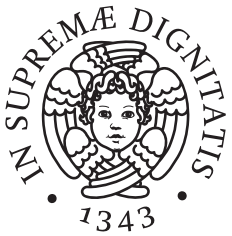
CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Electronic Thesis and Dissertation Archive - Università di Pisa



Università di Pisa

FACOLTÀ DI INGEGNERIA

Corso di Laurea in
INGEGNERIA ELETTRONICA

Tesi di Laurea Specialistica

Studio e progetto di un contatore multicanale configurabile ad alte prestazioni basato su piattaforma FPGA/DSP

Candidato
Andrea Lazzeri

Relatori
Prof. Roberto Saletti
Prof. Roberto Roncella
Ing. Federico Baronti

Anno Accademico 2006/2007

Ai miei genitori,
a Mario,
e a Marta.

Grazie, ancora una volta.

Indice

Introduzione

Nell'ambito del progetto di ricerca COFIN-PRIN "Development of Monolithic Photon-Counter Arrays for Transient High-Energy Phenomena and Adaptive Optics in Astrophysics" avviato nel 2003, il Dipartimento di Ingegneria dell'Informazione dell'Università di Pisa ha sviluppato, in una prima fase, un contatore multicanale a 60 canali, capace di raggiungere una finestre minima di integrazione di 20 µs con una profondità di conteggio di 16 b, e in grado di elaborare in tempo reale i risultati di volta in volta ottenuti.

Per contenere i costi e i tempi di implementazione, il sistema è stato realizzato utilizzando una scheda di sviluppo commerciale, basata su piattaforma FPGA/DSP. All'FPGA è stato affidato il compito di contare gli impulsi, mentre al DSP quello di elaborare in tempo reale i risultati di conteggio.

Successivamente, dopo alcuni sviluppi, è stato possibile raggiungere una finestra di 10 µs. La principale intuizione che ha portato ad un simile miglioramento è stata quella di ridurre la profondità di conteggio a 8 b.

Lo scopo di questa tesi è stato quello di tentare di migliorare ulteriormente le prestazioni del sistema, cercando di raggiungere quelle dei migliori contatori multicanale presenti attualmente sul mercato.

A tal fine, è stato opportuno studiare la dipendenza delle prestazioni dalla profondità di conteggio, e determinare le ipotesi sotto le quali, rendendola variabile, tali prestazioni possano effettivamente migliorare.

Successivamente, l'architettura dell'FPGA è stata completamente riprogettata, sia per rendere la profondità variabile, sia per sfruttare al meglio le risorse logiche disponibili; il firmware del DSP è stato ottimizzato in modo da velocizzare i tempi di elaborazione e rendere vere le ipotesi determinate nel corso del precedente studio.

Si è ottenuto in tal modo un nuovo contatore a 64 canali, capace di raggiungere una finestra minima di 24 µs, 14 µs e 8 µs, con una profondità di conteggio rispettivamente di 23 b, 16 b e 8 b, e ovviamente ancora in grado di elaborare i risultati in tempo reale.

Inoltre, tale contatore, se configurato per 2 canali, elaborando i dati in tempo reale fornisce delle prestazioni (relativamente al solo conteggio di impulsi) lievemente inferiori rispetto al migliore contatore multicanale presente in commercio, anch'esso a 2 canali. Si è ottenuto infatti 1.4 µs – 23 b, 744 ns – 16 b, 378 ns – 8 b, contro i 260 ns – 9 b del migliore contatore.

Viceversa, sempre configurato per 2 canali, ma senza elaborazione in tempo reale, il nuovo contatore risulta addirittura vincente, offrendo delle prestazioni di 378 ns – 23 b, 189 ns – 16 b, 88 ns – 8 b.

Nei capitoli che seguono si forniscono sia i fondamenti teorici necessari allo studio dei contatori multicanale, sia la descrizione dettagliata del progetto e delle scelte che hanno consentito di raggiungere i precedenti risultati.

Nel Capitolo 1 si descrive l'architettura generale di un contatore multicanale, definendone i principali parametri (quali numero di canali, massimo pulse-rate di ingresso, range e step di variazione della finestra di integrazione, profondità di conteggio e modalità di acquisizione); si riassumono le prestazioni offerte dai migliori contatori multicanale presenti attualmente sul mercato; si descrivono gli obiettivi precedentemente prefissati nell'ambito del progetto COFIN-PRIN, l'architettura della scheda di sviluppo utilizzata, e le prestazioni offerte dal vecchio contatore multicanale; si elencano i nuovi obiettivi e, infine, si anticipano i risultati ottenuti dal presente lavoro.

