=>
                if m_channel='1'then
                    msm_state<=trxing;
                else
                    msm_state<=wait_for_start_trx;
                end if;
                when trxing=>
                if m_abort='1'then
                    msm_state<=trx_aborted;
                elsif stop_trx='1'then
                    msm_state<=trx_stop;
                elsif m_channel='0'then
                    msm_state<=ending_trx;
                else
                    msm_state<=trxing;
                end if;
                when ending_trx=>
                msm_state<=idle;
                when trx_aborted=>
                msm_state<=idle;
                when trx_stop=>
                msm_state<=idle;
                end case;
                end if;
                end process master_state_machine;
        end behavioral;
```

Dma configure.

```
entity dma_config is
    port (
    clk:                in std_logic;
    reset:              in std_logic;
    s_write:            in std_logic;
    c_data_out:         in std_logic_vector(31 downto 0);
    sel_tar:            in std_logic;
    sel_dcr:            in std_logic;
    sel_dmareg:         in std_logic;
    fifoout_ndw:        in std_logic_vector(9 downto 0);
    fifoin_ndw:         in std_logic_vector(7 downto 0);
    ack:                in std_logic;--serve per flow status e quindi per il reg
    end_trx:            in std_logic;--comunica con dma ctrl insieme a pcitrxenabled
```

```
    tar:                out std_logic_vector(31 downto 0);
    stop_trx:           out std_logic;
    master:             out std_logic;
    dcr:                out std_logic_vector(31 downto 0);
    dma_status:         out std_logic_vector(21 downto 0);---serve per il registro
    start_trx:          out std_logic);
end dma_config;
architecture behavioral of dma_config is signal flow_status: std_logic_vector(2 downto 0);--segnali interni che
signal ndw_status: std_logic_vector(17 downto 0);--vanno al registro del circ. signal tstop_trx: std_logic;
begin stop_trx<=tstop_trx; DMA_CONF_REG: process(clk,reset) begin if(reset='0') then
        tar<=(others=>'0');
        dcr<=(others=>'0');
        master<='0';
        tstop_trx<='0';
    elsif (clk'event and clk='1') then
        if s_write='1' then
            if sel_tar='1' then
                tar<=c_data_out;
            elsif sel_dcr='1' then
                dcr<=c_data_out;
            elsif sel_dmareg='1'then
                master<=c_data_out(0);
                tstop_trx<=c_data_out(2);
            elsif tstop_trx='1'then
                tstop_trx<='0';
            end if;
            end if;
            end if;
        end process dma_conf_reg;
PCI_ENABLE_REGISTER: process (clk,reset) begin
    if(reset='0') then
        start_trx<='0';
    elsif (clk'event and clk='1') then
        if end_trx='1' then--end _trx abbassa start trx
            start_trx<='0';
        elsif s_write='1' and sel_dmareg='1'then
            start_trx<=c_data_out(1);--segnale che attiva la state machine del dma
        end if;
        end if;
    end process pci_enable_register;
STATUS_REGISTER: process(clk,reset)  --serve per il registro di lettura begin
    if(reset='0') then
        flow_status(2 downto 0) <= "000";
        ndw_status(15 downto 0) <= (others=>'0');
    elsif(clk'event and clk='1') then
        if(s_write = '1' and sel_dcr = '1') then
            flow_status(0) <= '1';---flow status e' un registro interno della dma
        elsif (ack = '1') then---che attraverso tre bit indica se si stanno scambiando
            flow_status(0) <= '0';---dati oppure no
        end if;                     ---- 1 bit:va a 1 se scrivo dcr e poi torna a zero a passaggio iniziato
                                    ---- 2 bit:va a 1 durante il passaggio finito questo torna a zero
        if(ack = '1') then          ---- 3 bit:va a 1 finito il passaggio torna a zero quando si scrive nuovamente il
            flow_status(1) <= '1';
        elsif(end_trx = '1') then
            flow_status(1) <= '0';
        end if;
        if(end_trx = '1') then
            flow_status(2) <= '1';
        elsif(s_write = '1' and sel_dcr = '1') then
            flow_status(2) <= '0';
        end if;
        ndw_status(7 downto 0) <= fifoin_ndw;---numero di parole contenute nella fifo
        ---uguale sotto
        ndw_status(17 downto 8) <= fifoout_ndw;
    end if;
end process STATUS_REGISTER;
DMA_STATUS <=  NDW_STATUS & '0' & FLOW_STATUS;
end behavioral;
```

Dma.

```
entity dma is
    port (
    clk:                in std_logic;
    reset:              in std_logic;
    m_read:             in std_logic;
    m_write:            in std_logic;
    s_write:            in std_logic;
```

```
    c_data_out:         in std_logic_vector(31 downto 0);
    sel_tar:            in std_logic;
    sel_dcr:            in std_logic;
    sel_dmareg:         in std_logic;
    fin_full:           in std_logic;
    fout_empty:         in std_logic;
    fifoout_ndw:        in std_logic_vector(9 downto 0);
    fifoin_ndw:         in std_logic_vector(7 downto 0);
    m_abort:            in std_logic;
    m_channel:          in std_logic;
    master:             out std_logic;
    fin_wr_en:          out std_logic;
    fout_rd_en:         out std_logic;
    fin_wr_exc:         out std_logic;
    fout_rd_exc:        out std_logic;
    tar_reg:            out std_logic_vector(31 downto 0);
    dcr_reg:            out std_logic_vector(31 downto 0);
    m_req:              out std_logic;
    m_reg:              out std_logic_vector(31 downto 0);
    m_reg_sel:          out std_logic_vector(1 downto 0);
    m_reg_wr:           out std_logic;
    master_ctrl:        out std_logic_vector(9 downto 0);--QUESTI ULTIMI SERVONO
    dma_status:         out std_logic_vector(21 downto 0);--PER IL REGISTRO
    pci_trx_enable:     out std_logic;
    end_trx:            out std_logic;
    trx_abort:          out std_logic;
    trx_stop:           out std_logic;
    m_stop:             out std_logic);
end dma;
architecture structural of dma is --component declarations--
component dma_ctrl port (
    clk:            in std_logic;
    reset:          in std_logic;
    m_write:        in std_logic;
    m_read:         in std_logic;
    m_abort:        in std_logic;
    m_channel:      in std_logic;
    s_write:        in std_logic;
    sel_tar:        in std_logic;
    sel_dcr:        in std_logic;
    start_trx:      in std_logic;
    tar_in:         in std_logic_vector(31 downto 0);
    dcr_in:         in std_logic_vector(31 downto 0);
    fin_full:       in std_logic;
    fout_empty:     in std_logic;
    master:         in std_logic;
    stop_trx:       in std_logic;
    fin_wr_en:      out std_logic;
    fout_rd_en:     out std_logic;
    fin_wr_exc:     out std_logic;
    fout_rd_exc:    out std_logic;
    ack:            out std_logic;
    m_req:          out std_logic;
    m_reg:          out std_logic_vector(31 downto 0);
    m_reg_sel:      out std_logic_vector(1 downto 0);
    m_reg_wr:       out std_logic;
    end_trx:        out std_logic;
    trx_abort:      out std_logic;--per interrupt handler
    m_stop:         out std_logic);
end component;
component dma_config
    port (
    clk:                in std_logic;
    reset:              in std_logic;
    s_write:            in std_logic;
    c_data_out:         in std_logic_vector(31 downto 0);
    sel_tar:            in std_logic;
    sel_dcr:            in std_logic;
    sel_dmareg:         in std_logic;
    fifoout_ndw:        in std_logic_vector(9 downto 0);
    fifoin_ndw:         in std_logic_vector(7 downto 0);
    ack:                in std_logic;--serve per flow status e quindi per il reg
    end_trx:            in std_logic;--comunica con dma ctrl insieme a pcitrxenabled
    tar:                out std_logic_vector(31 downto 0);
    stop_trx:           out std_logic;
```

