CAPITOLO 4

Il concentratore ottico

In questo capitolo, che costituisce parte del mio contributo originale al sistema di acquisizione ed elaborazione dei dati dello spettrometro per muoni dell'apparato ATLAS, verrà illustrato in dettaglio il concentratore ottico. Viene presentata una analisi del protocollo asincrono della trasmissione dati tra il rivelatore e l'elettronica *off-detector* e sono descritte le caratteristiche del sistema di collaudo del link ottico G^2Link. Viene presentato uno sviluppo del modello VHDL della scheda per definirne le specifiche e simularne il comportamento. Sono inoltre mostrati i vari schemi a blocchi, la struttura dei registri interni e i diagrammi di flusso per le singole funzioni che il concentratore è in grado di eseguire.

Il concentratore nella struttura di acquisizione

Come già descritto nel capitolo precedente, il concentratore ottico fa parte della struttura *off-detector* di acquisizione dati riportata in figura 4.1 (in cui il concentratore è indicato con RX). Questa struttura è installata nella sala di acquisizione *USA15*, situata a circa 80 metri dal punto di interazione dei fasci nell'acceleratore LHC. In tale struttura il ROD gestisce i dati di *read-out* ed è responsabile di gestire uno dei 32 settori in cui lo spettrometro è diviso (considerandone la suddivisione in $\eta>0$ e $\eta<0$). La gestione delle informazioni di trigger è svolta, invece, dalle schede *Sector Logic*. Ognuno dei settori gestiti dal ROD corrisponde a 2 settori logici in cui è suddiviso il sistema di trigger.

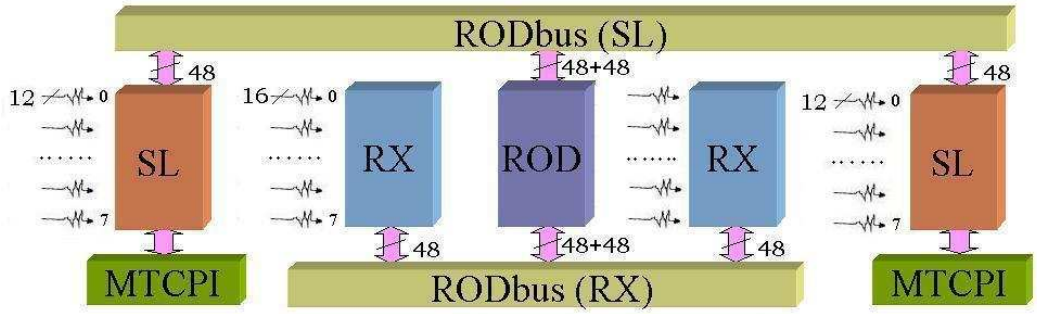

Figura 4.1 Schema del crate in cui è installato il concentratore

Il concentratore ottico riceve i dati di *read out* dalle schede PAD, fino ad un massimo di otto, tramite fibra ottica. Tali dati sono elaborati e trasferiti verso il ROD, su *backplane* dedicato, o verso il bus *VME*. La principale funzionalità del concentratore è la riorganizzazione dei dati di *read-out* in un unico pacchetto, come mostrato in figura 4.2. I dati sono selezionati in funzione degli stessi parametri di *bunch-crossing* e sono introdotti bit di controllo aggiuntivi. Eventi caratterizzati da uguali identificativi (*FEL1ID* e *FEBCID*), anche se giunti in istanti differenti al concentratore, vengono così riuniti in uno stesso *frame*, che costituisce l'uscita del concentratore

ottico. Tale procedura è detta "impacchettamento" o "*framing*". Il formato dei dati prodotti dal concentratore ottico è riportato nella figura 4.3

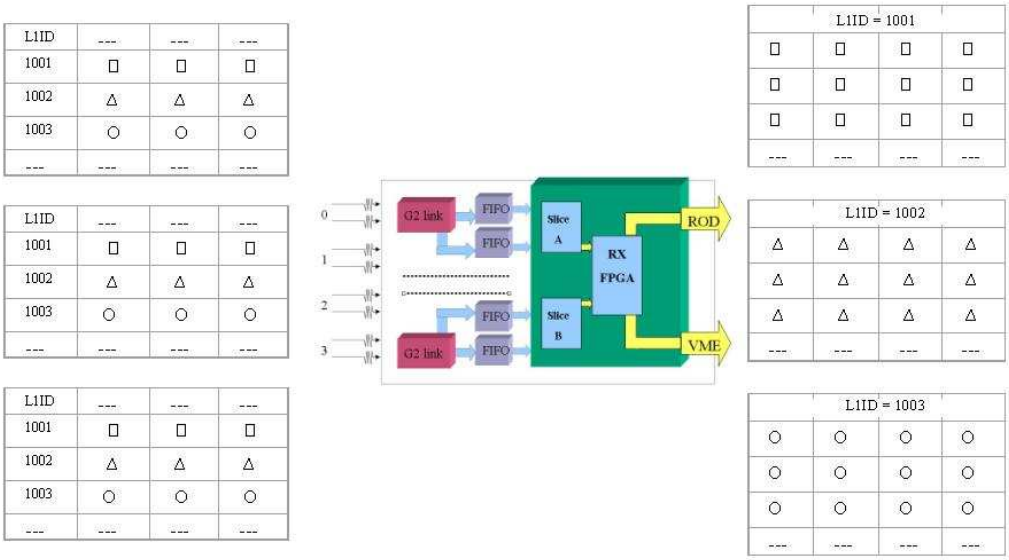

Figura 4.2 La funzione di *framing* del concentratore

Le altre funzionalità del concentratore ottico consistono in una procedura di controllo per il sistema di trasmissione dei dati sul *backplane* e in un sistema di comunicazione con il bus VME.

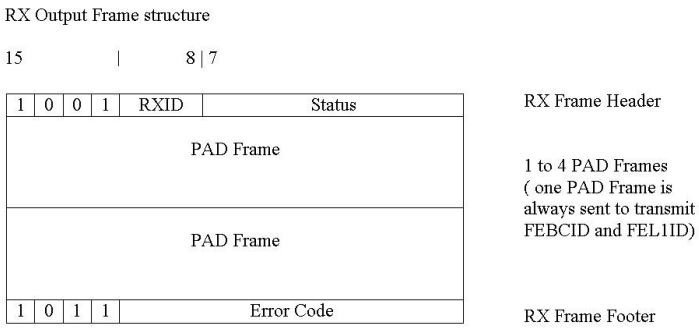

Figura 4.3 Il formato dei dati in uscita dal concentratore ottico

Il sistema di trasmissione dati viene controllato tramite l'invio di pattern predefiniti sul RODbus e verrà discusso in dettaglio nel prossimo capitolo. L'interfaccia VME consente la lettura dei registri interni del concentratore

ottico ed è utilizzata anche in fase di inizializzazione e collaudo della scheda. Infatti consente di verificare l'integrità dei componenti e il corretto funzionamento della logica all'interno del concentratore. E' inoltre possibile trasferire i dati in uscita senza processarli all'interno del concentratore, ma solo aggiungendo identificativi detti "*header*" e "*footer*".

I dati giungono al concentratore da PAD situate in postazioni diverse. E' possibile identificare una precisa relazione di fase tra il clock di ciascun trasmettitore di G^2Link, situato su ogni PAD, e il clock ricostruito dal ricevitore corrispondente, situato sul concentratore ottico. Tuttavia la differenza di fase tra i clock dei diversi link non è predicibile, in quanto dipende dalla lunghezza delle diverse fibre ottiche. A questo bisogna aggiungere la necessità di trovare una relazione di fase di tutti i clock ricostruiti con il clock della scheda. Per questo motivo, per consentire una corretta sincronizzazione dei dati provenienti dalle diverse PAD col segnale di clock del concentratore ottico, i dati sono immagazzinati in memorie *FIFO* (*First In First Out*) sincrone.

La figura 4.4 mostra la struttura dell'intera scheda. Come mostrato in figura 4.5, per ciascun canale del link ottico ci sarà una memoria *FIFO* e dunque un totale di 8 *FIFO* divise in due blocchi da 4.

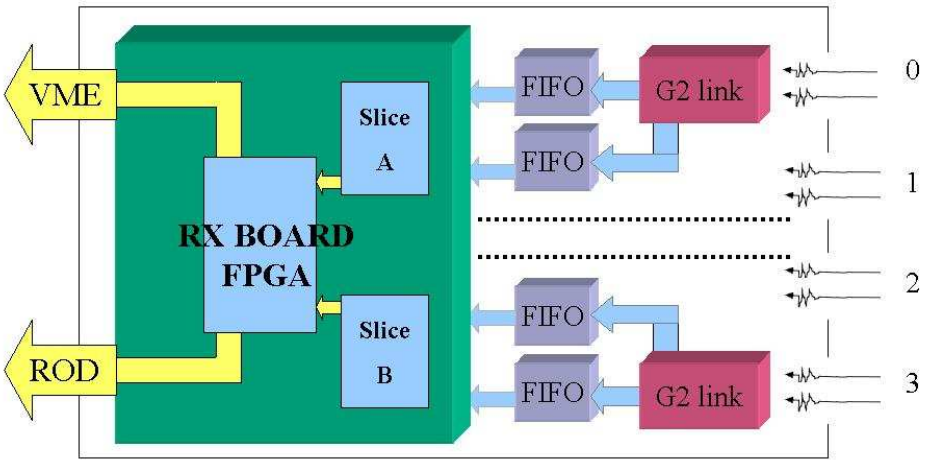

Figura 4.4 La struttura del concentratore ottico

Le memorie *FIFO*

Sulla scheda sono utilizzate le memorie *FIFO* della famiglia 72V3600, realizzate in tecnologia *CMOS*, prodotte dalla IDT. Queste *FIFO* funzionano seguendo protocolli sincroni di lettura e di scrittura. L'ingresso è controllato da un clock di scrittura (*WCLK*) e dall'ingresso *Write Enable* (*WEN**). I dati sono scritti nella *FIFO* su ciascun fronte di salita del *WCLK*, quando *WEN** è asserito. L'uscita della *FIFO* è controllata dal clock di lettura *RCLK* e dall'ingresso *Read Enable* (*REN**). Inoltre, l'ingresso *Output Enable* (*OE**) controlla lo stato di alta impedenza dell'uscita della *FIFO*. Se *OE** non è asserito, le uscite della *FIFO* sono poste in alta impedenza e i dati proposti in uscita dai registri interni della *FIFO* non sono disponibili all'esterno del dispositivo. Le frequenze di *WCLK* e di *RCLK* possono variare fino a 133 MHz e non c'è nessuna limitazione sulle relazioni reciproche tra le frequenze dei clock di lettura e di scrittura.