Nel Capitolo 2 si sviluppa un modello teorico che consente l'analisi generale delle prestazioni del sistema in funzione della profondità di conteggio; si determinano le ipotesi da verificare per garantire un significativo miglioramento delle prestazioni rendendo variabile tale profondità e, infine, si propone un grafico che consente di dedurre immediatamente le prestazioni del sistema, racchiudendo in sé tutte le informazioni da fornire a potenziali utilizzatori.

Nel Capitolo 3 si affronta dunque la nuova architettura implementata all'interno dell'FPGA: dapprima si descrivono i segnali relativi al bus di comunicazione tra FPGA e DSP, e si analizzano le scelte effettuate per gestire l'indirizzamento dei registri interni e i segnali di abilitazione alla lettura e alla scrittura; successivamente si evidenzia la suddivisione tra parte operativa (i vari ChM) e parte di controllo (il DCM); si descrive in dettaglio il generico ChM, soffermandosi soprattutto sul Formatter che, in particolare, consente di rendere variabile la profondità di conteggio; si descrive il funzionamento del DCM e, infine, quello delle sue macchine a stati.

Nel Capitolo 4 si elencano i principali compiti svolti dal DSP; si analizza la gestione delle comunicazioni sia verso l'FPGA, sia verso gli altri dispositivi presenti sulla scheda; si descrive la tecnica di Ping-Pong buffering, dimostrando analiticamente i vantaggi che introduce; si ricavano le relazioni necessarie per stimare le prestazioni del sistema; si descrive l'algoritmo di elaborazione real-time, e si analizza il diagramma di flusso.

Nel Capitolo 5 si descrivono le principali scelte effettuate per ottimizzare la sintesi e l'implementazione dell'architettura dell'FPGA; si descrive la tecnica di *unroll* utilizzata per ottimizzare i tempi di elaborazione del DSP e per verificare le ipotesi derivanti dall'analisi svolta nel Capitolo 2; si stimano le prestazioni del sistema utilizzando le relazioni determinate nel Capitolo 4 e, infine, se ne riportano i valori attuali ricavati da test sperimentali.

Ringraziamenti

Desidero ringraziare sentitamente il Prof. Roberto Saletti e l'Ing. Federico Baronti, per avermi dato l'opportunità di partecipare alla stesura di un articolo per la *10th Euromicro Conference on Digital System Design*, dal titolo "FPGA/DSP-based Configurable Multi-Channel Counter", nel quale sono stati pubblicati i risultati ottenuti da questa tesi.

Ringrazio inoltre i dottorandi del Laboratorio di Sistemi Elettronici del Dipartimento di Ingegneria dell'Informazione, che con pazienza ed amicizia mi hanno aiutato a risolvere i dubbi e i problemi che molto spesso ho incontrato durante lo svolgimento della tesi.

Infine, desidero ringraziare un grandissimo amico e maestro, Mario, che durante tutto il mio percorso universitario, e specialmente nei momenti più difficili e importanti, mi ha sempre incoraggiato e sostenuto con la sua intelligenza, il suo affetto, e la sua infinita disponibilità.

Capitolo 1. Contatori multicanale

1.1 Counter array: descrizione e funzionalità

Un *counter array* è un sistema elettronico digitale ad N_C canali di ingresso, capace di contare il numero di impulsi rilevati su ciascun canale in intervalli di tempo di durata T_W. Tali intervalli prendono il nome di *finestre di integrazione*. Allo scadere di ogni finestra, il counter array fornisce in uscita gli N_C risultati di conteggio relativi alla finestra appena conclusa, e mantiene tale informazione fino allo scadere della finestra successiva. Il numero di bit L utilizzati per rappresentare ciascun risultato prende il nome di *profondità di conteggio*, mentre il metodo secondo cui le varie finestre di integrazione si susseguono nel tempo prende il nome di *modalità di acquisizione*.

A seconda del tipo di implementazione, alcuni parametri del counter array sono fissi (come sempre avviene per N_C), mentre altri possono essere impostati di volta in volta dall'esterno. Quando tutti i parametri sono determinati, è possibile controllare l'avvio e l'arresto delle attività dell'array tramite opportuni comandi di *start* e *stop*.

Riguardo alle possibili modalità di acquisizione, le più frequenti sono due: *continua* o *triggerata*. In modalità continua, dopo aver ricevuto il comando di start, il counter array comincia subito a contare. Le finestre di integrazione si susseguono ininterrottamente, fino all'arrivo del comando di stop. Viceversa, in modalità triggerata, dall'esterno vengono impostati due ulteriori parametri: un ritardo D ed un numero N_W di finestre di integrazione. Dopo aver ricevuto il comando di start, il counter array si pone in attesa di un opportuno segnale di trigger proveniente dall'esterno. All'arrivo del trigger, l'array lascia passare il ritardo D prefissato, trascorso il quale comincia a contare. Dopo le N_W finestre di integrazione, il conteggio si blocca e l'array torna nuovamente in attesa di trigger. Il procedimento si ripete fino all'arrivo del comando di stop. Le due modalità appena descritte sono rappresentate in Fig. 1.1.