```
    master:             out std_logic;
    dcr:                out std_logic_vector(31 downto 0);
    dma_status:         out std_logic_vector(21 downto 0);---serve per il registro
    start_trx:          out std_logic);
end component; --signal declarations-- signal tACK,tSTART_TRX: std_logic; signal tEND_TRX,tTRX_ABORT: std_logic;
signal tTAR,tDCR: std_logic_vector(31 downto 0); signal tm_stop: std_logic; signal tMASTER,STOPTRX: std_logic;
begin master<=tMASTER; dma_config_block : dma_config port map(
    clk             =>  clk,
    reset           =>  reset,
    s_write         =>  s_write,
    c_data_out      =>  c_data_out,
    sel_tar         =>  sel_tar,
    sel_dcr         =>  sel_dcr,
    sel_dmareg      =>  sel_dmareg,
    ack             =>  tACK,
    fifoin_ndw      =>  fifoin_ndw,
    fifoout_ndw     =>  fifoout_ndw,
    end_trx         =>  tEND_TRX,
    tar             =>  tTAR,
    dcr             =>  tDCR,
    master          =>  tMASTER,
    stop_trx        =>  STOPTRX,
    dma_status      =>  dma_status,
    start_trx       =>  tSTART_TRX);
dma_ctrl_block  : dma_ctrl port map(
    clk             => clk,
    reset           => reset,
    m_read          => m_read,
    m_write         => m_write,
    m_abort         => m_abort,
    m_channel       => m_channel,
    master          => tMASTER,
    stop_trx        => STOPTRX,
    s_write         => s_write,
    sel_tar         => sel_tar,
    sel_dcr         => sel_dcr,
    start_trx       => tSTART_TRX,
    tar_in          => tTAR,
    dcr_in          => tDCR,
    fin_full        => fin_full,
    fout_empty      => fout_empty,
    ack             => tACK,
    end_trx         => tEND_TRX,
    fin_wr_en       => fin_wr_en,
    fout_rd_en      => fout_rd_en,
    fin_wr_exc      => fin_wr_exc,
    fout_rd_exc     => fout_rd_exc,
    trx_abort       => tTRX_ABORT,
    m_req           => m_req,
    m_reg           => m_reg,
    m_reg_sel       => m_reg_sel,
    m_reg_wr        => m_reg_wr,
    m_stop          => tm_stop);
end_trx<=tEND_TRX; tar_reg<= tTAR; dcr_reg<=tDCR; trx_abort<=tTRX_ABORT; trx_stop<=tm_stop; m_stop<=tm_stop;
pci_trx_enable<=tSTART_TRX;
end structural;
```

Fetcher.

```
 entity fetcher is
    port (
    clk:            in std_logic;
    reset:          in std_logic;
    mode:           in std_logic;
    time_abs:       in std_logic_vector(31 downto 0);
    ctrl_time:      in std_logic;
    valid:          out std_logic
    );
end fetcher;
architecture behavioral of fetcher is signal TR: std_logic_vector(31 downto 0);    -- 31 bit begin
counter_block: process(clk,reset)
    begin
    if reset='0'then
        tr<=(others=>'0');
    elsif clk='1'and clk'event then
        if mode='1' then
            tr<=tr+1;
        elsif tr="11111111111111111111111111111111" then
            tr<=(others=>'0');
        end if;
    end if;
```

```
    end process;
TEMPO : process(clk,reset) begin
    if reset='0'then
        valid<='0';
    elsif clk='1'and clk'event then
        if ctrl_time='1' and time_abs<=tr then
            valid<='1';
            else
            valid<='0';
        end if;
    end if;
    end process;
end behavioral;
```

Memory configure.

```
entity mem_config is
    port (
    clk:            in std_logic;
    reset:          in std_logic;
    s_write:        in std_logic;
    sel_memreg1:    in std_logic;
    sel_memreg2:    in std_logic;
    c_data_out:     in std_logic_vector(31 downto 0);
    trxing:         in std_logic;
    rnw:            out std_logic;
    start_addr:     out std_logic_vector(19 downto 0);
    burst:          out std_logic_vector(7 downto 0);
    mem_ctrl_status:out std_logic_vector(1 downto 0);
    stop_run:       out std_logic;
    start_trx:      out std_logic);
end mem_config;
architecture behavioral of  mem_config is --internal register signal tSTARTADD: std_logic_vector(19 downt
signal tBURST: std_logic_vector(7 downto 0); signal tMEMCTRLSTATUS: std_logic_vector(1 downto 0); signal
tSTARTTRX,tRNW: std_logic; begin
    mem_config: process(clk,reset)
    begin
    if reset='0'then
        tSTARTADD<=(others=>'0');
        tBURST<=(others=>'0');
        stop_run<='0';
        trnw<='0';
    elsif  clk='1' and clk'event then
            if sel_memreg1='1'and s_write='1' then
            tSTARTADD<=c_data_out(23 downto 4);
            trnw<=c_data_out(2);
            stop_run<=c_data_out(3);
            elsif sel_memreg2='1'and s_write='1' then
                tBURST<=c_data_out(7 downto 0);
            end if;
            end if;
        end process;
        vaicoltrx: process(clk,reset)
        begin
        if reset='0' then
            tSTARTTRX<='0';
        elsif clk'event and clk='1' then
            if trxing='1'then
                tSTARTTRX<='0';
            elsif trxing='0'then
                if s_write='1'then
                    if sel_memreg1='1'then
                        tSTARTTRX<=c_data_out(1);
                    end if;
                end if;
            end if;
            end if;
        end process;
        status_reg: process(clk,reset)
        begin
            if  reset='0'then
                tMEMCTRLSTATUS<="00";
            elsif clk'event and clk='1'then
                if tSTARTTRX='1'then
                    tMEMCTRLSTATUS(0)<='1';
                else
                    tMEMCTRLSTATUS(0)<='0';
                end if;
                if trxing='0' then
                    tMEMCTRLSTATUS(1)<='0';
```

```
            else
                tMEMCTRLSTATUS(1)<='1';
            end if;
        end if;
        end process;
        --segnali in uscita
        mem_ctrl_status<=tMEMCTRLSTATUS;
        burst<=tBURST;
        start_addr<=tSTARTADD;
        start_trx<=tSTARTTRX;
        rnw<=trnw;
            end behavioral;
```

Memory control.