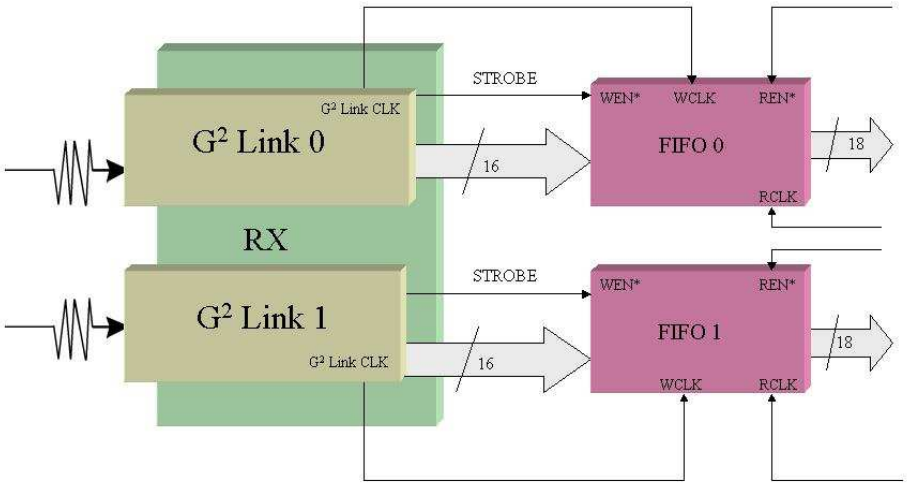

Figura 4.5 La connessione tra G^{2}Link e *FIFO*

I dati da scrivere nella *FIFO* ed il clock di scrittura provengono dal ricevitore del G^{2}Link. Si presenta l'esigenza di far coesistere il protocollo sincrono di trasmissione del G^{2}Link, che propone dati in uscita dal

ricevitore a ciascun periodo di clock, e la struttura intrinsecamente asincrona dei dati di *read out* del rivelatore. Tali dati devono essere trasferiti ai successivi livelli di elaborazione solo in presenza di un segnale di validazione (*L1A*) proveniente dal processore di trigger. La struttura dei dati trasmessi tramite il G^2Link è fondamentale per garantire un corretto trasferimento delle informazioni. Questi dati sono composti da parole di 16 bit più un bit di *strobe,* che svolge le funzioni di *Write Enable* per la *FIFO*. In questo modo sono scritti nella *FIFO* solo i dati per i quali il corrispondente bit di *strobe* è asserito. La figura 4.6 mostra la procedura di scrittura in una *FIFO*. I dati sono prodotti dal ricevitore del G^2Link in corrispondenza del fronte di discesa del clock ricostruito dal G^2Link. In tal caso, sul successivo fronte di salita del clock, i dati ed il bit di *strobe* sono stabili per poter essere scritti nella *FIFO*.

Al livello della scheda RX, dunque, le *FIFO* disaccoppiano la fase asincrona di trasmissione dei dati di *read-out* del rivelatore dalla successiva fase, sincrona, di elaborazione. Tuttavia, la gestione delle condizioni di *FIFO* vuota o *FIFO* piena richiede risincronizzazioni all'interno delle *FIFO* che introducono una ulteriore latenza di alcuni cicli di clock nella sequenza delle operazioni di trasmissione ed elaborazione dei dati.

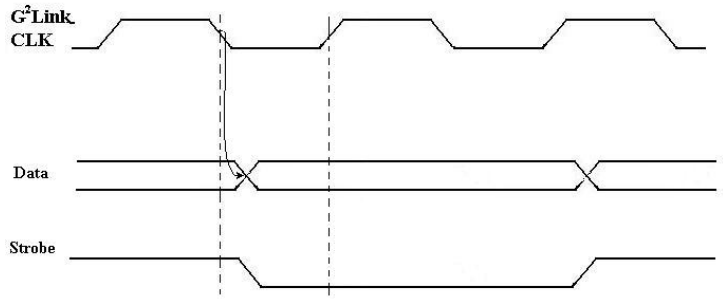

Figura 4.6 La procedura di scrittura in una *FIFO* sincrona

La progettazione

La fase di progettazione del concentratore ottico costituisce un mio contributo originale allo sviluppo del DAQ di ATLAS e rappresenta essenzialmente uno studio di fattibilità del sistema. E' stata valutata l'integrabilità di un elemento complesso come il concentratore ottico all'interno della struttura di acquisizione dati dell'esperimento ATLAS. Sono stati analizzati i principali problemi che si possono presentare nella realizzazione del concentratore ottico, dando risalto, in fase di progettazione, alla procedura di acquisizione dei dati all'interno della scheda e alla fase di comunicazione, su *backplane* dedicato, con i successivi elementi del sistema di elaborazione dati dell'esperimento. La progettazione del concentratore ottico è stata fatta avvalendosi di un CAD elettronico, che consente la simulazione e la realizzazione su componenti programmabili FPGA delle funzioni della scheda stessa. Inoltre, nella fase di progettazione è stato largamente utilizzato il linguaggio di programmazione VHDL (*Very high speed integrated circuit Hardware Description Language*) [6]. Il VHDL è un linguaggio che si è andato affermando nel corso degli ultimi venti anni per la progettazione, lo sviluppo di modelli e la simulazione di dispositivi digitali. Il VHDL è stato sviluppato all'inizio degli anni '80 allo scopo di avere un linguaggio standard per la descrizione di sistemi elettronici, che non fosse il tradizionale foglio di lavoro ("schematico") composto da porte logiche e flip-flop. Le principali utilità del VHDL sono i rapidi tempi di sviluppo nella stesura del codice ed una maggiore esportabilità verso differenti componenti.

La figura 4.7 mostra i due possibili utilizzi di un file VHDL. Il file può essere utilizzato per la simulazione e per uno studio sulle possibilità di realizzazione del progetto. E' inoltre possibile procedere alla effettiva realizzazione del progetto su componenti programmabili FPGA o su ASIC. Infatti, un progetto in VHDL consiste in un file che può venire successivamente elaborato da uno dei programmi commerciali di

compilazione. Questa procedura è detta sintesi. E' questo programma di sintesi che produce, dal file VHDL, una *netlist*, cioè una descrizione puntuale della struttura hardware composta da porte logiche e flip-flop interconnessi. Il fatto che sia un programma di sintesi a generare la *netlist* del progetto è un enorme vantaggio che offre la progettazione in VHDL. Infatti questo tipo di progettazione svincola il progettista dall'analisi puntuale del circuito e dal controllo delle macchine a stati, nonchè dalla riduzione delle classi di equivalenza delle macchine e dalla minimizzazione degli stati. Inoltre l'evoluzione delle tecnologie di produzione di componenti elettronici fa sì che, nel giro di pochi anni, un progetto debba essere più volte spostato verso i nuovi componenti all'avanguardia. Per restare al passo con le nuove tecnologie è quindi cruciale un approccio che sia, come il VHDL, il più possibile indipendente dalla tecnologia. La procedura di sintesi permette di esportare un progetto verso nuovi componenti senza dover eseguire modifiche, utilizzando soltanto le opportune librerie di sintesi.

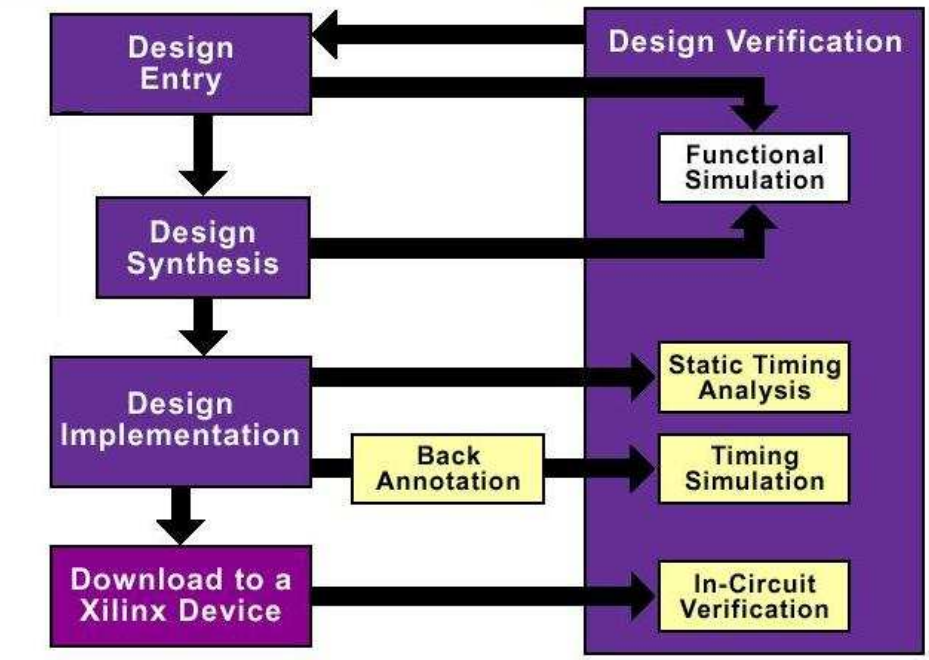

Figura 4.7 I possibili utilizzi di un file VHDL

La *netlist* subisce una successiva elaborazione che consiste nella realizzazione sul dispositivo programmabile FPGA del progetto. Questa procedura è detta *implementation*. Durante la fase di *implementation* sono calcolati anche i ritardi di propagazione dei segnali all'interno del dispositivo programmabile FPGA, tramite la procedura detta *Back Annotation*. Viene così prodotto un modello del progetto che può essere ulteriormente simulato oppure realizzato in un dispositivo programmabile.

Il concentratore ottico

La figura 4.8 mostra lo schema a blocchi del concentratore ottico, nel quale sono evidenziati i differenti elementi che gestiscono la logica.

La fase di lettura dei dati dalle *FIFO* è gestita da due blocchi logici identici e questa scelta è dettata essenzialmente da due motivi.

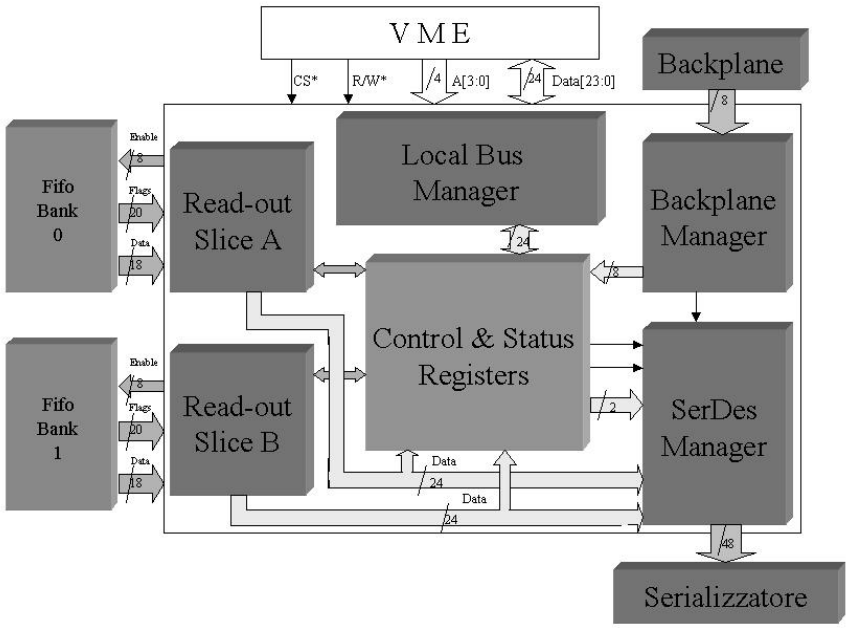

Figura 4.8 Lo schema a blocchi del concentratore ottico

Dovendo gestire otto *FIFO*, una per ciascun link ottico collegato alla scheda, è stato ritenuto opportuno limitare a quattro il numero di *FIFO* che ogni blocco logico può ispezionare. Infatti la fase di lettura di otto *FIFO* da

parte di un unico blocco potrebbe far trascorrere un numero considerevole cicli di clock. Come conseguenza, sebbene le *FIFO* abbiano una capacità di alcune decine di *Kbyte*, qualche *FIFO* potrebbe riempirsi e portare a perdita di sincronismo e di dati. Inoltre i dati uscenti dalle *FIFO* sono trasferiti nella FPGA tramite un bus. Dover gestire la lettura delle otto *FIFO* con un unico bus di comunicazione limita le prestazioni in frequenza. Infatti, la gestione di otto *FIFO* piuttosto che quattro comporterebbe un sostanziale aumento delle connessioni costituenti il bus. Questo potrebbe rendere problematico il funzionamento a 40 MHz, a causa della conseguente crescita dei fenomeni di interferenza e di effetti di capacitivi legati all'aumento della lunghezza delle piste e ad un maggior carico delle linee. Ulteriore motivazione per la lettura di quattro *FIFO*, piuttosto che otto, è rappresentata dalla struttura di lettura dati del ROD. Come si vede dalla figura 4.9, un ROD riceve i dati da ciascuno dei due blocchi logici di una scheda RX. Il suo compito principale è eseguire un impacchettamento di tali dati e quindi svolge una funzione analoga al concentratore. La gestione dei dati di quattro *FIFO*, dunque, rende plausibile anche una successiva esportazione della struttura logica del concentratore al ROD.