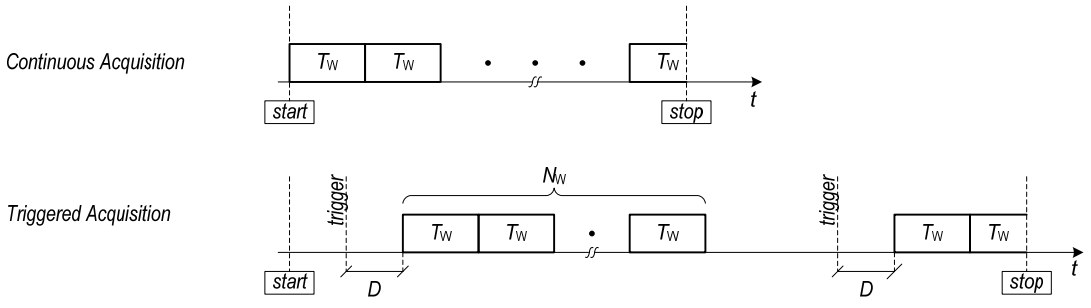


Fig. 1.1 Modalità di acquisizione di un counter array

1.2 Sistemi di misura e contatori multicanale

In settori scientifici quali astronomia, chimica, fisica, geologia, biomedicina e ingegneria dei processi industriali, i counter array svolgono un ruolo di fondamentale importanza all'interno di complessi sistemi di misura. In particolare, tali sistemi sono costituiti da un insieme di N_C sensori, da un counter array, da un'interfaccia di comunicazione, da un PC e, i più evoluti, anche da una unità dedicata di real-time processing (Fig. 1.2).

I sensori utilizzati sono in genere di due tipi: alcuni traducono la grandezza fisica osservata (ad esempio pressione, temperatura) in un segnale digitale di opportuna frequenza, altri segnalano il manifestarsi di un certo fenomeno fisico (ad esempio l'impatto di una particella su una determinata superficie di rilevamento) con un singolo impulso digitale di durata fissata. Collegando le uscite dei sensori ad un counter array, il sistema è in grado di conoscere il numero di fronti inviati da ciascun sensore su intervalli temporali di T_W secondi. A partire da tali risultati, tramite opportune elaborazioni è possibile risalire ad altre misure a vario titolo richieste.

I risultati forniti dal counter array sono quindi inviati sia all'interfaccia di comunicazione, sia all'unità di real-time processing. A sua volta, l'unità di real-time processing invia i dati processati alla stessa interfaccia e, contemporaneamente, svolge funzioni di controllo sia sui sensori, sia sui processi di condizionamento delle grandezze fisiche da essi osservate.

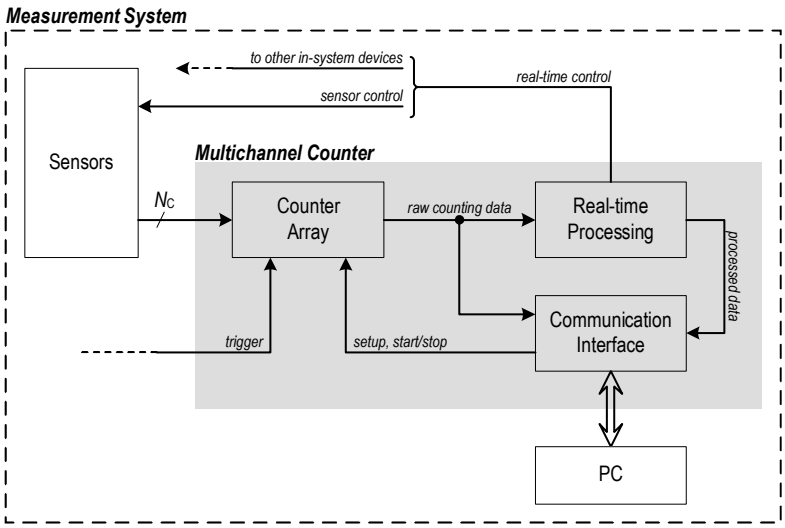


Fig. 1.2 Architettura di un sistema di misura basato su contatore multicanale

Tramite l'interfaccia di comunicazione, il PC è dunque in grado di ricevere i dati provenienti sia dal counter array, sia dall'unità di processing, e di gestirli secondo le più diverse modalità. Ad esempio, può ulteriormente elaborarli in real-time, può salvarli su opportuni supporti di memorizzazione (come ad esempio un hard-disk) ed elaborarli off-line, oppure può semplicemente utilizzarli per monitorare l'andamento nel tempo delle grandezze osservate e lo stato di funzionamento del sistema di misura. In particolare, le operazioni di

monitoraggio sono solitamente svolte a partire dai dati processati, sia perché, in questo modo, il PC deve solo visualizzare i dati senza alcuna elaborazione aggiuntiva, sia perché il data-rate proveniente dall'unità di processing è solitamente molto minore di quello proveniente dal counter array, e quindi di più facile gestione.