```
 entity mem_ctrl is
    port (
    clk:            in std_logic;
    reset:          in std_logic;
    burst:          in std_logic_vector(7 downto 0);
    start_addr:     in std_logic_vector(19 downto 0);
    start_trx:      in std_logic;
    fin_empty:      in std_logic;
    fout_full:      in std_logic;
    mode:           in std_logic;
    rnw:            in std_logic;
    stop_run:       in std_logic;
    valid:          in std_logic;--gli ultimi cinque segnali dovrebbero provenire dal fetcher
    mem_out:        in std_logic_vector(31 downto 0);--DATA TO MEM
    data_fromfifo:  in std_logic_vector(31 downto 0);
    write_end:      out std_logic;
    data_tofifo:    out std_logic_vector(31 downto 0);
    time_abs:       out std_logic_vector(31 downto 0);
    ctrl_time:      out std_logic;
    slow_reg:       out std_logic;
    data_reg:       out std_logic;
    fifoin_rden:    out std_logic;
    fifoout_wren:   out std_logic;
    fin_rd_exc:     out std_logic;
    fout_wr_exc:    out std_logic;
    trxing:         out std_logic;
    stop_emu:       out std_logic;
    burst_lenght:   out std_logic_vector(5 downto 0);
    mem_in:         out std_logic_vector(31 downto 0);--DATA FROM MEM
    --memory signals
        Addr      : OUT    STD_LOGIC_VECTOR (19 DOWNTO 0);
        Adsc_n    : OUT    STD_LOGIC;
        Gw_n      : OUT    STD_LOGIC;
        Ce_n      : OUT    STD_LOGIC;
        Oe_n      : OUT    STD_LOGIC);
end mem_ctrl;
architecture behavioral of mem_ctrl is type macchina is
(idle,init_trx,read_header,data_header,slow_header,abs_time,read,
                  write,read_deselect,wait_incadd);
--internal signals signal mem_state:  macchina; signal slow,data,header_stop: std_logic; signal burst_len:
std_logic_vector(5 downto 0); signal add: integer range 0 to 1154967295; signal word: integer range 0 to 255;
signal last_data: std_logic; signal incadd: std_logic; signal stop,reg:std_logic; signal
tfifoout_wren,tfifoin_rden: std_logic;
begin fifoout_wren<=tfifoout_wren; fifoin_rden<=tfifoin_rden; data_tofifo<=mem_out; adsc_n <= '0'; --registro
data_reg<='1' when mem_state=data_header else '0'; slow_reg<='1' when mem_state=slow_header else '0';
---fifo signals
mem_in<= data_fromfifo when mem_state=write else (others=>'0'); data_tofifo<=mem_out; tfifoin_rden<='1'when
mem_state=write else '0'; tfifoout_wren<='1'when mem_state=read or mem_state=slow_header or(mem_state=abs_time
and valid='1')or mem_state=data_header else '0';
---exceptions
write_end<= '1'when mem_state=write and last_data='1' else '0'; fin_rd_exc<=tfifoin_rden and fin_empty;
fout_wr_exc<=tfifoout_wren and fout_full;
---fetcher signals
trxing<='1'when mem_state=init_trx or mem_state=read_header else '0';
---controlli di membank
Gw_n<='0'when mem_state=write else '1'; Ce_n<='1'when mem_state=idle or mem_state=read_deselect else '0';
Oe_n<='1'when mem_state=idle or mem_state=write else '0';
-----internal process
last_data<='1' when word=1 else '0'; ADDR<=conv_std_logic_vector(add,20); incadd<='1' when mem_state=write or
(mem_state=read_header and slow='1') or (mem_state=abs_time and valid='1')or(mem_state=read_header and data='1')
or mem_state=data_header or (mem_state=read and last_data='0') or (mem_state=init_trx and rnw='1') else '0';
----head decoder
slow<= '1' when (mem_out(31)='0'and mem_out(30)='1'and mem_state=read_header) else '0'; data<= '1' when
```

```
(mem_out(31)='1'and mem_out(30)='0'and mem_state=read_header)else '0'; header_stop<='1'when mem_out(31)='
mem_out(30)='1'else '0';
----abs time
ctrl_time<='1'when mem_state=abs_time else '0'; time_abs<=mem_out when mem_state=abs_time;
burst_len<=(others=>'0') when (mem_state=idle or mode='0')else mem_out(21 downto 16); burst_lenght<=burst
    machine: process(clk,reset)
    begin
        if reset='0' then
            mem_state<=idle;
        elsif clk='1'and clk'event then
            case mem_state is
                when idle=>
                    if mode='0' and start_trx='1'then
                    mem_state<=init_trx;
                    elsif mode='1' then
                    mem_state<=read_header;
                else
                    mem_state<=idle;
                end if;
                when init_trx=>
                if rnw='0'then
                    mem_state<=write;
                elsif rnw='1'then
                    mem_state<=read;
                end if;
                when read_header=>
                if header_stop='1' then
                    mem_state<=read_deselect;
                elsif slow='1'and mode='1' then
                    mem_state<=slow_header;
                elsif slow='1' and mode='1'then
                    mem_state<=read_deselect;
                elsif data='1'then
                    mem_state<=data_header;
                else mem_state<=read_header;
                end if;
                when read_deselect=>
                mem_state<=idle;
                when slow_header=>
                mem_state<=abs_time;
                when abs_time=>
                if valid='1'then
                mem_state<=wait_incadd;
                else
                    mem_state<=abs_time;
                end if;
                when wait_incadd=>
                mem_state<=read_header;
                when data_header=>
                    mem_state<=read;
                when write=>
                if last_data='1'then
                    if mode='0'then
                        mem_state<=idle;
                    end if;
                    end if;
                when read=>
                if last_data='1'then
                if mode='0'then
                        mem_state<=idle;
                    elsif mode='1'then
                        mem_state<=read_header;
                    end if;
                end if;
                end case;
                end if;
                end process machine;
ADDRCTRL: process(clk,reset)
    begin
    if reset='0'then
            add<=0;
    elsif clk='1'and clk'event then
            if mem_state=idle and (start_trx='1'or mode='1') then
                add<=conv_integer(start_addr);
            elsif incadd='1'then
                add<=add+1;
            end if;
    end if;
    end process;
WORDCONTROL: process(clk,reset) begin
    if reset='0'then
        word<=1;
    elsif clk='1'and clk'event then
        if mode='0'and mem_state=init_trx then
            word<=conv_integer(burst);
```

```
        elsif mode='1'and mem_state=read_header then
            word<=conv_integer(burst_len);
        end if;
        if mem_state=read or mem_state=data_header or mem_state=write then
            word<=word-1;
        end if;
        end if;
    end process;
STOPPALO: process(clk,reset) begin
    if reset='0'then
    stop<='0';
    elsif clk='1'and clk'event then
    if header_stop='1' and mem_state=read_header then
    stop<='1';
    elsif stop_run='1'and mode='1' then
    stop<='1';
    else
    stop<='0';
    end if;
    end if;
    end process;
    stop_emu<=stop;
    end behavioral;
```

Memory controller.

```
entity mem is
    port(
    clk:            in std_logic;
    reset:          in std_logic;
    mode:           in std_logic;
    mem_out:        in std_logic_vector(31 downto 0);
    data_fromfifo:  in std_logic_vector(31 downto 0);
    s_write:        in std_logic;
    sel_memreg1:    in std_logic;
    sel_memreg2:    in std_logic;
    fin_empty:      in std_logic;
    fout_full:      in std_logic;
    c_data_out:     in std_logic_vector(31 downto 0);
    burst:          out std_logic_vector(7 downto 0);
    mem_ctrl_status:out std_logic_vector(1 downto 0);
    write_end:      out std_logic;
    trxing:         out std_logic;
    rnw:            out std_logic;
    slow_reg:       out std_logic;
    data_reg:       out std_logic;
    fifoin_rden:    out std_logic;
    fifoout_wren:   out std_logic;
    fin_rd_exc:     out std_logic;
    fout_wr_exc:    out std_logic;
    stop_emu:       out std_logic;
    start_trx:      out std_logic;
    burst_lenght:   out std_logic_vector(5 downto 0);
    mem_in:         out std_logic_vector(31 downto 0);
    start_add:      out std_logic_vector(19 downto 0);
    data_tofifo:    out std_logic_vector(31 downto 0);
        ----memory control
    ram_ad:         out std_logic_vector(19 downto 0);
    ram_CE0:        out std_logic;
    ram_adsc:       out std_logic;
    ram_we:         out std_logic;
    ram_oe:         out std_logic
    );
end mem;
architecture structural of mem is component mem_ctrl
    port (
    clk:            in std_logic;
    reset:          in std_logic;
    burst:          in std_logic_vector(7 downto 0);
    start_addr:     in std_logic_vector(19 downto 0);
    start_trx:      in std_logic;
    fin_empty:      in std_logic;
    fout_full:      in std_logic;
    mode:           in std_logic;
    rnw:            in std_logic;
    stop_run:       in std_logic;
    valid:          in std_logic;--gli ultimi cinque segnali dovrebbero provenire dal fetcher
    mem_out:        in std_logic_vector(31 downto 0);--DATA TO MEM
```