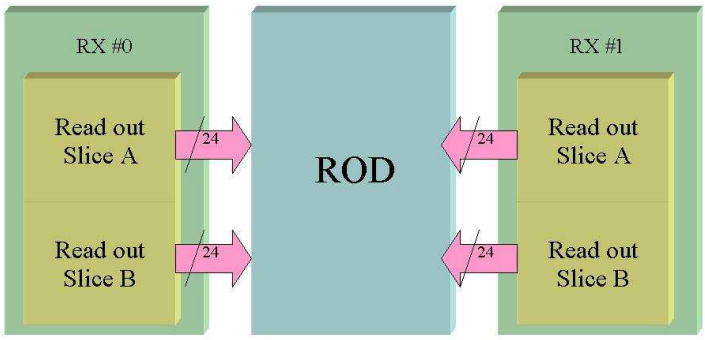

Figura 4.9 La struttura di lettura 4 a 1 del ROD

La logica di ingresso e trattamento dei dati è gestita dai due blocchi indicati con *Slice A* e *Slice B*, ciascuno dei quali acquisisce 18 bit (16 bit di dati + 2 bit di *strobe*) da una delle quattro *FIFO* che ogni *slice* può leggere. Con l'aggiunta di altri 6 bit di controllo all'interno di ciascuna s*lice* si

ottengono i 24 bit che ogni *slice* invia verso il ROD, tramite serializzatori, sul *backplane* dedicato (*RODbus RX)*, oppure verso il bus *VME*. Le principali informazioni legate al funzionamento del concentratore sono contenute nel blocco di registri *Control & Status Registers (CSR)*, che sarà ampiamente descritto nel corso di questo capitolo. Oltre alle *flag* di stato delle *FIFO (Empty FIFO = FIFO* vuota *o Full FIFO= FIFO* piena) il *CSR* contiene anche informazioni sulle diverse procedure che il concentratore può eseguire, come la selezione tra trasmissione verso *VME* o verso il ROD, oppure le istruzioni per il funzionamento in modalità di diagnostica (*Self test*) che la scheda è in grado di svolgere. Il concentratore, all'interno di ciascuna *Read Out Slice*, esegue anche un controllo sulla corretta trasmissione dei dati, analizzando la sequenza di *header* e *footer* che sono contenuti nei *frames* di ingresso.

Gli altri blocchi visibili in figura 4.8 sono il *SerDes Manager*, che consente la trasmissione dei dati ai serializzatori, il *Backplane Manager*, che comunica con il ROD scambiando i segnali di servizio come LV1A, BCR, ECR, ed il *Local Bus Manager*, che consente la comunicazione della scheda con un bus VME. Nel corso del capitolo questi blocchi saranno analizzati in dettaglio.

La *Read Out Slice*

La figura 4.10 mostra lo schema a blocchi di una singola *Read Out Slice*. Ciascuna *Slice* si occupa di un banco di 4 *FIFO*, di cui gestisce i segnali di *EMPTY FIFO** (che indica che la memoria *FIFO* è vuota) e *FULL FIFO** (che indica che la memoria della *FIFO* è piena), asserendo a sua volta i segnali di *READ ENABLE** e *OUTPUT ENABLE**.

Il bit di *READ ENABLE**, se asserito, permette che i dati immagazzinati nella *FIFO* vengano promossi nei *buffer* di uscita della *FIFO* in corrispondenza di ciascun fronte di salita del clock di lettura. Il bit di

*OUTPUT ENABLE** quando asserito, abilita i driver tristate di uscita e permette che i dati nei *buffer* di uscita di una *FIFO* possano pilotare le linee di uscita. Affinchè i dati siano letti dalla *FIFO* è dunque necessario che siano asseriti sia *READ ENABLE** che *OUTPUT ENABLE**. Inoltre, il ciclo di lettura dovrà avvenire quando la *FIFO* non è vuota, e dunque non dovrà essere asserito il bit di *EMPTY FIFO**. La gestione delle *FIFO* è affidata al blocco *FIFO MANAGER*. Tramite i 4 bit di *FIFO MASK*, controllati attraverso *VME* dal *CSR*, è possibile selezionare le *FIFO* su cui lavorare, ad esempio per collaudare una *FIFO* per volta o per escludere *FIFO* che non sono collegate a ricevitori ottici.

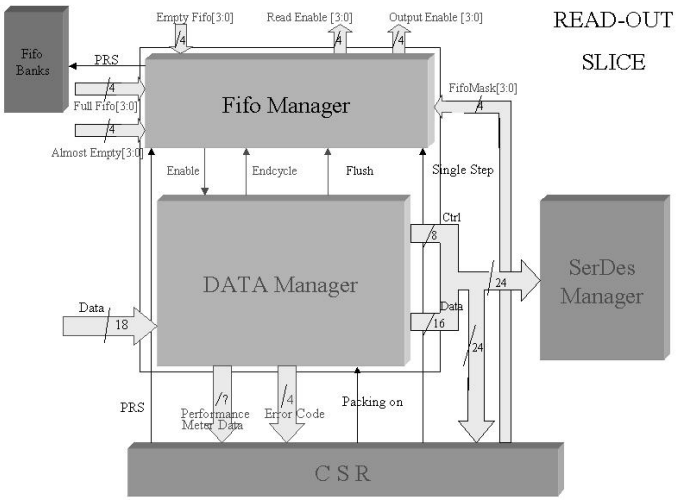

Figura 4.10 Lo schema a blocchi della *Read Out Slice*

Il blocco *DATA MANAGER* si occupa del trattamento dei dati. I dati in ingresso sono suddivisi in 16 bit di dati + 2 bit di controllo: in uscita dal *DATA MANAGER*, invece, ai 16 bit di dati sono aggiunti altri 8 bit di controllo, che possono essere trasferiti al *SERDES MANAGER*, responsabile della gestione dei serializzatori, o al *CSR*, che provvede alla successiva trasmissione al *LOCAL BUS MANAGER* che comunica col *VME*. I bit di controllo sono, tra gli altri, i due bit necessari per identificare la *FIFO* letta, un bit di errore, il bit che identifica quale delle due *Read Out*

Slice è selezionata o il bit che comunica che una *FIFO*, pur non essendo vuota, non contiene dati riguardanti il giusto *frame*. Il *DATA MANAGER* trasferisce inoltre al *CSR* un codice di 4 bit riguardante errori sui dati (*ERROR CODE*). Il *CSR* fornisce alla *Read Out Slice* il bit di *Packing On*, che abilita l'impacchettamento dei dati (altrimenti promossi in uscita senza essere trattati), il bit di *Run/Single Step*, che fornito al *FIFO MANAGER* permette di leggere dalle *FIFO* selezionate una sola parola (o un singolo *frame*) da trasmettere in uscita, e il bit *PRS (PARTIAL RESET*) che funziona da reset per le *FIFO* cancellandone il contenuto. Tra il *FIFO MANAGER* e il *DATA MANAGER* vengono scambiati i seguenti segnali:

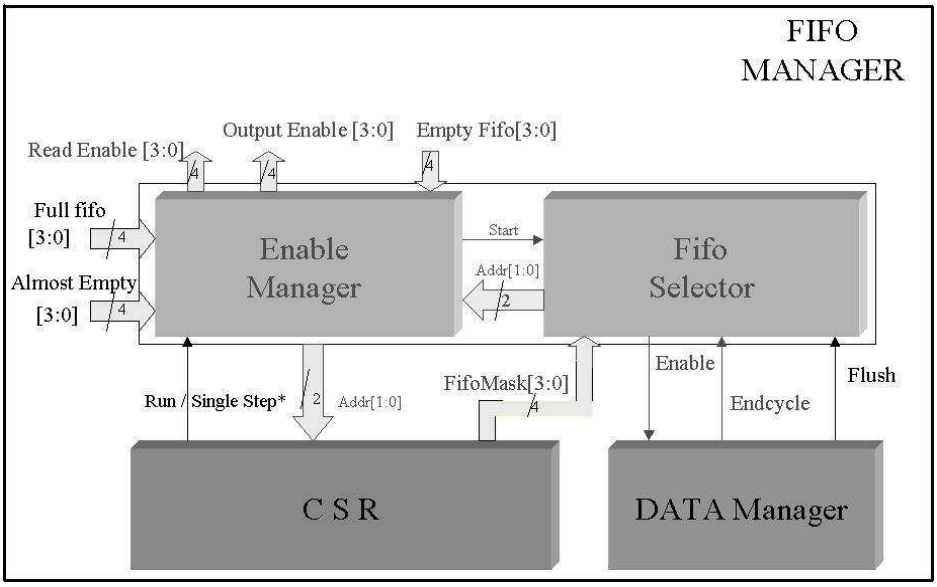

Figura 4.11 Lo schema a blocchi del *FIFO MANAGER*

• *Enable*, che abilita il *DATA MANAGER* a processare i dati;

• *Endcycle*, che indica che un intero PAD *frame* in una *FIFO* è stato letto e consente al *FIFO MANAGER* di asserire l'*OUTPUT ENABLE** della successiva *FIFO* da leggere;

• *Flush*, che in caso di errore di lettura dalle *FIFO* può essere utilizzato per recuperare il sincronismo di lettura dei dati.

La figura 4.11 mostra lo schema a blocchi del *FIFO MANAGER*. Esso si compone di due blocchi: *ENABLE MANAGER* e *FIFO SELECTOR*. *ENABLE MANAGER* è il blocco che si interessa delle *flag* di stato delle 4 *FIFO* collegate a ciascuna *Read Out Slice*, ricevendo dunque i segnali *EMPTY FIFO** e *FULL FIFO**. Asserisce, inoltre, le abilitazioni per la lettura delle *FIFO*, cioè *READ ENABLE** e *OUTPUT ENABLE**. Quando almeno una delle 4 *FIFO* è non vuota, l'*ENABLE MANAGER* asserisce il segnale di *Start*, che fa partire la selezione delle *FIFO* da leggere, eseguita dal blocco *FIFO SELECTOR*. Tale selezione darà come risultato l'indirizzo *ADDR[1:0]* di una delle 4 *FIFO* da leggere, l'unica di cui verranno abilitate le uscite con *READ ENABLE** e *OUTPUT ENABLE**. Il blocco *FIFO SELECTOR* scambia col *DATA MANAGER* i già citati segnali *Enable, Endcycle* e *Flush*, mentre il blocco *ENABLE MANAGER* scambia col *CSR* il bit di *Run/ Single Step** e i due bit di indirizzo *ADDR[1:0]* della *FIFO* letta.