L'elaborazione svolta dall'unità di real-time processing dipende dunque non solo dalla misura che si vuole effettuare, ma anche dalle eventuali esigenze di controllo e monitoraggio.

Un esempio di elaborazione svolta da tale unità è l'accumulo, per ciascun canale, dei risultati forniti dal counter array in un numero prefissato N_{ACC} di finestre consecutive. In questo modo, è come se l'unità di processing fornisse una nuova integrazione su finestre ampie $N_{\mathrm{ACC}} \cdot T_{\mathrm{W}}$ secondi. Un'informazione di questo tipo può essere utilizzata sia per operazioni di monitoraggio che di controllo. In particolare, è possibile controllare se il pulse-rate medio di ciascun sensore supera o meno una certa soglia di sicurezza stabilita dal costruttore e, in caso affermativo, attuare opportuni interventi di protezione. Ad esempio, è possibile limitare le cause di tale problema intervenendo sui dispositivi di condizionamento delle grandezze fisiche in ingresso, oppure è possibile disattivare temporaneamente i vari sensori agendo direttamente sui relativi circuiti di alimentazione.

Si osservi inoltre che tramite l'interfaccia di comunicazione, il PC è in grado di impostare i parametri del counter array e di controllare la sua attività inviando i comandi di start e stop.

Infine, dato che nei sistemi di misura più avanzati il counter array è sempre affiancato da un'interfaccia di comunicazione e da una unità di real-time processing, con il termine *contatore multicanale* si è soliti indicare l'insieme dei tre dispositivi, così come è evidenziato dall'area ombreggiata di Fig. 1.2.

1.3 Contatori multicanale presenti sul mercato

Per quanto descritto nei paragrafi precedenti, i principali elementi di valutazione di un contatore multicanale sono:

- il numero di canali N_{C};
- il massimo pulse-rate f_{INmax} di ingresso;
- il range e lo step di variazione di T_{W};
- il range ed i possibili valori di L;
- le possibili modalità di acquisizione;
- le eventuali elaborazioni real-time.

Utilizzando un tale set di parametri, è interessante poter confrontare le prestazioni fornite dai migliori contatori multicanale presenti attualmente sul mercato, qui di seguito elencati:

- PicoQuant *PicoHarp 300*, [1];
- Ortec *MCS-PCI*, [2];
- Addi-Data *APCI-1710*, [3];
- KineticSystems *P635/V635*, [4];
- SeaBird *SBE-31*, [5].

I primi tre sono appositamente progettati per essere inseriti in sistemi dedicati al conteggio di particelle e alla misura di intervalli temporali ad alta risoluzione, mentre gli altri due sono progettati per essere inseriti in frequenzimetri ad alta precisione.

Dato che ciascun contatore è progettato per una specifica applicazione, il set di parametri fornito dai costruttori per caratterizzare tali dispositivi è spesso diverso da caso a caso. Al fine di fornire un quadro generale coerente con l'architettura descritta in Fig. 1.2 e tale da consentire l'analisi comparativa dei vari contatori presi in considerazione, si è dunque cercato di rendere i vari set di parametri uniformi a quello scelto per il confronto, ricavando – tramite opportuni procedimenti analitici – quei particolari parametri che, a seconda dei casi, non sono direttamente specificati dai costruttori.

Per quanto riguarda N_{C}, i contatori presi in esame presentano rispettivamente 2, 1, 12, 8 e 31 canali.

Per quanto riguarda f_{INmax}, i contatori in questione riescono a gestire un massimo pulse-rate di ingresso pari rispettivamente a 10 MHz, 150 MHz, 5 MHz, 100 kHz e 98.3 kHz.

Riguardo invece al range e allo step di variazione di T_{W}, dato che è difficile e costoso fare il primo estremamente ampio e, allo stesso tempo, il secondo estremamente piccolo, a seconda della destinazione commerciale del contatore in questione si preferisce ottimizzare l'uno a scapito dell'altro. Nei sistemi dedicati al conteggio di particelle si cerca infatti di avere step di variazione molto piccoli, mentre in quelli dedicati al calcolo di frequenze si cerca di avere range molto ampi.