```
    data_fromfifo:  in std_logic_vector(31 downto 0);
    data_tofifo:    out std_logic_vector(31 downto 0);
    time_abs:       out std_logic_vector(31 downto 0);
    write_end:      out std_logic;
    ctrl_time:      out std_logic;
    slow_reg:       out std_logic;
    data_reg:       out std_logic;
    fifoin_rden:    out std_logic;
    fifoout_wren:   out std_logic;
    fin_rd_exc:     out std_logic;
    fout_wr_exc:    out std_logic;
    trxing:         out std_logic;
    stop_emu:       out std_logic;
    burst_lenght:   out std_logic_vector(5 downto 0);
    mem_in:         out std_logic_vector(31 downto 0);--DATA FROM MEM
    --memory signals
        Addr      : OUT    STD_LOGIC_VECTOR (19 DOWNTO 0);
        Adsc_n    : OUT    STD_LOGIC;
        Gw_n      : OUT    STD_LOGIC;
        Ce_n      : OUT    STD_LOGIC;
        Oe_n      : OUT    STD_LOGIC);
end component; component mem_config
    port (
    clk:            in std_logic;
    reset:          in std_logic;
    s_write:        in std_logic;
    sel_memreg1:    in std_logic;
    sel_memreg2:    in std_logic;
    c_data_out:     in std_logic_vector(31 downto 0);
    trxing:         in std_logic;
    rnw:            out std_logic;
    start_addr:     out std_logic_vector(19 downto 0);
    burst:          out std_logic_vector(7 downto 0);
    mem_ctrl_status:out std_logic_vector(1 downto 0);
    stop_run:       out std_logic;
    start_trx:      out std_logic);
end component ;
component fetcher
        port (
    clk:            in std_logic;
    reset:          in std_logic;
    mode:           in std_logic;
    time_abs:       in std_logic_vector(31 downto 0);
    ctrl_time:      in std_logic;
    valid:          out std_logic
    );
end component;
---signal declarations------
signal tSTART_TRX,tTRXING,HEAD,CTRLTIME ,VALID,CTRLFET,STOPRUN,trnw: std_logic; signal tSTARTADDR:
std_logic_vector(19 downto 0); signal tBURST:  std_logic_vector(7 downto 0); signal TIMEABS: std_logic_ve
downto 0); begin
MEMCONF : mem_config port map(
    clk             =>  clk,
    reset           =>  reset,
    s_write         =>  s_write,
    sel_memreg1     =>  sel_memreg1,
    sel_memreg2     =>  sel_memreg2,
    c_data_out      =>  c_data_out,
    trxing          =>  tTRXING,
    burst           =>  tBURST,
    start_addr      =>  tSTARTADDR,
    mem_ctrl_status =>  mem_ctrl_status,
    rnw             =>  trnw,
    stop_run        =>  STOPRUN,
    start_trx       =>  tSTART_TRX);
MEMCTRL : mem_ctrl port map(
    clk             =>  clk,
    reset           =>  reset,
    burst           =>  tBURST,
    start_addr      =>  tSTARTADDR,
    start_trx       =>  tSTART_TRX,
    mode            =>  mode,
    valid           =>  VALID,
    time_abs        =>  TIMEABS,
    ctrl_time       =>  CTRLTIME,
    fifoin_rden     =>  fifoin_rden,
    fifoout_wren    =>  fifoout_wren,
```

```
    fin_rd_exc      =>  fin_rd_exc,
    fout_wr_exc     =>  fout_wr_exc,
    fin_empty       =>  fin_empty,
    fout_full       =>  fout_full,
    slow_reg        =>  slow_reg,
    data_reg        =>  data_reg,
    trxing          =>  tTRXING,
    stop_run        =>  STOPRUN,
    stop_emu        =>  stop_emu,
    write_end       =>  write_end,
    mem_in          =>  mem_in,
    rnw             =>  trnw,
    mem_out         =>  mem_out,
    data_fromfifo   =>  data_fromfifo,
    data_tofifo     =>  data_tofifo,
    burst_lenght    =>  burst_lenght,
    --memory signals
       Addr => ram_ad,
       Adsc_n => ram_adsc,
       Gw_n => ram_we,
       Ce_n => ram_CE0,
       Oe_n => ram_oe
       );
FETBLOCK : fetcher port  map(
    clk              => clk,
    reset            => reset,
    mode             => mode,
    valid            => VALID,
    time_abs         => TIMEABS,
    ctrl_time        => CTRLTIME
    );
trxing<=tTRXING; burst<=tBURST; start_add<=tSTARTADDR; start_trx<=tSTART_TRX; rnw<=trnw;
    end structural;
```

Memoria.

```
--------------------------------------------------------------------------------
-- --    File Name: MT58L1MY18F.VHD --     Revision: 2.0 --         Date: April 3rd, 2002 --        Model: Bus
Functional --    Simulator: Aldec, ModemSim, NCDesktop -- -- Dependencies: None -- --       Author: Son P. Huynh
--        Email: sphuynh@micron.com --        Phone: (208) 368-3825 --      Company: Micron Technology, Inc. --
Part #: MT58L1MY18F (1M x 18) -- --  Description: Micron 18 Meg SyncBurst SRAM (Flow-through) -- --
Disclaimer: THESE DESIGNS ARE PROVIDED "AS IS" WITH NO WARRANTY --              WHATSOEVER AND MICRON
SPECIFICALLY DISCLAIMS ANY --              IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR --
A PARTICULAR PURPOSE, OR AGAINST INFRINGEMENT. -- --              Copyright (c) 1997 Micron Semiconductor
Products, Inc. --              All rights researved -- -- Rev  Author          Phone         Date
Changes -- ---  --------------  ------------  ----------  ------------------------------ -- 2.0  Son P. Huynh
208-368-3825  04/03/2002  - Fix Burst counter --      Micron Technology, Inc. --
--------------------------------------------------------------------------------
LIBRARY ieee;
    USE ieee.std_logic_1164.all;
    USE ieee.std_logic_unsigned.all;
ENTITY MT58L1MY18F IS
    GENERIC (
        -- Clock
        tKC       : TIME    :=  8.0 ns;  -- Timing are for -6.8
        tKH       : TIME    :=  1.8 ns;
        tKL       : TIME    :=  1.8 ns;
        -- Output Times
        tKQHZ     : TIME    :=  3.8 ns;
        -- Setup Times
        tAS       : TIME    :=  1.8 ns;
        tADSS     : TIME    :=  1.8 ns;
        tAAS      : TIME    :=  1.8 ns;
        tWS       : TIME    :=  1.8 ns;
        tDS       : TIME    :=  1.8 ns;
        tCES      : TIME    :=  1.8 ns;
        -- Hold Times
        tAH       : TIME    :=  0.5 ns;
        tADSH     : TIME    :=  0.5 ns;
        tAAH      : TIME    :=  0.5 ns;
        tWH       : TIME    :=  0.5 ns;
        tDH       : TIME    :=  0.5 ns;
        tCEH      : TIME    :=  0.5 ns;
        -- Bus Width and Data Bus
        addr_bits : INTEGER := 20;
        data_bits : INTEGER := 18
    );
    PORT (
        Dq        : INOUT STD_LOGIC_VECTOR (data_bits - 1 DOWNTO 0) := (OTHERS => 'Z');
        Addr      : IN    STD_LOGIC_VECTOR (addr_bits - 1 DOWNTO 0);
        Mode      : IN    STD_LOGIC; ---'0'
        Adv_n     : IN    STD_LOGIC; ---'1'
        Clk       : IN    STD_LOGIC;
```