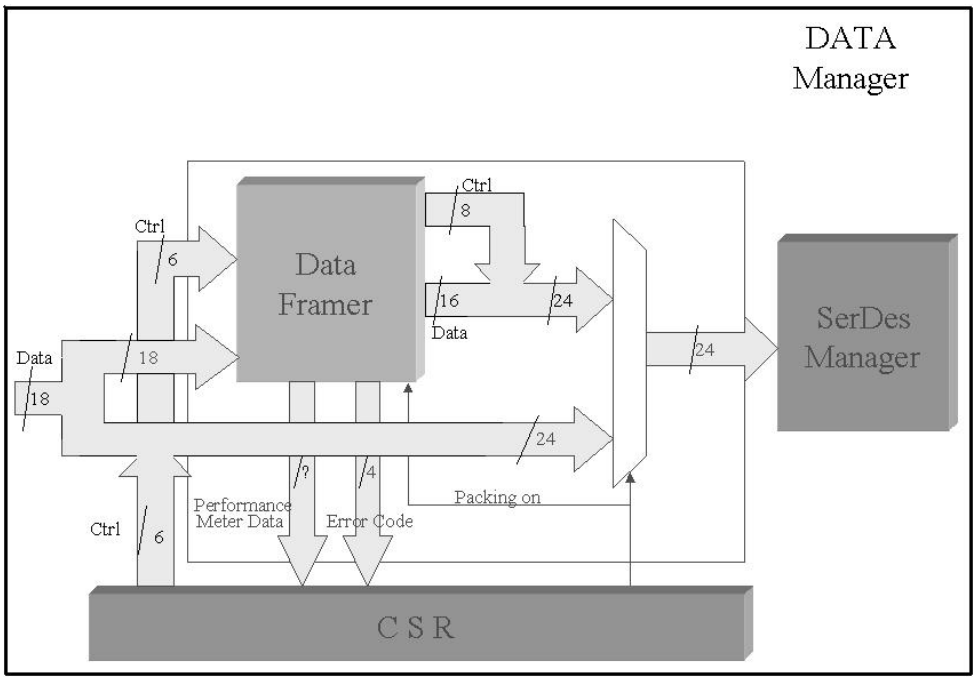

Figura 4.12 Lo schema a blocchi del *DATA MANAGER*

In figura 4.12 è riportato il percorso dati all'interno del *DATA MANAGER*. Con l'aggiunta di 6 bit di controllo, i 18 bit di ingresso (16 bit dati e 2 bit di controllo) formano parole da 24 bit. Tali parole possono essere trasferite direttamente in uscita, verso *VME* o verso serializzatori, oppure essere trattati all'interno del blocco *DATA FRAMER*. Il bit *Packing On*, fornito dal *CSR* anche al blocco *DATA FRAMER*, è responsabile della selezione che consente di trasmettere in uscita i dati impacchettati o meno.

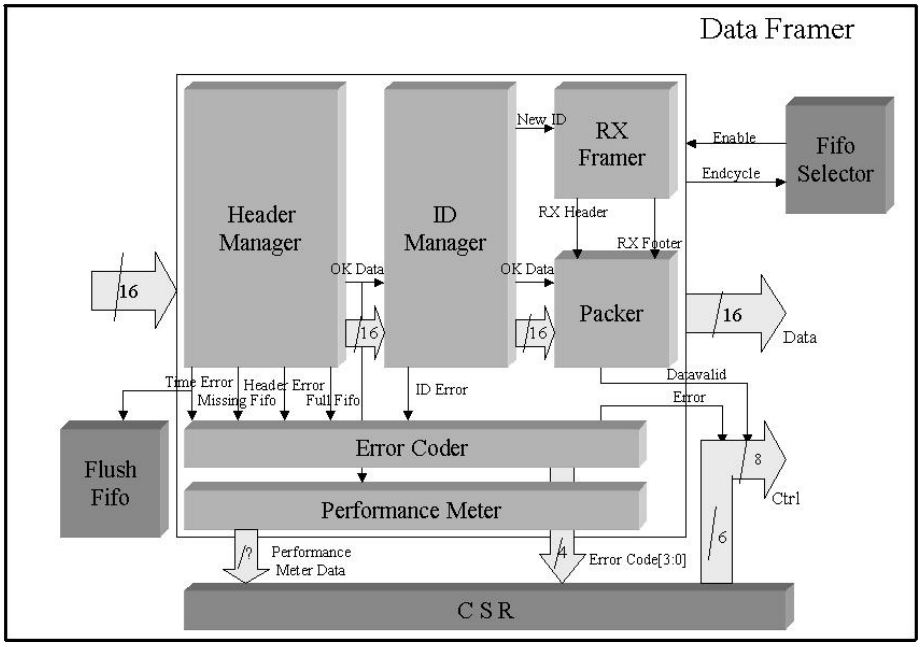

Figura 4.13 Lo schema a blocchi del *DATA FRAMER*

La figura 4.13 riporta una prima e preliminare struttura del *DATA FRAMER*. I 16 bit di dati in ingresso passano due livelli di controllo, uno relativo alla corretta successione di *header* dei *PAD frames* letti ed uno relativo alla coerenza di tutti gli identificativi all'interno del singolo *PAD frame*. Per ciò che riguarda il controllo degli *header*, nel caso di una corretta successione dei dati ci si aspetta, nei 4 bit più significativi, una sequenza ben definita (ad esempio come quella riportata in figura 4.14, che mostra il caso di un solo *CM frame* all'interno del *PAD frame*).

Header sequence

15	14	13	12	
0	1	0	1	*Pad header*
1	1	0	0	*CM header*
1	0	0	0	*CM subheader*
0	0	-	-	*CM data*
0	0	-	-	*CM data*
…	…	-	-	*CM data*
…	…	-	-	*CM data*
0	0	-	-	*CM data*
0	0	-	-	*CM data*
0	1	0	0	*CM footer*
0	1	1	1	*Pad footer*

Figura 4.14 La sequenza degli *header* in ciascun PAD *frame*, nel caso di un solo CM *frame* contenuto

Per quello che riguarda il controllo sugli identificativi, invece, osserviamo che ciascun CM *frame* contiene un *FEBCID* e un *FEL1ID*. Affinchè i dati di più CM *frames* all'interno di un unico PAD *frame* si riferiscano allo stesso evento, sarà necessario che in un unico PAD *frame* tutti gli identificativi *FEBCID* siano uguali tra loro, così come siano tutti uguali i *FEL1ID*.

Nel caso del rilevamento di un errore, il blocco *ERROR CODER* si occupa della trasmissione di un codice di errore al *CSR*. Gli errori possono verificarsi, oltre che per un errore di *header* o di *ID*, anche se una *FIFO* è piena o se da una *FIFO* non sono più proposti dati in uscita.

Il blocco *RX FRAMER* è responsabile della costruzione di *RX header*, *RX footer* e *RXID* dell' *RX frame*, mostrato in figura 4.15. Gli identificativi aggiunti ai dati in uscita sono " 1 0 0 1 " per *RX header* e " 1 0 1 1 " per *RX footer*. *RXID* è un identificativo di ciascuna delle schede RX.

L'impacchettamento viene realizzato dal blocco *PACKER*, che riceve *RX header* ed *RX footer* e i dati in ingresso, li raggruppa e li promuove in uscita insieme al bit *Datavalid*, distintivo della validità dei dati in uscita.

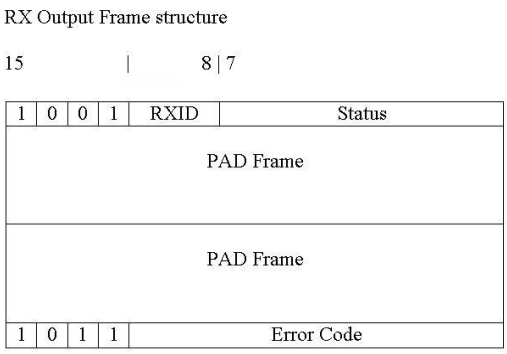

Figura 4.15 Il *frame* prodotto dal concentratore

A titolo di esempio è riportato in Appendice B il file in *VHDL* strutturale del componente *READ OUT SLICE*, mostrato in figura 4.10. Nel file non sono presenti i blocchi logici *RX Framer, ID Manager, Flush Fifo, Error Coder e Performance Meter* mostrati in figura 4.13. Il file consente la lettura dei dati dalle 4 *FIFO* che ogni *Read Out Slice* può leggere, proponendo un pacchetto in uscita. In particolare, il componente gestisce i segnali di *Empty Fifo**, *Output Enable** e *Read Enable** e permette di costruire pacchetti di parole da 24 bit. Ai 18 bit in ingresso sono aggiunti bit di controllo quali, ad esempio, i due bit di indirizzo della *FIFO* letta, un bit di errore e il bit di validazione dei dati.

Sono riportati nelle tavole 4.1-4.4 i fogli schematici e la simulazione su *CAD* elettronico del componente *READ OUT SLICE.* La simulazione nella tavola 4.3 mostra la fase di lettura dalle *FIFO*. In particolare sono asseriti i bit di *OUTPUT ENABLE** e *READ ENABLE** per l'acquisizione dei dati dalla *FIFO* identificata dai due bit di indirizzo *ADDR*. La simulazione nella tavola 4.4 mostra la fase di costruzione del pacchetto di dati in uscita. Sono riconosciuti in sequenza tutti gli *header* delle parole in ingresso: *PAD_HEADER*, *CM_HEADER*, *CM_SUB_HEADER*, *CM_FOOTER*, *PAD_FOOTER.* Si può notare come il bit di *strobe* conferisca validità solo ai dati interessanti per i successivi livelli di elaborazione.