Ad esempio, PicoHarp 300 fornisce uno step di qualche picosecondo all'interno di un range che si estende per 2 ordini di grandezza, da qualche centinaia di nanosecondi a qualche decina di microsecondi. Viceversa P635 e V635, con uno step di un millisecondo, riescono a scansionare un range che si estende per 6 ordini di grandezza, dal millisecondo alle migliaia di secondi.

Il massimo valore T_{Wmax} di T_{W} è legato alla massima profondità di conteggio L_{C} e a f_{INmax} dalla seguente relazione:

$$L_{\mathrm{C}} = \left\lceil \log_2 \left(T_{\mathrm{Wmax}} \cdot f_{\mathrm{INmax}} \right) \right\rceil. \tag{1.1}$$

Tale relazione comporta importanti scelte architetturali in fase di progetto. Infatti, una volta stabilito f_{INmax}, quanto più grande è T_{Wmax}, tanto più grande sarà L_{C} e, di conseguenza, tanto maggiori saranno le risorse logiche necessarie per gestire ogni singolo canale dell'array.

In generale, quando L è impostabile in funzione delle condizioni di utilizzo, tutte le scelte di L verificanti $L \leq L_{\mathrm{C}}$ sono a priori disponibili. Per i contatori presi in considerazione, invece, il valore di L è fisso, e si ha quindi $L = L_{\mathrm{C}}$, con L_{C} pari rispettivamente a 9, 30, 32, 24 e 12 bit.

Man mano che T_{W} diminuisce, il rate con cui il counter array fornisce i propri risultati di conteggio aumenta. Di conseguenza, osservando l'architettura di Fig. 1.2, il minimo valore T_{Wmin} di T_{W} dipende

- in generale, dalla banda di comunicazione disponibile tra array e dispositivi a valle, e dal massimo bit-rate sostenibile in ingresso dal più "lento" di tali dispositivi, che, come spesso accade, è l'unità di real-time processing;
- in particolare, dal modo con cui la banda di comunicazione è utilizzata e dalla complessità delle particolari elaborazioni real-time richieste.

Inoltre, come si vedrà in seguito (Paragrafi 1.5 e 2.1), nel caso in cui L sia impostabile in funzione delle condizioni di utilizzo, T_{Wmin} dipende anche da L.

Di conseguenza, in fase di progetto, per raggiungere un determinato T_{Wmin} si può intervenire non solo scegliendo opportuni dispositivi, tecnologie ed algoritmi real-time che consentano velocità maggiori, ma anche prevedendo – a differenza di quanto viene fatto nei contatori presi in esame – la possibilità che L figuri tra i parametri settabili dall'utente.

A seconda della particolare applicazione cui sono destinati, i contatori in esame mettono a disposizione dell'utente un gran quantitativo di modalità di acquisizione, comunque riconducibili a quella continua e a quella triggerata. Osserviamo che non tutti sono in grado di garantire finestre di integrazione perfettamente consecutive. PicoHarp 300, infatti, introduce un seppur breve tempo morto (*dead-time*) tra una finestra e la successiva, ineliminabile e di durata inferiore al centinaio di nanosecondi. Per ciascun contatore è inoltre possibile selezionare la sorgente di trigger tra un determinato insieme di segnali, sia interni che esterni. Ad esempio, lo stesso PicoHarp 300 mette a disposizione quattro ingressi dedicati a segnali di trigger (o *marker*) provenienti dall'esterno, ciascuno dei quali gioca un ruolo diverso a seconda della modalità di acquisizione prescelta.

Riguardo alle elaborazioni real-time, PicoHarp 300, MCS-PCI e SBE-31 offrono funzioni di accumulo (*histogram*) e di media temporale. Tali contatori sono anche in grado di generare opportuni segnali sincronizzati al processo di acquisizione, con cui poter controllare i vari sensori collegati in ingresso. In partico-

lare, MCS-PCI consente di generare una forma d'onda a dente di sega o triangolare, il cui valore di picco è impostabile dall'utente.

Per ciascun contatore in esame, il costruttore fornisce appositi software di corredo da utilizzare sul PC collegato. Tali software non solo consentono operazioni di monitoraggio, controllo e archiviazione, ma sono anche in grado di effettuare un grande quantitativo di elaborazioni real-time, solitamente molto complesse. In generale, tali elaborazioni sono strettamente legate alle applicazioni per le quali il contatore è progettato. Infine, nella maggior parte dei casi, i costruttori forniscono il supporto (driver e librerie) necessario all'utente per poter sviluppare autonomamente ulteriori applicazioni real-time, in ambienti quali Visual C++ [6] e LabView [7].