```
        Adsc_n    : IN    STD_LOGIC;
        Adsp_n    : IN    STD_LOGIC;---'1'
        Bwa_n     : IN    STD_LOGIC;---'1'
        Bwb_n     : IN    STD_LOGIC;---'1'
        Bwe_n     : IN    STD_LOGIC;---'1'
        Gw_n      : IN    STD_LOGIC;
        Ce_n      : IN    STD_LOGIC;
        Ce2       : IN    STD_LOGIC;---'1'
        Ce2_n     : IN    STD_LOGIC;---'0'
        Oe_n      : IN    STD_LOGIC;
        Zz        : IN    STD_LOGIC);---'0'
END MT58L1MY18F;
ARCHITECTURE behave OF MT58L1MY18F IS
    TYPE   memory IS ARRAY (2 ** addr_bits - 1 DOWNTO 0) OF STD_LOGIC_VECTOR (data_bits / 2 - 1 DOWNTO 0)
    SIGNAL doe : STD_LOGIC;
    SIGNAL dout : STD_LOGIC_VECTOR (data_bits - 1 DOWNTO 0) := (OTHERS => 'Z');
    SIGNAL bwan, bwbn, ce, clr : STD_LOGIC;
BEGIN
    bwan <= ((Bwa_n OR Bwe_n) AND Gw_n) OR (NOT(Ce_n) AND NOT(Adsp_n));
    bwbn <= ((Bwb_n OR Bwe_n) AND Gw_n) OR (NOT(Ce_n) AND NOT(Adsp_n));
    ce   <= NOT(Ce_n) AND Ce2 AND NOT(Ce2_n);
    clr  <= NOT(Adsc_n) OR (NOT(Adsp_n) AND NOT(Ce_n));
    ---bwan=1 e bwbn=1 se Gw_n=1
    ---ce=1 se ce_n=0
    ---clr=1 se adsc_n=0
    main : PROCESS
        -- Memory Array
        VARIABLE bank0, bank1 : memory;
        -- Address Registers
        VARIABLE addr_reg_in  : STD_LOGIC_VECTOR (addr_bits - 1 DOWNTO 0) := (OTHERS => '0');
        VARIABLE addr_reg_out : STD_LOGIC_VECTOR (addr_bits - 1 DOWNTO 0) := (OTHERS => '0');
        -- Burst Counter
        VARIABLE bcount : STD_LOGIC_VECTOR (1 DOWNTO 0) := "00";
        VARIABLE baddr0 : STD_LOGIC;
        VARIABLE baddr1 : STD_LOGIC;
        -- Other Registers
        VARIABLE din     : STD_LOGIC_VECTOR (data_bits - 1 DOWNTO 0) := (OTHERS => 'Z');
        VARIABLE ce_reg  : STD_LOGIC;
        VARIABLE bwa_reg : STD_LOGIC;
        VARIABLE bwb_reg : STD_LOGIC;
    BEGIN
        WAIT ON Clk;
        IF Clk'EVENT AND Clk = '1' AND Zz = '0' THEN
            -- Address Register
            IF clr = '1' THEN ---se adsc_n='0'
                addr_reg_in := Addr;--- set dell'address reg. d'ingresso
            END IF;
            -- Binary Counter and Logic
            IF Mode = '1' AND clr = '1' THEN---mai vera
                bcount := "00";
            ELSIF Mode = '0' AND clr = '1' THEN---mode nel top e' sempre a zero
                bcount := Addr(1 DOWNTO 0);
            ELSIF Adv_n = '0' AND clr = '0' THEN---mai vera
                bcount(1) := bcount(0) XOR bcount(1);
                bcount(0) := NOT(bcount(0));
            END IF;
            -- Burst Address Decode
            IF Mode = '1' THEN ---mai vera
                baddr0 := bcount(0) XOR addr_reg_in(0);
                baddr1 := bcount(1) XOR addr_reg_in(1);
            ELSE
                baddr0 := bcount(0);---sempre vera
                baddr1 := bcount(1);
            END IF;
            -- Output Address
            addr_reg_out (addr_bits - 1 DOWNTO 2) := addr_reg_in (addr_bits - 1 DOWNTO 2);
            addr_reg_out (1) := baddr1;
            addr_reg_out (0) := baddr0;
            -- Byte Write Register
            bwa_reg := NOT(bwan);
            bwb_reg := NOT(bwbn);---uguali a 1 se Gw_n=0
            -- Enable Register
            IF clr = '1' THEN
                ce_reg := ce;---se ce_n=0
            END IF;
            -- Input Register
            IF (ce_reg = '1' AND (bwa_reg = '1' OR bwb_reg = '1')) THEN
                din := Dq;---se Gw_n=0 e ce_n=0
            END IF;
```

```
        -- Byte Write Driver
        IF ce_reg = '1' AND bwa_reg = '1' THEN---se Gw_n=0 e ce_n=0
            bank0 (CONV_INTEGER(addr_reg_out)) := din ( 8 DOWNTO 0);
        END IF;
        IF ce_reg = '1' AND bwb_reg = '1' THEN---se Gw_n=0 e ce_n=0
            bank1 (CONV_INTEGER(addr_reg_out)) := din (17 DOWNTO 9);
        END IF;
        -- Output Register
        IF (NOT(bwa_reg = '1' OR bwb_reg = '1')) THEN---se Gw_n=1
            dout ( 8 DOWNTO 0) <= bank0 (CONV_INTEGER(addr_reg_out));
            dout (17 DOWNTO 9) <= bank1 (CONV_INTEGER(addr_reg_out));
        END IF;
        -- Data Out Enable
        doe <= ce_reg AND (NOT(bwa_reg OR bwb_reg));
    END IF;---doe=1 se Gw_n=1 e ce_n=0
END PROCESS main;
-- Output buffer
WITH (NOT(Oe_n) AND NOT(Zz) AND doe) SELECT
    Dq <= TRANSPORT dout AFTER tKQHZ WHEN '1',---oe_n=0 Gw_n=1 ce_n=0
          (OTHERS => 'Z') AFTER tKQHZ WHEN '0',
          (OTHERS => 'Z') AFTER tKQHZ WHEN OTHERS;
-- Checking for setup time violation
Setup_check : PROCESS
BEGIN
    WAIT ON Clk;
    IF Clk'EVENT AND Clk = '1' THEN
        ASSERT(Addr'LAST_EVENT >= tAS)
            REPORT "Addr Setup time violation -- tAS"
            SEVERITY WARNING;
        ASSERT(Adsc_n'LAST_EVENT >= tADSS)
            REPORT "Adsc_n Setup time violation -- tADSS"
            SEVERITY WARNING;
        ASSERT(Adsp_n'LAST_EVENT >= tADSS)
            REPORT "Adsp_n Setup time violation -- tADSS"
            SEVERITY WARNING;
        ASSERT(Adv_n'LAST_EVENT >= tAAS)
            REPORT "Adv_n Setup time violation -- tAAS"
            SEVERITY WARNING;
        ASSERT(Bwa_n'LAST_EVENT >= tWS)
            REPORT "Bwa_n Setup time violation -- tWS"
            SEVERITY WARNING;
        ASSERT(Bwb_n'LAST_EVENT >= tWS)
            REPORT "Bwb_n Setup time violation -- tWS"
            SEVERITY WARNING;
        ASSERT(Bwe_n'LAST_EVENT >= tWS)
            REPORT "Bwe_n Setup time violation -- tWS"
            SEVERITY WARNING;
        ASSERT(Gw_n'LAST_EVENT >= tWS)
            REPORT "Gw_n Setup time violation -- tWS"
            SEVERITY WARNING;
        ASSERT(Ce_n'LAST_EVENT >= tCES)
            REPORT "Ce_n Setup time violation -- tCES"
            SEVERITY WARNING;
        ASSERT(Ce2_n'LAST_EVENT >= tCES)
            REPORT "Ce2_n Setup time violation -- tCES"
            SEVERITY WARNING;
        ASSERT(Ce2'LAST_EVENT >= tCES)
            REPORT "Ce2 Setup time violation -- tCES"
            SEVERITY WARNING;
    END IF;
END PROCESS;
-- Checking for hold time violation
Hold_check : PROCESS
BEGIN
    WAIT ON Clk'DELAYED(tAH), Clk'DELAYED(tADSH), Clk'DELAYED(tAAH), Clk'DELAYED(tWH), Clk'DELAYED(tCEH);
    IF Clk'DELAYED(tAH)'EVENT AND Clk'DELAYED(tAH) = '1' THEN
        ASSERT(Addr'LAST_EVENT > tAH)
            REPORT "Addr Hold time violation -- tAH"
            SEVERITY WARNING;
    END IF;
    IF Clk'DELAYED(tADSH)'EVENT AND Clk'DELAYED(tADSH) = '1' THEN
        ASSERT(Adsc_n'LAST_EVENT > tADSH)
            REPORT "Adsc_n Hold time violation -- tADSH"
            SEVERITY WARNING;
        ASSERT(Adsp_n'LAST_EVENT > tADSH)
            REPORT "Adsp_n Hold time violation -- tADSH"
            SEVERITY WARNING;
    END IF;
    IF Clk'DELAYED(tAAH)'EVENT AND Clk'DELAYED(tAAH) = '1' THEN
        ASSERT(Adv_n'LAST_EVENT > tAAH)
```