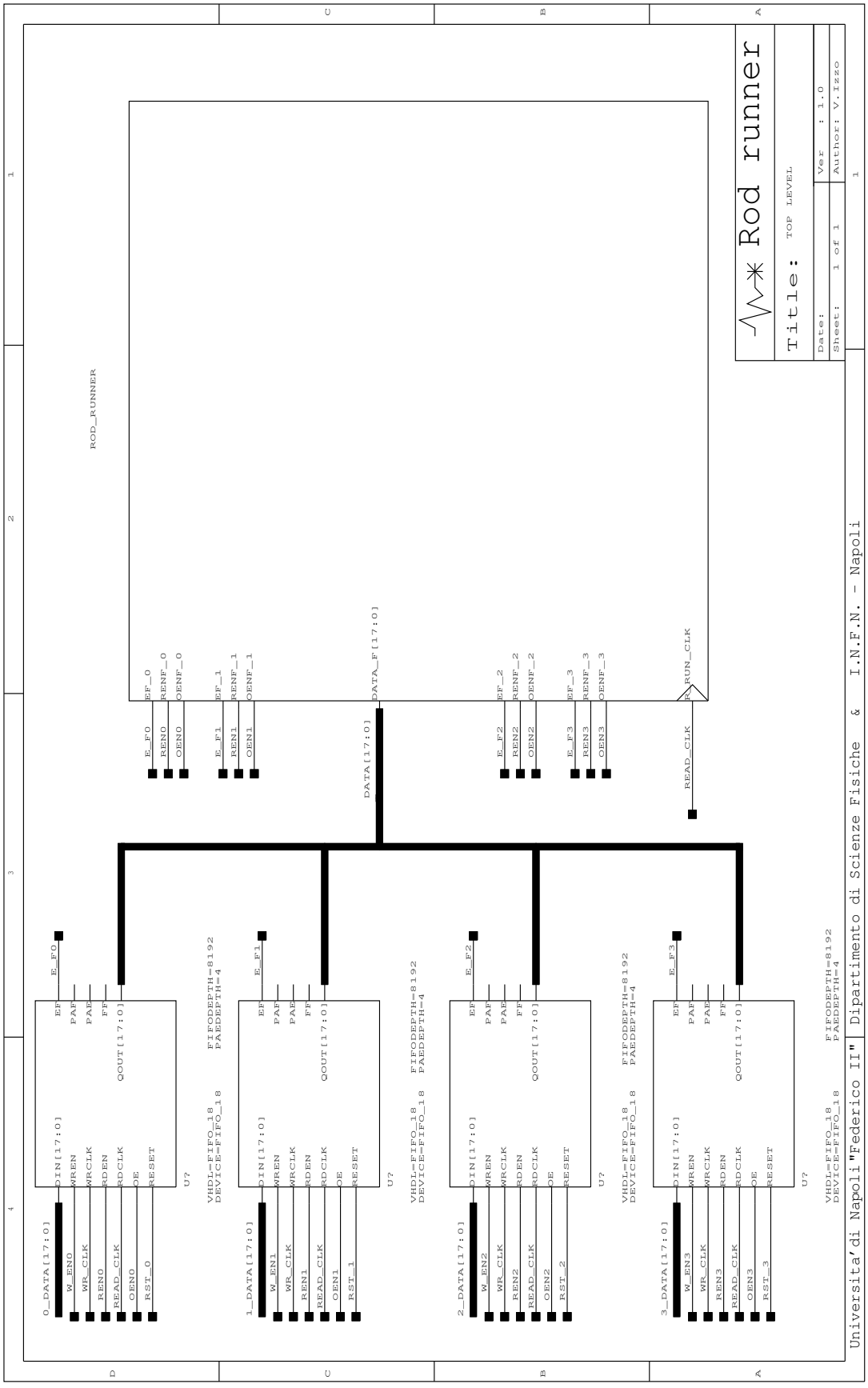

Tavola 4.1 Lo schematico con il concentratore e le *FIFO*

Tavola 4.2 Lo schematico con l'entità VHDL

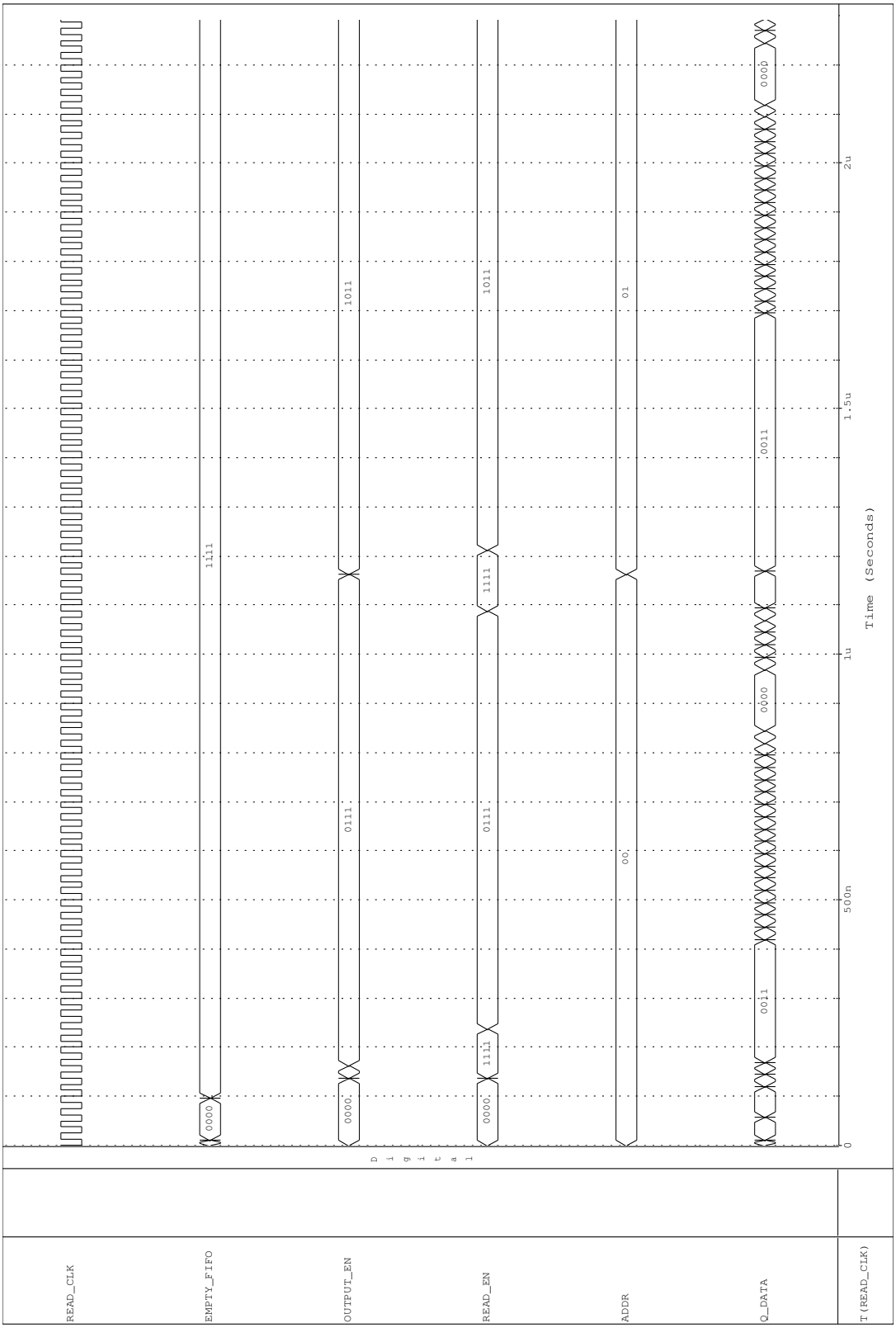

Tavola 4.3 La simulazione della lettura delle *FIFO*

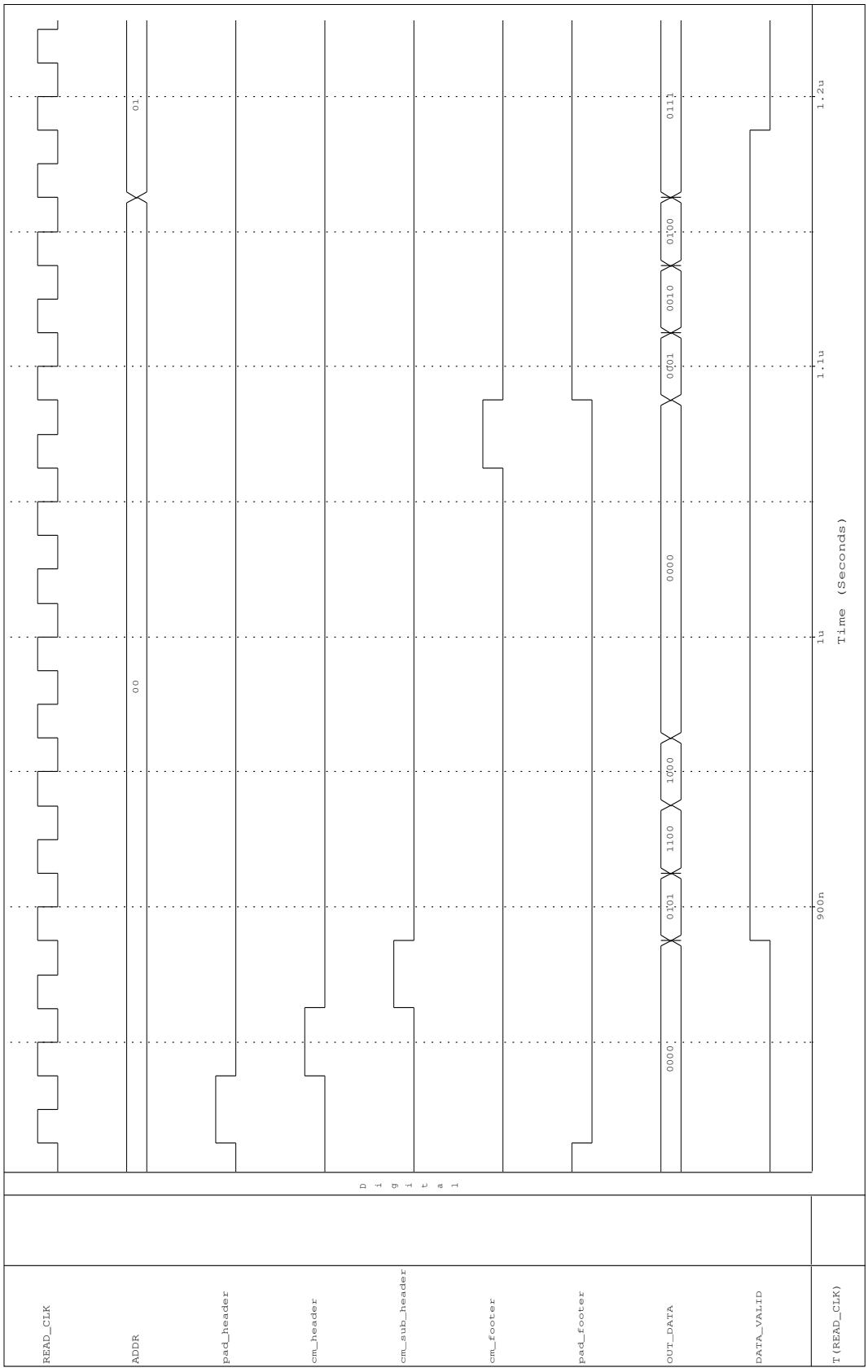

Tavola 4.4 La simulazione del framing dei dati di una *FIFO*

Il *SerDes Manager*

La figura 4.16 mostra il blocco *SERDES MANAGER*. I 24 bit prodotti da ciascuna *Read Out Slice*, divisi in 16 bit di dati + 8 bit di controllo, costituiscono parole da 48 bit. In uscita verso il serializzatore possono essere trasmesse queste parole da 48 bit oppure parole da 48 bit generate dal blocco *SELF TEST MANAGER*. I dati generati dal *SELF TEST MANAGER* danno la possibilità di utilizzare una procedura di diagnostica del processo di trasmissione dati. L'invio di un *pattern* di dati predefinito, infatti, permette di riconoscere, in fase di collaudo dell'intero sistema, eventuali errori di trasmissione o di realizzazione della scheda.

L'intera procedura di trasmissione in modalità *Self Test* verrà illustrata nel corso di questo capitolo. La selezione sulla trasmissione da effettuare avviene tramite il bit *RUN/SDT**, gestito dal *CSR*. All'interno del blocco *SELF TEST MANAGER* si trovano tre identici generatori di sequenze pseudorandom (*LFSR*): ciascuno genera una sequenza massimale di 2^{15}-1 parole di 16 bit.