1.4 Le richieste del settore astronomico: specifiche dei contatori multicanale per Fast Transient Imaging (FTI) e Adaptive Optics (AO)

In astronomia, esistono due importanti applicazioni svolte dai più avanzati osservatori terrestri: *Fast Transient Imaging* (FTI) e *Adaptive Optics* (AO).

L'applicazione FTI consiste essenzialmente nell'acquisizione ad elevatissimi frame-rate dell'immagine fornita dal telescopio. In questo modo, analizzando ogni singolo fotogramma, è possibile osservare in dettaglio quei fenomeni estremamente veloci (*high transient phenomena*) quali ad esempio le emissioni delle pulsar e l'esplosione delle supernovae. Tipicamente, il frame-rate è selezionabile in base al tipo di fenomeno che si vuole osservare.

L'applicazione AO consiste invece nella correzione in tempo reale delle distorsioni ottiche introdotte dall'atmosfera terrestre. Tali distorsioni, infatti, variano rapidamente nel tempo ed in modo aleatorio, rendendo sensibilmente sfocate le immagini visualizzate dal telescopio. A partire dai fotogrammi acquisiti secondo le tecniche di FTI, per ciascun pixel di ogni fotogramma viene calcolato in tempo reale un apposito fattore di correzione, o *segnale di curvatura*. L'insieme dei vari segnali di curvatura viene quindi inviato ad uno speciale specchio deformabile, inserito direttamente nel cammino ottico del telescopio. Tale specchio, infatti, modificando opportunamente la sua forma geometrica, riesce a compensare efficacemente le distorsioni introdotte dall'atmosfera. Anche se l'algoritmo solitamente usato per il calcolo dei segnali di curvatura è relativamente semplice, la teoria sulla quale si fonda l'ottica adattiva è molto complessa, e lo stesso si può dire per la tecnologia che sta alla base degli specchi deformabili.

Per poter determinare correttamente i segnali di curvatura è necessario che l'oggetto puntato dal telescopio sia sufficientemente luminoso o, in alternativa, che ci sia un altro oggetto luminoso (*guide star*) adeguatamente vicino a quello che si vuole osservare. Quest'ultimo oggetto può essere una vera e propria stella – e in tal caso si parla di *natural guide star* (NGS) – oppure una stella

"simulata" utilizzando impulsi laser periodici di opportuna potenza – e in tal caso si parla di *laser guide star* (LGS).

Le applicazioni AO che sfruttano le NGS (AO-NGS) sono identiche a quelle tradizionali, quelle cioè in cui l'oggetto puntato è – da solo – sufficientemente luminoso. Viceversa le applicazioni AO basate su LGS (AO-LGS) necessitano di una diversa modalità di acquisizione dei fotogrammi. Infatti, per ottenere buoni risultati, il cannone che spara gli impulsi laser nel cielo è solitamente posizionato in stretta vicinanza del telescopio. In seguito all'invio di un impulso, la potenza del laser è talmente elevata che i sensori presenti nel telescopio restano "abbagliati" per un certo *tempo di decadimento*, durante il quale non ha senso acquisire fotogrammi. Trascorso il tempo di decadimento, l'acquisizione deve subito cominciare, e può proseguire fino ad un istante prima dell'invio dell'impulso successivo.

Attualmente, esempi concreti di tali applicazioni sono svolte dai telescopi di ESO [8] nell'ambito dei progetti ULTRACAM (FTI) [9], MACAO (AO-NGS) [10] e SINFONI (AO-LGS) [11].

Nelle applicazioni FTI ed AO, l'acquisizione dei fotogrammi richiede ovviamente l'utilizzo di sensori a matrice di pixel. Attualmente, i più utilizzati sono i CCD. Tuttavia, in seguito ad alcuni importanti risultati ottenuti negli ultimi anni, la ricerca si sta spingendo verso l'utilizzo dei nuovi sensori SPADA (Single Photon Avalanche Diode Array) [12]-[15]. I sensori SPADA, infatti, presentano numerosi vantaggi rispetto ai CCD. Tra questi, i più importanti sono la maggiore velocità, semplicità e affidabilità, il minor costo e – soprattutto – la natura intrinsecamente digitale. Infatti, ciascuno SPAD che compone l'array, grazie ad opportuni circuiti di polarizzazione e di controllo, è in grado di inviare un impulso digitale per ciascun fotone rilevato, risolvendo definitivamente i problemi di *read-out noise* [12]. Ai terminali, un sensore SPADA a N_C pixel si presenta dunque come una sorgente di impulsi digitali ad N_C canali.

Volendo impiegare i sensori SPADA nelle applicazioni FTI ed AO, il metodo da seguire per poter acquisire le immagini fornite dal telescopio è quello di contare i fotoni rilevati da ciascun pixel all'interno di una determinata finestra di integrazione e, successivamente, di mappare ogni risultato di conteggio su un'opportuna scala di luminanza (ad esempio la scala dei grigi).