```
                REPORT "Adv_n Hold time violation -- tAAH"
                SEVERITY WARNING;
        END IF;
        IF Clk'DELAYED(tWH)'EVENT AND Clk'DELAYED(tWH) = '1' THEN
            ASSERT(Bwa_n'LAST_EVENT > tWH)
                REPORT "Bwa_n Hold time violation -- tWH"
                SEVERITY WARNING;
            ASSERT(Bwb_n'LAST_EVENT > tWH)
                REPORT "Bwb_n Hold time violation -- tWH"
                SEVERITY WARNING;
            ASSERT(Bwe_n'LAST_EVENT > tWH)
                REPORT "Bwe_n Hold time violation -- tWH"
                SEVERITY WARNING;
            ASSERT(Gw_n'LAST_EVENT > tWH)
                REPORT "Gw_n Hold time violation -- tWH"
                SEVERITY WARNING;
        END IF;
        IF Clk'DELAYED(tCEH)'EVENT AND Clk'DELAYED(tCEH) = '1' THEN
            ASSERT(Ce_n'LAST_EVENT > tCEH)
                REPORT "Ce_n Hold time violation -- tCEH"
                SEVERITY WARNING;
            ASSERT(Ce2_n'LAST_EVENT > tCEH)
                REPORT "Ce2_n Hold time violation -- tCEH"
                SEVERITY WARNING;
            ASSERT(Ce2'LAST_EVENT > tCEH)
                REPORT "Ce2 Hold time violation -- tCEH"
                SEVERITY WARNING;
        END IF;
    END PROCESS;
END behave;
```

Backend.

```
 entity backend is
  port(
       Clk : in STD_LOGIC;
       m_read : in STD_LOGIC_vector(3 downto 0);
       m_write : in STD_LOGIC_vector(3 downto 0);
       reset : in STD_LOGIC;
       c_sm: In std_logic_vector(11 downto 0);
       C_data_out : in STD_LOGIC_VECTOR(31 downto 0);
       s_addr : in STD_LOGIC_VECTOR(4 downto 0);
       s_read : in STD_LOGIC_VECTOR(6 downto 0);
       s_write : in STD_LOGIC_VECTOR(5 downto 0);
       m_abort: in std_logic;
       m_channel: in std_logic;
       m_req: Out std_logic;
       m_reg: Out std_logic_vector(31 downto 0);
       m_reg_sel: Out std_logic_vector(1 downto 0);
       m_reg_wr: Out std_logic;
       m_stop: Out std_logic;
       int_request: Out std_logic;
       int_reset: Out std_logic;
       ram_ADSC : out STD_LOGIC;
       ram_CE0 : out STD_LOGIC;
       ram_OE : out STD_LOGIC;
       ram_WE : out STD_LOGIC;
       ram_ad : out STD_LOGIC_VECTOR(19 downto 0);
       s_data_in : out STD_LOGIC_VECTOR(31 downto 0);
       m_data_in : out STD_LOGIC_VECTOR(31 downto 0);
       ram_data : inout STD_LOGIC_VECTOR(31 downto 0)
  );
end backend;
architecture behavioral of backend is
---- Component declarations -----
    component fifo_1024x32
  port (
       clock : in STD_LOGIC;
       data : in STD_LOGIC_VECTOR(31 downto 0);
       rdreq : in STD_LOGIC;
       wrreq : in STD_LOGIC;
       aclr: in std_logic;
       empty : out STD_LOGIC;
       full : out STD_LOGIC;
       q : out STD_LOGIC_VECTOR(31 downto 0);
       usedw : out STD_LOGIC_VECTOR(9 downto 0)
  );
end component; component fifo_256x32
  port (
       clock : in STD_LOGIC;
       data : in STD_LOGIC_VECTOR(31 downto 0);
```

```
       rdreq : in STD_LOGIC;
       wrreq : in STD_LOGIC;
       aclr: in std_logic;
       empty : out STD_LOGIC;
       full : out STD_LOGIC;
       q : out STD_LOGIC_VECTOR(31 downto 0);
       usedw : out STD_LOGIC_VECTOR(7 downto 0)
  );
end component; component dma
    port (
    clk:                in std_logic;
    reset:              in std_logic;
    m_read:             in std_logic;
    m_write:            in std_logic;
    s_write:            in std_logic;
    c_data_out:         in std_logic_vector(31 downto 0);
    sel_tar:            in std_logic;
    sel_dcr:            in std_logic;
    sel_dmareg:         in std_logic;
    fin_full:           in std_logic;
    fout_empty:         in std_logic;
    fifoout_ndw:        in std_logic_vector(9 downto 0);
    fifoin_ndw:         in std_logic_vector(7 downto 0);
    m_abort:            in std_logic;
    m_channel:          in std_logic;
    master:             out std_logic;
    fin_wr_en:          out std_logic;
    fout_rd_en:         out std_logic;
    fin_wr_exc:         out std_logic;
    fout_rd_exc:        out std_logic;
    tar_reg:            out std_logic_vector(31 downto 0);
    dcr_reg:            out std_logic_vector(31 downto 0);
    m_req:              out std_logic;
    m_reg:              out std_logic_vector(31 downto 0);
    m_reg_sel:          out std_logic_vector(1 downto 0);
    m_reg_wr:           out std_logic;
    master_ctrl:        out std_logic_vector(9 downto 0);--QUESTI ULTIMI SERVONO
    dma_status:         out std_logic_vector(21 downto 0);--PER IL REGISTRO
    pci_trx_enable:     out std_logic;
    end_trx:            out std_logic;
    trx_abort:          out std_logic;
    trx_stop:           out std_logic;
    m_stop:             out std_logic);
end component;
component mem
    port(
    clk:            in std_logic;
    reset:          in std_logic;
    mode:           in std_logic;
    mem_out:        in std_logic_vector(31 downto 0);
    data_fromfifo:  in std_logic_vector(31 downto 0);
    s_write:        in std_logic;
    sel_memreg1:    in std_logic;
    sel_memreg2:    in std_logic;
    fin_empty:      in std_logic;
    fout_full:      in std_logic;
    c_data_out:     in std_logic_vector(31 downto 0);
    burst:          out std_logic_vector(7 downto 0);
    mem_ctrl_status:out std_logic_vector(1 downto 0);
    write_end:      out std_logic;
    trxing:         out std_logic;
    rnw:            out std_logic;
    slow_reg:       out std_logic;
    data_reg:       out std_logic;
    fifoin_rden:    out std_logic;
    fifoout_wren:   out std_logic;
    fin_rd_exc:     out std_logic;
    fout_wr_exc:    out std_logic;
    stop_emu:       out std_logic;
    start_trx:      out std_logic;
    burst_lenght:   out std_logic_vector(5 downto 0);
    mem_in:         out std_logic_vector(31 downto 0);
    start_add:      out std_logic_vector(19 downto 0);
    data_tofifo:    out std_logic_vector(31 downto 0);
        ----memory control
```