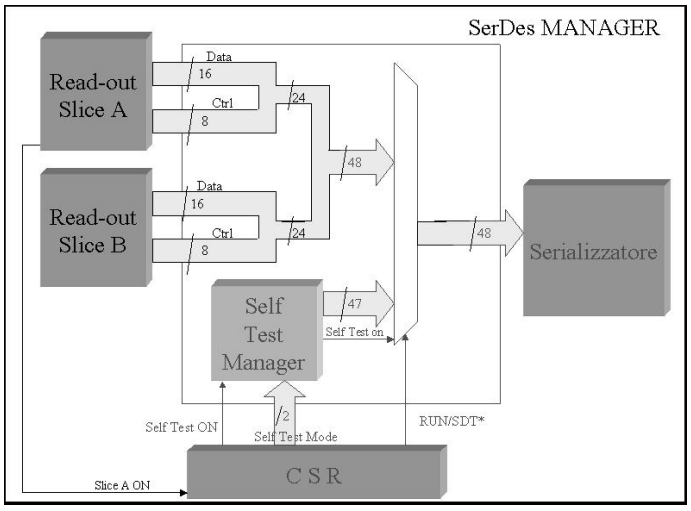

Figura 4.16 Lo schema a blocchi del *SERDES MANAGER*

Con i tre *LFSR* sarà dunque possibile generare parole di 48 bit, di cui 47 saranno trasmessi. Nel prossimo capitolo viene dedicato ampio spazio ai generatori di sequenze pseudorandom e alle proprietà di tali sequenze. Un

bit aggiuntivo è il bit *Self Test On*, che indica che i dati trasmessi dal serializzatore sono quelli generati dal *SELF TEST MANAGER*. Il *CSR* controlla, oltre al bit di *Self Test On*, anche un codice di 2 bit, *Self Test Mode*, che gestisce la procedura di *Self Test* e il cui significato verrà illustrato nel corso di questo capitolo.

Il *Local Bus Manager*

La comunicazione tra il concentratore ottico ed il bus *VME* avviene attraverso accessi in lettura e scrittura nei registri del *CSR*, effettuati tramite il blocco logico *Local Bus Manager*.

Oltre ai 24 bit su cui vengono trasmessi i dati contenuti nei registri del *CSR*, per le operazioni di lettura e scrittura sono necessari i bit *Chip Select C/S*,* il bit *Read/ Write R/W** e i 4 bit di indirizzo *A[3:0]* della figura 4.8. I bit di indirizzo selezionano uno dei possibili registri del *Control & Status Register* in cui è possibile andare a leggere o a scrivere dati. I bit *C/S** e *R/W** sono utilizzati per le procedure di lettura e di scrittura descritte in seguito. La figura 4.17 mostra il diagramma di flusso per un ciclo di lettura, mentre la figura 4.18 mostra quello per un ciclo di scrittura.

Una volta che *C/S** è = 0 vengono decodificati i 4 bit di indirizzo corrispondenti ad uno dei registri del *CSR*. Dopo la fase di indirizzamento viene controllato il bit *R/W**.

• Se *R/W** = 1 parte il ciclo di lettura dei dati dal registro selezionato. Tali dati vengono acquisiti nei registri di *I/O* (Ingresso / Uscita) e posti sul bus interno della scheda per la trasmissione sul bus *VME*. Quando *C/S** torna = 1 il bus dei dati viene riportato allo stato logico di alta impedenza e il sistema si predispone per un nuovo ciclo di lettura o di scrittura.

• Se *R/W** = 0 parte il ciclo di scrittura. In questo caso nei registri di *I/O* sono posti i dati che vengono successivamente scritti nel registro del *CSR*

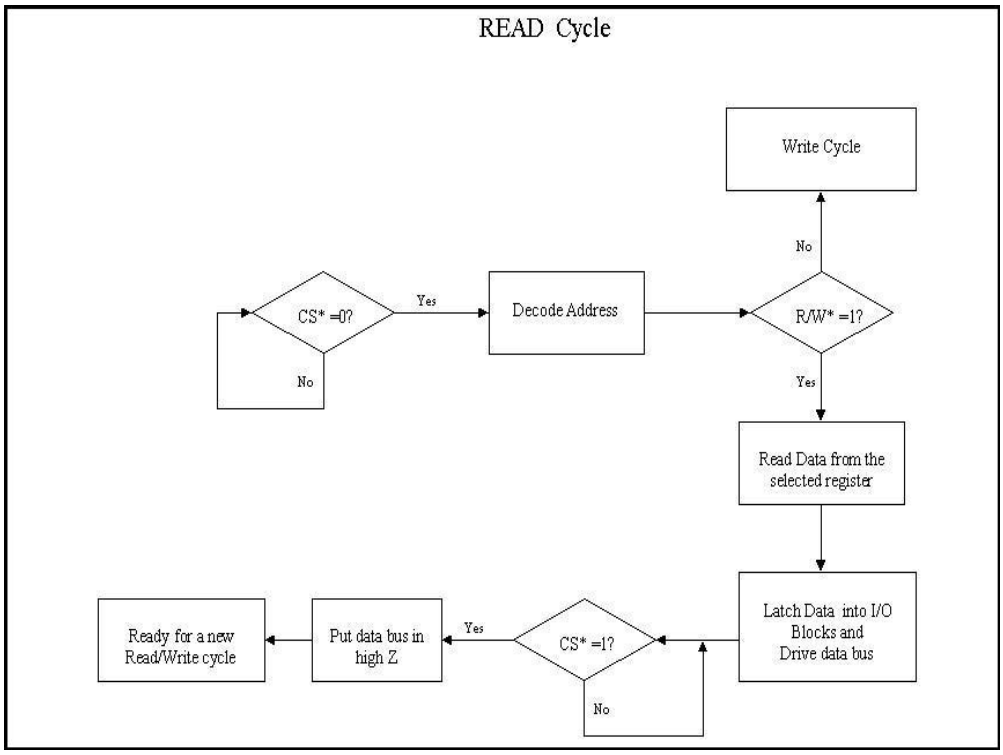

Figura 4.17 Il ciclo di lettura dal *CSR*

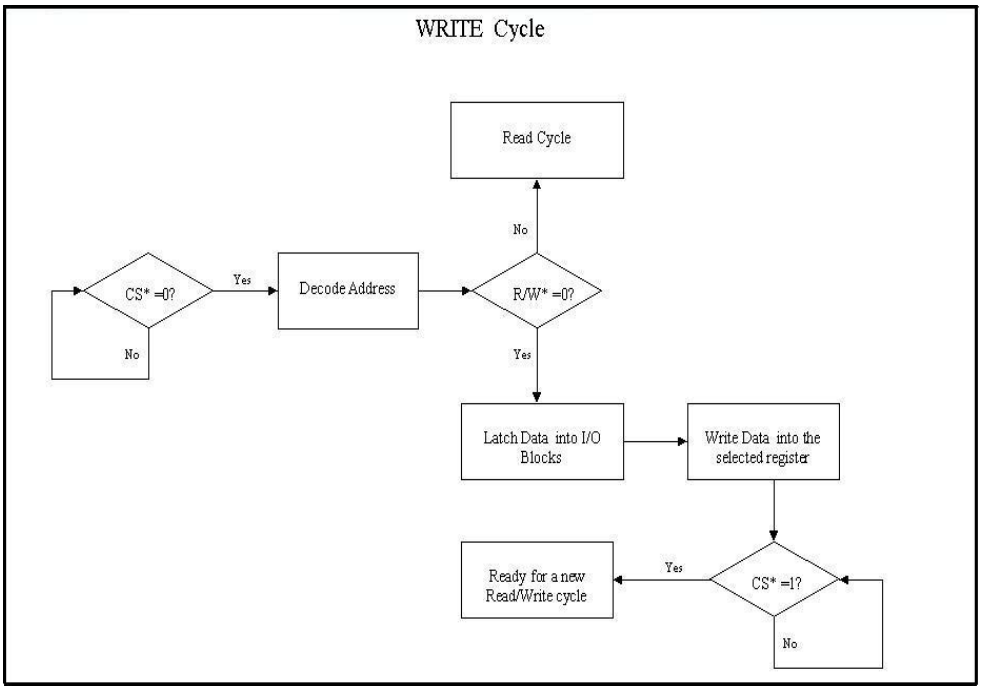

Figura 4.18 Il ciclo di scrittura nel *CSR*

selezionato dai bit di indirizzo *A[3:0]*. Quando *C/S** torna = 1 il bus dei dati viene rilasciato, riportandolo allo stato logico di alta impedenza e il sistema è pronto per un nuovo ciclo di lettura o di scrittura.

Il *Control & Status Register*

Nelle figure 4.19 sono riportati i contenuti del *Control & Status Register*. Come è stato appena descritto, l'accesso ai registri *CSR* avviene tramite *VME* attraverso il blocco *Local Bus Manager*.

Alcuni dei registri sono a sola lettura, come quelli delle *flag* di *EMPTY FIFO** o di *FULL FIFO**, in cui sono raggruppate in 8 bit le informazioni sulle *flag* delle *FIFO* connesse alle due *Slice* del concentratore.

Viceversa, il registro *FIFO MASK*, che consente di selezionare quali *FIFO* leggere e quali escludere dalla lettura, è un registro a cui si può accedere in lettura e in scrittura. Le *FIFO* vengono lette in successione, dalla 0 alle 3 di ciascuna *Slice*, a partire dalla prima *FIFO* caratterizzata dal valore di *FIFO MASK* = 0. Così, se il valore di *FIFO MASK0* = 0, la prima *FIFO* viene letta e poi in successione le altre *FIFO* per le quali *FIFO MASK* è uguale a 0; ad esempio, se *FIFO MASK0* = 1 e *FIFO MASK1*=0, la lettura delle *FIFO* comincia dalla seconda *FIFO*.

Altri due banchi di memoria del *CSR* sono il registro relativo alle informazioni del *backplane*, e cioè riguardanti i segnali di sincronizzazione *ECID* (che occupa 9 bit), *BCID* (12 bit), *L1A* (1 bit), *ECR* (1 bit), *BCR* (1 bit), e il registro relativo al blocco logico *DATA MANAGER*. Il registro che si riferisce al *DATA MANAGER* conterrà informazioni sulla procedura di impacchettamento (*Packing On*, 1 bit), sulla presenza di errori (*ERROR*, 1 bit) e il codice di 3 bit che identifica il tipo di errore (ad esempio un errore di successione di *header*, o di identificativi, *ECID* o *BCID*, o un errore riguardante assenza di dati nelle *FIFO* letta).