Risulta dunque immediata la necessità di includere nei sistemi di misura dedicati a FTI ed AO un contatore multicanale basato sull'architettura di Fig. 1.2. Per potersi interfacciare perfettamente agli altri dispositivi già presenti, tale contatore dovrà soddisfare alcune importanti specifiche. Considerando ad esempio i sistemi [9], [10] e [11], tali specifiche sono riassunte in Tab. 1.1.

	FTI	AO-NGS	AO-LGS
N_C	60		
f_{INmax}	20 MHz		
T_W	10 µs ÷ 100 ms, 10 µs linear steps		10 µs
L	16 bit (with counters saturation)		
Acquisition Mode	Continuous with no dead-time		Triggered with no dead-time, D = 2.5 µs, N_W = 14
Real-Time Elaboration	Accumulation	Sinusoidal Waveform Generation Curvature Signals Computation	
PC Tasks	Setup Parameters and Control Commands Upload, System Monitoring		
	Direct-to-disk counting data recording		
	Incoming Image Preview	Incoming Curvature Signals Preview	

Tab. 1.1 Specifiche richieste al contatore multicanale per applicazioni di FTI, AO-NGS e AO-LGS

Il numero di canali richiesto è pari a 60, ed è imposto dai particolari dispositivi ottici presenti nei telescopi di ESO. Inoltre, il massimo pulse-rate di ingresso richiesto è pari a 20 MHz, ossia al più grande valore attualmente sostenibile dai sensori SPADA presenti in letteratura [12].

Riguardo al valore di L, si osservi che con $T_{\mathrm{Wmax}} = 100$ ms e $f_{\mathrm{INmax}} = 20$ MHz, applicando la (1.1) si ottiene $L_{\mathrm{C}} = 21$ bit. Tuttavia, le specifiche richiedono contatori a $L = 16$ bit con saturazione. Questo significa che nel caso in cui le condizioni di funzionamento siano tali da causare l'overflow di alcuni contatori dell'array, il risultato fornito da tali contatori deve essere congelato a $\mathrm{FFFF_h}$ e l'utente deve essere avvisato dell'avvenuta saturazione. Una tale scelta di L è giustificata dal fatto che, nelle tipiche condizioni di funzionamento, è raro ottenere risultati che necessitino più di 16 bit. Di conseguenza, forzando $L = 16$ e implementando i meccanismi di saturazione, si risparmiano numerose risorse logiche.

Si osservi inoltre che per ciascuna delle tre applicazioni è richiesto l'utilizzo di una unità di real-time processing.

Per FTI, tale unità deve svolgere una semplice funzione di accumulo, come quella descritta nel Paragrafo 1.2. Accumulando i risultati relativi a N finestre consecutive di durata T_{W}, è come se l'unità di elaborazione fornisse in uscita una sequenza di fotogrammi molto più lenta di quella fornita dall'array. In altre parole, è come avere un fotogramma ogni $T_{\mathrm{ACC}} = N \cdot T_{\mathrm{W}}$ secondi anziché un fotogramma ogni T_{W} secondi. Mentre i fotogrammi provenienti dall'array sono inviati al PC per essere direttamente archiviati su hard-disk, quelli provenienti dall'unità di elaborazione sono inviati al PC per il solo scopo di monitoraggio. In quest'ultimo caso, il PC non deve far altro che visualizzare tali fotogrammi, senza alcuna operazione aggiuntiva. Ovviamente, scegliendo N in modo opportuno è possibile regolare il frame-rate di visualizzazione.

Per entrambe le applicazioni AO, l'unità di elaborazione non solo deve calcolare i segnali di curvatura, ma deve anche generare un'opportuna sequenza di campioni a 16 bit relativi ad una determinata forma d'onda sinusoidale. Tali campioni saranno inviati ad un apposito convertitore D/A e successivamente ad uno stadio di amplificazione. La sinusoide ricostruita, insieme agli stessi segnali di curvatura, sarà quindi inviata ai dispositivi dedicati al controllo degli specchi deformabili. Riguardo alla generazione della sequenza dei campioni, l'utente può specificare i seguenti parametri:

- ampiezza massima;
- frequenza $1.5 \div 3$ kHz con step di 1 Hz.