```
    ram_ad:         out std_logic_vector(19 downto 0);
    ram_CE0:        out std_logic;
    ram_adsc:       out std_logic;
    ram_we:         out std_logic;
    ram_oe:         out std_logic
    );
end component;
component addr_select
    port(
    s_addr      :               in std_logic_vector(4 downto 0);
    sel_DMAREG:                 out std_logic;
    sel_TAR:                    out std_logic;
    sel_DCR:                    out std_logic;
    sel_MEMREG1:                out std_logic;
    sel_MEMREG2:                out std_logic;
    sel_NSFP:                   out std_logic;
    sel_INTREG:                 out std_logic
        );
end component ;
component registro
    port (
    clk:                        in std_logic;
    reset:                      in std_logic;
    s_read:                     in std_logic;
    s_write:                    in std_logic;
    c_data_out:                 in std_logic_vector(31 downto 0);
    sel_TAR:                    in std_logic;
    sel_DCR:                    in std_logic;
    sel_DMAREG:                 in std_logic;
    sel_NSFP:                   in std_logic;
    sel_INTREG:                 in std_logic;
    sel_MEMREG1:                in std_logic;
    sel_MEMREG2:                in std_logic;
    --dma
    tar:                        in std_logic_vector(31 downto 0);
    dcr:                        in std_logic_vector(31 downto 0);
    dma_status:                 in std_logic_vector(21 downto 0);
    pci_trx_enable:             in std_logic;
    end_trx:                    in std_logic;
    trx_abort:                  in std_logic;
    trx_stop:                   in std_logic;
    master:                     in std_logic;
    -- memory controller
    start_add:                  in std_logic_vector(19 downto 0);
    burst:                      in std_logic_vector(7 downto 0);
    burst_len:                  in std_logic_vector(5 DOWNTO 0);
    start_trx:                  in std_logic;
    mode:                       in std_logic;
    rnw:                        in std_logic;
    mem_ctrl_status:            in std_logic_vector(1 downto 0);
    --fifo
    fifoout_rd_exc:             in std_logic;
    fifoin_rd_exc:              in std_logic;
    fifoout_wr_exc:             in std_logic;
    fifoin_wr_exc:              in std_logic;
    ----NumSlowFramePresent
    emu_stop:                   in std_logic;
    slow:                       in std_logic;
    data:                       in std_logic;
    ---interrupt
    vector_mask:                in std_logic_vector(8 downto 0);
    interrupt_vector:           in std_logic_vector(8 downto 0);
    slow_int:                   out std_logic;
    s_data_in:                  out std_logic_vector(31 downto 0)
    );
    end component ;
component switch port (
    clk:        in std_logic;
    reset:      in std_logic;
    sel_memreg1:in std_logic;
    s_write:    in std_logic;
    stop:       in std_logic;
    c_data_out: in std_logic_vector(31 downto 0);
    emu_stop:   out std_logic;
    mode:       out std_logic
    );
```

```
    end component ;
component tristate
    port(
        DIn: In std_logic_vector(31 downto 0);
        DOut: Out std_logic_vector(31 downto 0);
        Dir: In std_logic;
        Dq: InOut std_logic_vector(31 downto 0)
    );
end component; component resetta port (
    clk:        in std_logic;
    reset:      in std_logic;
    sel_dmareg: in std_logic;
    s_write:    in std_logic;
    c_data_out: in std_logic_vector(31 downto 0);
    azzera:     out std_logic
    );
    end component ;
component interrupt port ( clk:                    in std_logic; reset:                  in std_logic;
sel_intreg:             in std_logic; s_write:                in std_logic; s_read:                 in
std_logic; c_data_out:             in std_logic_vector(31 downto 0); fifoout_rd_exc:         In std_logic;
fifoout_wr_exc:         In std_logic; fifoin_rd_exc:          In std_logic; fifoin_wr_exc:          In
std_logic; end_trx:                In std_logic; trx_stop:               In std_logic; trx_abort:
In std_logic; slow_int:               in std_logic; write_end:              in std_logic; int_reset:
Out std_logic; int_request:            Out std_logic; Vector_Mask:            Out std_logic_vector(8 downto 0);
Interrupt_Vector:       Out std_logic_vector(8 downto 0)); end component;
---- signals    declaration used on the diagram ----
--- addr signal--
signal SEL_TAR,SEL_DCR,SEL_DMAREG,SEL_MEMREG1,SEL_MEMREG2,SEL_NSFP,SEL_INTREG:std_logic; --fifo signal -- signal
FIN_EMPTY,FIN_FULL,MEM_FIN_WREN,DMA_FIN_WREN: std_logic; signal
FOUT_EMPTY,FOUT_FULL,FIFOOUT_DOUT,MEM_FOUT_WREN,DMA_FOUT_RDEN:std_logic; signal FOUT_USEDW: std_logic_vector(9
downto 0); signal FIN_USEDW: std_logic_vector(7 downto  0); signal fifo_reset: std_logic; signal
FIFOIN_DOUT,FIFOOUT_DIN: std_logic_vector(31 downto 0);
----dma signals
signal TAR,DCR :std_logic_vector(31 downto 0); signal DMA_STATUS: std_logic_vector(21 downto 0); signal
PCI_TRX_ENABLE,START_TRX,END_TRX,TRX_ABORT,TRX_STOP: std_logic; signal FIN_WR_EXC,FOUT_RD_EXC:std_logic;
--- MODE--
signal tMODE: std_logic; signal tMASTER: std_logic; signal VERSO: std_logic;
--- memory controller signals ---
signal MEMIN,MEMOUT: std_logic_vector(31 downto 0); signal START_ADD:   std_logic_vector(19 downto 0); signal
FIN_RD_EXC,FOUT_WR_EXC:std_logic; signal STOPRUN:  std_logic; signal tram_ad: std_logic_vector(19 downto 0);
signal WRITEND: std_logic; --REGISTER SIgnals signal BURST:std_logic_vector(7 downto 0); signal
BURSTLEN:std_logic_vector(5 downto 0); signal INTERRUPTVECTOR:std_logic_vector(8 downto 0); signal MASKVECTOR:
std_logic_vector(8 downto 0); signal MEM_CTRL_STATUS: std_logic_vector(1 downto 0); signal TRXING:std_logic;
signal EMUSTOP: std_logic; signal NUMSLOWFRAME: std_logic; signal SLOW,DATA: std_logic; signal
SLOWINT:std_logic; signal RNW:std_logic;
-----RESET
signal AZZERA: std_logic; begin
    VERSO<=rnw or tmode;
    fifo_reset<=not(AZZERA);
    FIFOIN : fifo_256x32
  port map(
       clock => clk,
       data => c_data_out,
       empty => FIN_EMPTY,
       full => FIN_FULL,
       aclr => fifo_reset,
       q => FIFOIN_DOUT,
       rdreq => MEM_FIN_WREN,
       usedw => FIN_USEDW,
       wrreq => DMA_FIN_WREN
  );
FIFOUT : fifo_1024x32
  port map(
       clock => clk,
       data => FIFOOUT_DIN,
       empty => FOUT_EMPTY,
       full => FOUT_FULL,
       aclr => fifo_reset,
       q => m_data_in,
       rdreq => DMA_FOUT_RDEN,
       usedw => FOUT_USEDW,
       wrreq => MEM_FOUT_WREN
  );
DMA_BLOCK : dma
    port map(
    clk             =>  clk,
    reset           =>  AZZERA,
```