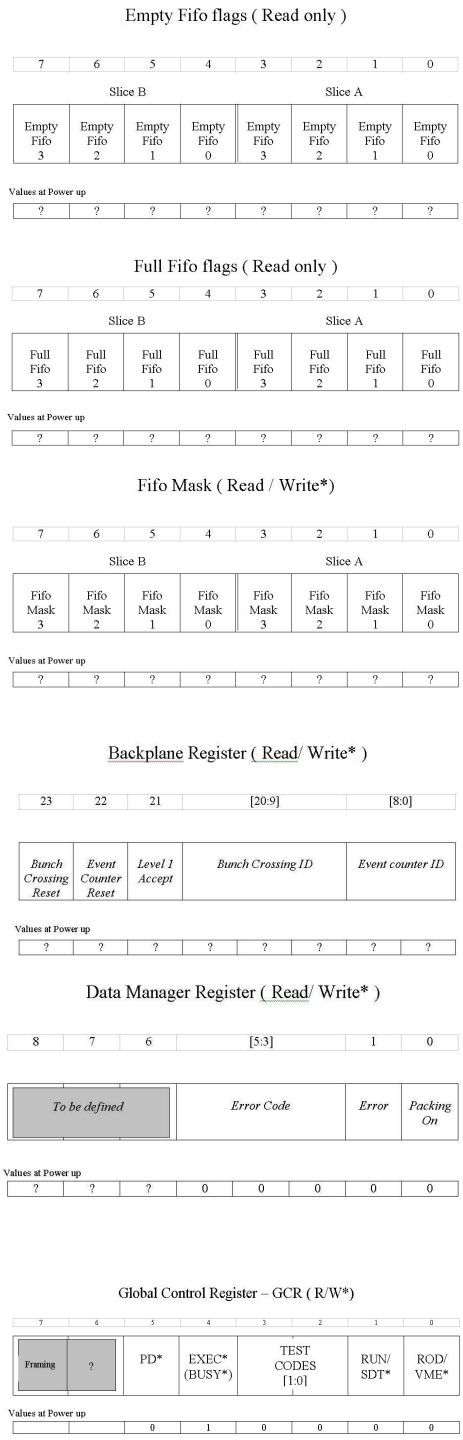

Figura 4.19 I contenuti dei registri del *Control & Status Register*

Infine c'è il registro *Global Control Register*, che controlla le operazioni che il concentratore ottico può eseguire. Come si può osservare dalla figura

4.20, gestendo tutti i bit del *Global Control Register*, dal bit meno significativo al bit più significativo, si può selezionare la funzione che il concentratore ottico esegue, come discusso in seguito.

Il *Global Control Register*

Il *Global Control Register* è il registro tramite il quale si possono definire le operazioni che il concentratore ottico può svolgere. Tali operazioni sono definite intervenendo sugli 8 bit che costituiscono il registro.

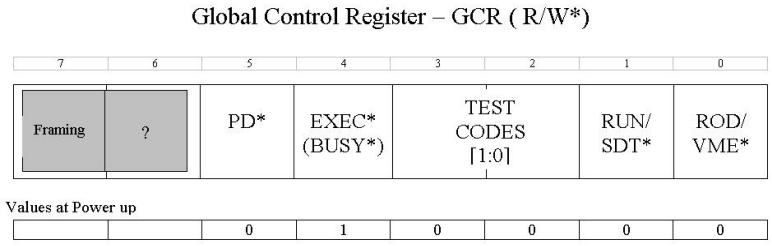

Figura 4.20 Il contenuto del *Global Control Register*

Il bit 0 è *ROD/VME**. Se *ROD/VME** =0 i dati sono indirizzati dalle *FIFO* verso il *VME*, in modalità *Single Step*, ossia prendendo, da ogni *FIFO* in una *Slice*, un singolo PAD *frame*: elaborando fino a 4 PAD *frames* viene prodotto un RX *frame*, da presentare in uscita su *VME*. Se *ROD/VME** =1 i dati sono diretti verso il ROD, tramite serializzatori: in tal caso, la selezione dei dati da trasmettere avviene col bit *RUN/SDT**. All'accensione, il sistema entra in modalità *VME* e dunque eseguirà la funzione *Single Step*, dopo che sarà stato fornito l'indirizzo della *Slice* da leggere. La figura 4.21 mostra lo schema del diagramma di flusso per la lettura da *VME*.

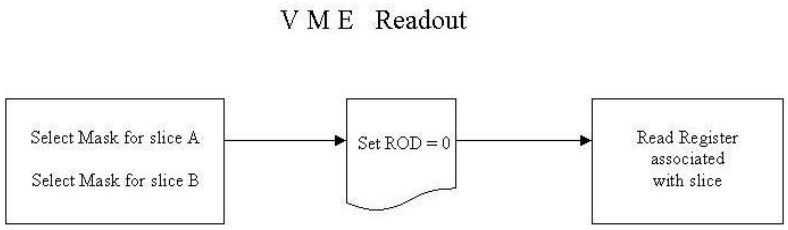

Figura 4.21 Il diagramma di flusso per la lettura da VME

Il bit 1 è *RUN/SDT**. Se *RUN/SDT**=0 il *SerDes Manager* produce i dati da trasmettere tramite il blocco logico *Self Test Manager*, in funzione del codice *TEST CODES [1:0]* e del bit *EXEC**. Se *RUN/SDT**=1 i dati sono diretti dalle *FIFO* verso il ROD: le *FIFO* sono lette in funzione dei dati scritti all'interno del registro *FIFO MASK*. All'accensione della scheda il sistema si predispone con *RUN/SDT**=0, ossia pronto per eseguire il test del serializzatori con il *pattern* prodotto dal *Self Test Manager*.

I bit 2 e 3 costituiscono il codice *TEST CODES [1:0]* che definisce la modalità di funzionamento *SerDes Test*. Il test è basato su un generatore di sequenze pseudorandom (*LFSR*) che genera una sequenza massimale di 2^{15}-1 parole di 16 bit. Il *LFSR* è replicato tre volte per creare parole da 48 bit. Il test viene eseguito quando il bit *EXEC** è basso.

I valori del codice *TEST CODES* sono:

0 0 = viene eseguito il reset del *LFSR*.

0 1 = *Single Mode*: viene generata una sola parola da 16 bit.

1 0 = *Burst Mode*: viene generata una intera sequenza, di 65535 parole.

1 1 = *Free Run Mode*: sono generate sequenze ininterrottamente.

La figura 4.22 mostra l'insieme delle procedure da seguire per portare il concentratore nella modalità *SerDes Test*.

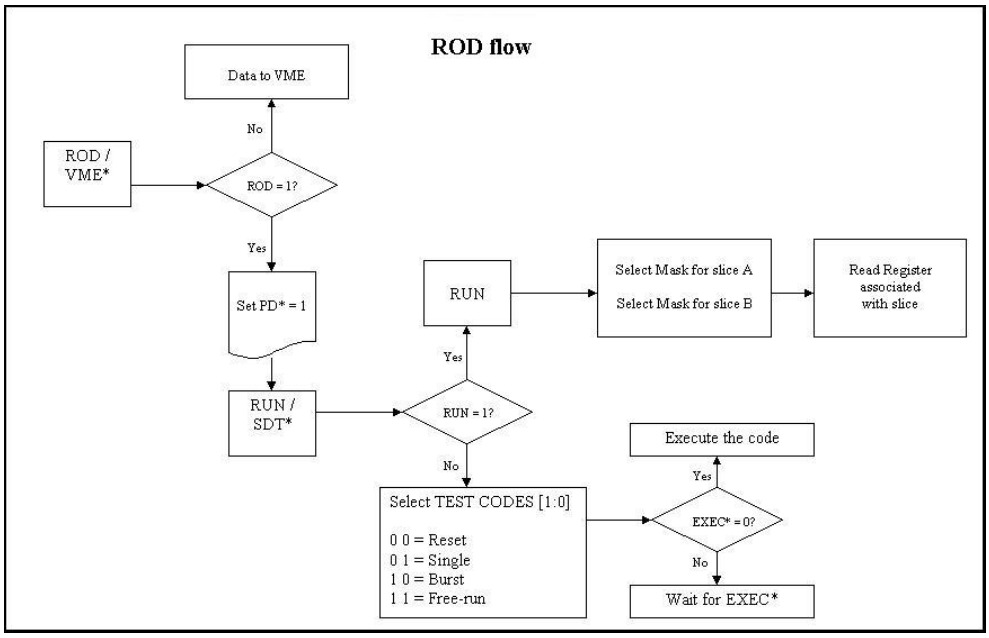

Figura 4.22 Il diagramma di flusso per la selezione del *SerDes Test*

Le operazioni *Single Mode* e *Burst Mode* non si possono interrompere, il *Free Run Mode*, invece, può essere interrotto dall'utente asserendo il bit *EXEC**=1. Inoltre nel *Burst Mode* e nel *Free Run Mode* viene eseguita una operazione di reset del *LFSR* e, dunque, le sequenze partono sempre dalla stessa parola. All'accensione della scheda il codice è "0 0". Il bit *EXEC** permette di eseguire il *SerDes Test* selezionato attraverso il codice *TEST CODES [1:0]*. Se *EXEC** = 0 il test viene eseguito ed il bit deve restare basso per tutta la durata del test. Se *EXEC** = 1 il sistema è pronto per eseguire il test. Inoltre è possibile asserire il bit =1 durante la generazione di sequenze in modalità *Free Run Mode* interrompendo, in tal caso, la generazione della sequenza. All'accensione della scheda il bit *EXEC** vale 1 e dunque disabilita l'esecuzione di qualsiasi funzione di *SerDes Test*. La figura 4.23 mostra l'insieme delle procedure da seguire per eseguire uno dei tre modi di funzionamento del *SerDes Test* o per l'invio dei dati su serializzatore.

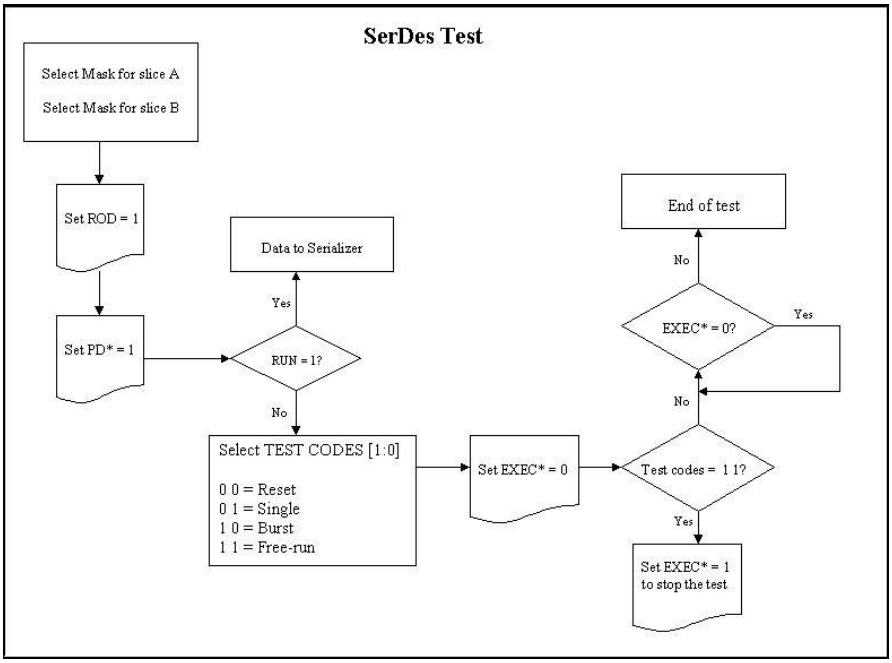

Figura 4.23 Il diagramma di flusso per l'esecuzione del *SerDes Test*

Il bit *PD** permette di abilitare o disabilitare la funzione *Power Down* dei serializzatori. Se *PD**=0 le uscite *LVDS* dei serializzatori vengono portate nello stato di alta impedenza (*TRI-STATE*). *PD** = 1 abilita le uscite *LVDS* dei serializzatori che, quindi, possono trasmettere i dati. All'accensione della scheda il bit PD* vale 0 e garantisce che non ci sia trasferimento dei dati verso il ROD.

I bit 7 ed 8 sono riservati per usi futuri o per successive correzioni alle procedure elencate.

Il collaudo del G^{2}LINK

Per il collaudo del link ottico G^{2}LINK è stata utilizzata una piattaforma di test realizzata per valutare il *Bit Error Rate* (*BER*). Nella tecnologia delle telecomunicazioni, il *Bit Error Rate* esprime la percentuale di bit che sono affetti da errore, rispetto al numero totale dei bit ricevuti durante un trasferimento di dati; tipicamente si esprime tramite una potenza negativa di dieci. Il *BER* è una indicazione di quanto spesso sia necessario ritrasmettere

un dato o un insieme di dati, a causa di una cattiva trasmissione, o di quanto spesso i dati vadano perduti, come nel caso dello spettrometro per muoni dell'apparato ATLAS in cui non è possibile una ritrasmissione. Valori tipicamente tollerabili sono di ~ $10^{-6} \div 10^{-14}$, corrispondenti ad un errore ogni $10^{6} \div 10^{14}$ bit trasmessi. Per il collaudo del sistema G^2Link è stata utilizzata una frequenza di trasferimento seriale di 40 MHz. In tal caso, trasferendo 800 Mbit/s, un *BER* di ~ $10^{-6} \div 10^{-14}$ si traduce in un numero di errori al secondo tra 800 e 8×10^{-6}.

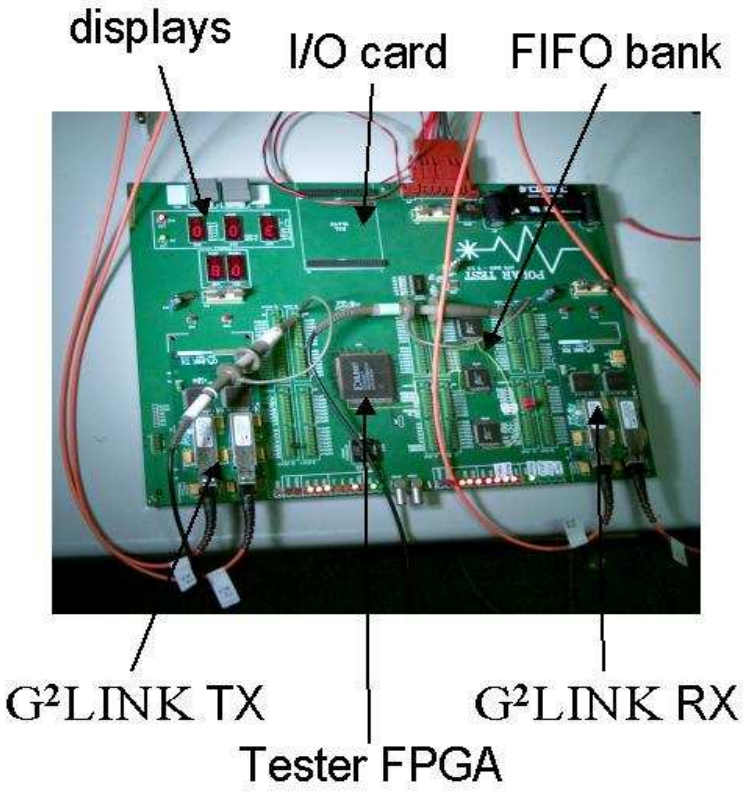

Figura 4.24 La piattaforma di collaudo per il link ottico

La piattaforma è costituita da una scheda, realizzata in FR4, che ospita un modulo trasmettitore G^2LINK TX ed un modulo ricevitore G^2LINK RX. Il sistema è alimentato a 5 V. Un convertitore DC-DC deriva la tensione a 3.3 V per l'alimentazione dei componenti del sistema G^2LINK.

La piattaforma di collaudo è gestita da una FPGA *XC4020E* prodotta dalla *Xilinx*. Il modulo consente di pilotare le linee di controllo dei dispositivi G^2LINK TX e G^2LINK RX.

La FPGA di test permette la selezione di differenti sequenze di dati da inviare, eseguendo un confronto tra i dati ricevuti e quelli inviati. Sono inoltre mostrati su *display* gli errori eventualmente verificatisi durante la trasmissione. Il *BER*, infatti, può dipendere da numerosi parametri, ed è dunque necessario esplorare, in fase di collaudo, tutte le situazioni che possono presentarsi e sottoporre il link ottico ad uno stress di trasmissione che possa far variare anche significativamente il *BER*.

In particolare, sono state esplorate tre situazioni differenti, a seconda della sequenza inviata:

• una sequenza per la quale, in una singola parola di 16 bit, tutti i bit sono uguali o a zero o a uno;

• una sequenza con parole in cui 15 bit sono uguali tra loro ed uno differente e tali che, al colpo di clock successivo, il bit differente si sia spostato di una posizione (dette "*walking one*" o "*walking zero*");

• una sequenza in cui nei 16 bit si susseguono alternativamente zero ed uno.

Le sequenze "costanti", cioè quelle che hanno una parola con tutti 0 (1) e la parola successiva con tutti 1 (0), così come le sequenze "*walking one*" o "*walking zero*", possono mettere in difficoltà il sistema di ricostruzione del clock, da parte del ricevitore.

Questa problematica è legata a variazioni del valore medio del segnale trasmesso e poi ricostruito e sarà analizzata in maggior dettaglio a proposito dei serializzatori, nel prossimo capitolo. Le sequenze con parole costanti, inoltre, permettono di ispezionare il campo dei quattro bit di controllo che vengono inviati insieme ai dati. Il sistema trasmettitore-ricevitore utilizza un meccanismo di recupero del corretto livello di tensione in continua. Una

sequenza di 16 bit costanti, dunque, è la situazione estrema di stress che può presentarsi per tale meccanismo di recupero.

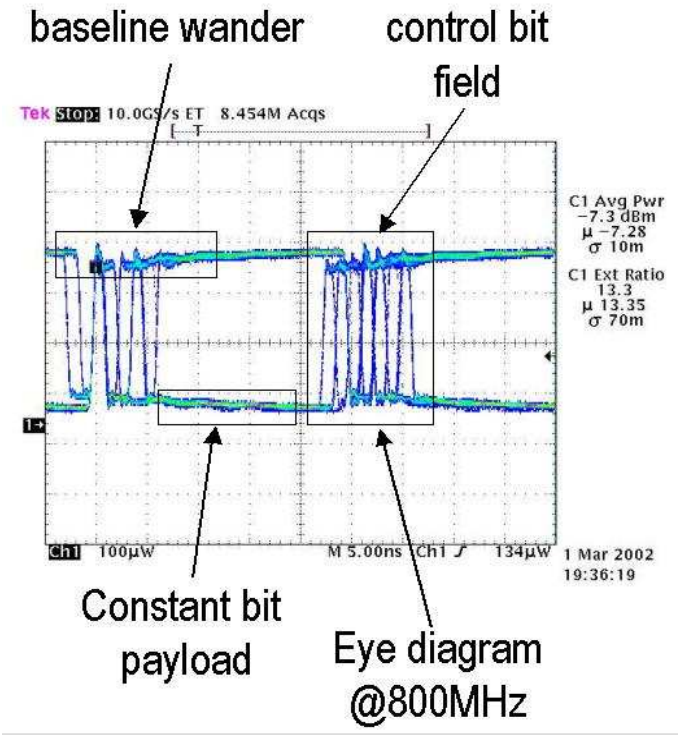

Figura 4.25 I risultati all'oscilloscopio del traffico dati relativo ad una sequenza con parole costanti

Tale comportamento del sistema di trasmissione si può rilevare valutando il cosiddetto *baseline wander* nella figura 4.25. Come si vede i livelli logici alto e basso non sono perfettamente orizzontali, a causa delle variazioni del livello di continua che tali particolari sequenze dati comportano. Queste variazioni possono portare alla saturazione del dispositivo di ricostruzione del segnale nel ricevitore del G^2Link.

Un altro parametro da tenere in considerazione è il *jitter*, temporale o in ampiezza, del segnale trasmesso. Una buona figura di merito per ottenere una stima di tali effetti è il cosiddetto "diagramma ad occhio". La figura 4.26 mostra il diagramma ad occhio per la situazione di "*walking one*".

Un diagramma ad occhio si può ottenere, tramite un oscilloscopio a *sampling*, a partire da un sistema ad alta frequenza che dia in uscita una

lunga sequenza pseudorandom (*PRBS*). Le sequenze pseudorandom sono particolari sequenze utilizzate in tecnologia delle telecomunicazioni per sollecitare determinati comportamenti di un sistema. Una dettagliata trattazione sarà presentata nel corso del prossimo capitolo, in cui viene analizzato in dettaglio il sistema di trasmissione dati sul *backplane*. Il diagramma ad occhio è una figura di merito che si ottiene integrando in una finestra temporale delle dimensioni di un periodo di clock le informazioni relative a più eventi del segnale acquisito. Il grafico che si ottiene da tale campionamento sarà una sovrapposizione di "zero" e di "uno", come risulta in figura 4.26.

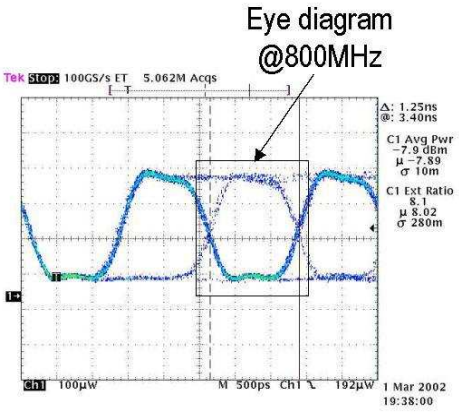

Figura 4.26 Diagramma ad occhio per una sequenza "*walking one*"

Tramite un diagramma ad occhio si possono ottenere informazioni sulle prestazioni del sistema di trasmissione. Ad esempio, la larghezza delle bande orizzontali fornisce informazioni sulle variazioni di ampiezza (*jitter*), o la pendenza delle transizioni permette di misurare i tempi di salita e di discesa degli impulsi.

La figura 4.27 mostra il diagramma ad occhio per una sequenza in cui vengono trasmesse parole che presentano una continua alternanza di zero e di uno ("*alternate payload*"). Tale sequenza garantisce la massima densità di transizioni sulla linea, ma in questa situazione il *baseline wander* è minimizzato.

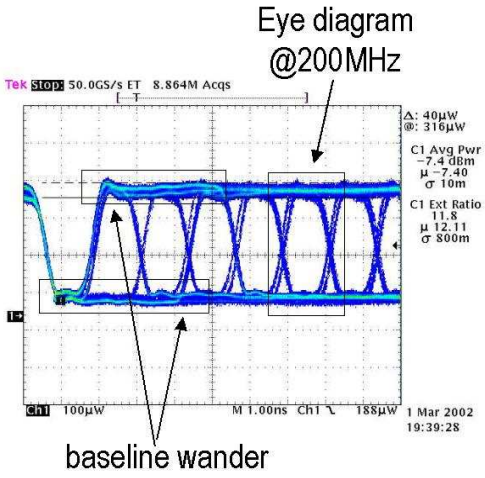

Figura 4.27 Diagramma ad occhio per la sequenza "*alternate payload*"

Il collaudo effettuato in laboratorio ed il *BER* ottenuto mostrano che sia l'effetto di *baseline wander* sia il verificarsi di *jitter,* temporali o in ampiezza, sono trascurabili. Infatti, durante il collaudo, durato oltre un mese, sono stati trasferiti oltre 4×10^{15} bit, ma non sono stati osservati errori.