```
    m_write         =>  m_write(0),
    m_read          =>  m_read(0),
    s_write         =>  s_write(0),
    c_data_out      =>  c_data_out,
    sel_tar         =>  SEL_TAR,
    sel_dcr         =>  SEL_DCR,
    sel_dmareg      =>  SEL_DMAREG,
    fifoout_ndw     =>  FOUT_USEDW,
    fifoin_ndw      =>  FIN_USEDW,
    m_abort         =>  m_abort,
    m_channel       =>  m_channel,
    fin_full        =>  FIN_FULL,
    fout_empty      =>  FOUT_EMPTY,
    tar_reg         =>  TAR,
    dcr_reg         =>  DCR,
    master          =>  tMASTER,
    PCI_TRX_ENABLE   =>  PCI_TRX_ENABLE,
    fin_wr_en       =>  DMA_FIN_WREN,
    fout_rd_en      =>  DMA_FOUT_RDEN,
    fin_wr_exc      =>  FIN_WR_EXC,
    fout_rd_exc     =>  FOUT_RD_EXC,
    m_req           =>  m_req,
    m_reg           =>  m_reg,
    m_reg_sel       =>  m_reg_sel,
    m_reg_wr        =>  m_reg_wr,
    dma_status      =>  DMA_STATUS,
    end_trx         =>  END_TRX,
    trx_abort       =>  TRX_ABORT,
    trx_stop        =>  TRX_STOP,
    m_stop          =>  m_stop);
MEMORY : mem port map(
    clk             =>  clk,
    reset           =>  AZZERA,
    mode            =>  tMODE,
    mem_out         =>  MEMOUT,
    s_write         =>  s_write(0),
    sel_MEMREG1     =>  SEL_MEMREG1,
    sel_MEMREG2     =>  SEL_MEMREG2,
    c_data_out      =>  c_data_out,
    burst           =>  BURST,
    burst_lenght    =>  BURSTLEN,
    mem_ctrl_status =>  MEM_CTRL_STATUS,
    trxing          =>  TRXING,
    mem_in          =>  MEMIN,
    fifoin_rden     =>  MEM_FIN_WREN,
    fifoout_wren    =>  MEM_FOUT_WREN,
    fin_rd_exc      =>  FIN_RD_EXC,
    fout_wr_exc     =>  FOUT_WR_EXC,
    data_fromfifo   =>  FIFOIN_DOUT,
    data_tofifo     =>  FIFOOUT_DIN,
    fin_empty       =>  FIN_EMPTY,
    fout_full       =>  FOUT_FULL,
    write_end       =>  WRITEND,
    stop_emu        =>  EMUSTOP,
    slow_reg        =>  SLOW,
    data_reg        =>  DATA,
    RNW             =>  RNW,
    start_trx       =>  start_trx,
        ----memory control
    ram_ad          =>  tram_AD,
    ram_CE0         =>  ram_CE0,
    ram_adsc        =>  ram_ADSC,
    ram_we          =>  ram_WE,
    ram_oe          =>  ram_OE
    );
ADDRESS : addr_select
        port map(
    s_addr                => s_addr,
    sel_DMAREG            => SEL_DMAREG,
    sel_TAR               => SEL_TAR,
    sel_DCR               => SEL_DCR,
    sel_MEMREG1           => SEL_MEMREG1,
    sel_MEMREG2           => SEL_MEMREG2,
    sel_NSFP              => SEL_NSFP,
    sel_INTREG            => SEL_INTREG
    );
MODE : switch
        port map(
    clk             =>clk,
    reset           =>AZZERA,
    sel_MEMREG1     =>SEL_MEMREG1,
    s_write         =>s_write(0),
```

```
    c_data_out      =>c_data_out,
    emu_stop        =>NUMSLOWFRAME,
    stop            =>EMUSTOP,
    mode            =>tMODE
    );
DIREZIONE : tristate
        port  map(
    din             => MEMIN,
    dout            => MEMOUT,
    dir             => VERSO,
    dq              => ram_data
    );
REGISTRI : registro port map(
    clk                   =>clk,
    reset                 =>AZZERA,
    s_read                =>s_read(0),
    c_data_out            =>c_data_out,
    s_write               =>s_write(0),
    sel_DMAREG            =>SEL_DMAREG,
    sel_MEMREG1           =>SEL_MEMREG1,
    sel_TAR               =>SEL_TAR,
    sel_DCR               =>SEL_DCR,
    sel_MEMREG2           =>SEL_MEMREG2,
    sel_NSFP              =>SEL_NSFP,
    sel_INTREG            =>SEL_INTREG,
    tar                   =>TAR,
    dcr                   =>DCR,
    dma_status            =>DMA_STATUS,
    pci_trx_enable        =>PCI_TRX_ENABLE,
    fifoout_wr_exc        =>FOUT_WR_EXC,
    fifoin_wr_exc         =>FIN_WR_EXC,
    fifoout_rd_exc        =>FOUT_RD_EXC,
    fifoin_rd_exc         =>FIN_RD_EXC,
    start_add             =>tram_ad,
    burst                 =>BURST,
    RNW                   =>rnw,
    burst_len             =>BURSTLEN,
    start_trx             =>START_TRX,
    mem_ctrl_status       =>MEM_CTRL_STATUS,
    s_data_in             =>s_data_in,
    end_trx               =>END_TRX,
    master                =>tMASTER,
    mode                  =>tMODE,
    emu_stop              =>NUMSLOWFRAME,
    slow                  =>SLOW,
    data                  =>DATA,
    trx_abort             =>TRX_ABORT,
    trx_stop              =>TRX_ABORT,
    slow_int              =>SLOWINT,
    interrupt_vector      =>INTERRUPTVECTOR,
    vector_mask           =>MASKVECTOR
    );
ANNIENTA : resetta port map(
    clk                   => clk,
    reset                 => reset,
    sel_dmareg            => SEL_DMAREG,
    s_write               => s_write(0),
    c_data_out            => c_data_out,
    azzera                => AZZERA
     );
INTERRUPTHANDLER: interrupt port map( clk                          => clk, reset                         => AZZERA,
sel_intREG                => sel_INTREG, s_write                    => s_write(0), s_read
=> s_read(0), c_data_out                => c_data_out, fifoout_rd_exc             => FOUT_RD_EXC,
fifoout_wr_exc            => FOUT_WR_EXC, fifoin_rd_exc             => FIN_RD_EXC, fifoin_wr_exc
=> FIN_WR_EXC, end_trx                  => END_TRX, trx_stop                  => TRX_STOP, write_end
=> WRITEND, trx_abort                => TRX_ABORT, int_reset                => int_reset, int_request
=> int_request, slow_int                 => SLOWINT, Vector_Mask               => MASKVECTOR,
Interrupt_Vector          => INTERRUPTVECTOR );
ram_ad<=tram_ad; end architecture ;
```