Affinché l'algoritmo per il calcolo dei segnali di curvatura funzioni correttamente, l'utente deve specificare non solo tutti i parametri relativi alla sinusoide, ma deve anche scegliere un numero di *periodi di osservazione*, tra 1 e 256. Tale algoritmo prevede, per ciascun canale, due differenti operazioni di accumulo. La prima consiste nell'accumulare gli impulsi ricevuti durante tutti i semiperiodi *positivi* di osservazione. La seconda consiste invece nell'accumulare gli impulsi ricevuti durante tutti i semiperiodi *negativi* di osservazione. Relativamente al canale i-esimo, al termine del numero prefissato di periodi di osservazione, si considera il risultato A_i del primo accumulo, e il risultato B_i del secondo. Il segnale di curvatura C_i relativo al canale i-esimo è dato da:

$$C_i = \frac{A_i - B_i}{A_i + B_i} \tag{1.2}$$

Di conseguenza, trascorso il numero prefissato di periodi di osservazione, l'unità di real-time processing calcola gli N_{C} segnali C_i, gli invia al PC a scopo di monitoraggio, e ricomincia da capo le operazioni di accumulo.

Si osservi infine che l'algoritmo appena descritto è valido sia per AO-NGS che per AO-LGS. Tuttavia, nel caso di AO-LGS, tale algoritmo non è eseguito continuamente, ma solo durante l'attività del counter array. Quest'ultima è infatti regolata da una precisa modalità di acquisizione triggerata i cui parametri D ed N_{W} sono riportati in Tab. 1.1.

1.5 Realizzazione del primo contatore multicanale per FTI

Dall'analisi fin qui svolta, risulta evidente che nessuno dei migliori contatori multicanale presenti sul mercato è in grado di soddisfare appieno tutte le specifiche richieste dalle applicazioni FTI ed AO.

Di conseguenza, nell'ambito del progetto di ricerca COFIN-PRIN "Development of Monolithic Photon-Counter Arrays for Transient High-Energy Phenomena and Adaptive Optics in Astrophysics" avviato nel 2003, il Dipartimento di Ingegneria dell'Informazione dell'Università di Pisa è stato incaricato di progettare un apposito contatore multicanale che soddisfi appieno le specifiche riportate in Tab. 1.1.

Inoltre, al fine di rendere tale contatore immediatamente utilizzabile nell'ambito di [9], [10] e [11], in cui i sistemi di misura sono dei veri e propri sistemi distribuiti, si è pensato di far svolgere al PC di Fig. 1.2, oltre alle funzioni richieste dalle specifiche, anche quella di server TCP/IP. In questo modo, un qualunque altro PC client si può facilmente collegare al PC server e, dopo aver ottenuto il controllo esclusivo del contatore multicanale, può impostare i parametri di funzionamento, controllare lo stato e, una volta terminato il processo di acquisizione, scaricare i dati archiviati. L'architettura definitiva è dunque quella riportata in Fig. 1.3.

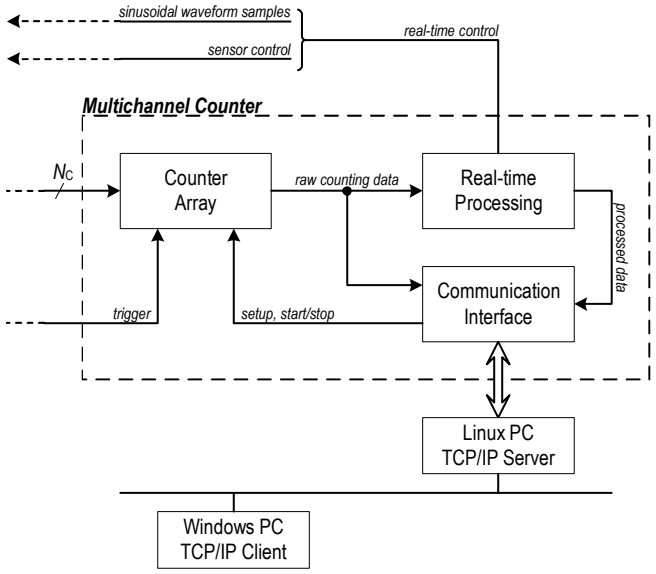


Fig. 1.3 Architettura del contatore multicanale sviluppato dal Dipartimento di Ingegneria dell'Informazione dell'Università di Pisa

In particolare, per l'implementazione dell'applicazione server è stato scelto il sistema operativo Linux, dato che offre la possibilità di utilizzare una grande quantità di risorse di programmazione open-source, facilmente reperibili in rete. Per l'implementazione dell'applicazione client, invece, dovendo realizzare complesse interfacce grafiche di controllo e monitoraggio, si è scelto di utilizzare l'ambiente LabView su sistema operativo Windows.

Per contenere il più possibile i tempi e i costi di progetto, si è pensato di realizzare il contatore multicanale utilizzando una delle molte schede di sviluppo disponibili in commercio. Una tale strategia, infatti, comporta costi decisamente minori rispetto ad una soluzione ASIC che, tuttavia, avrebbe sicuramente consentito di raggiungere prestazioni globalmente migliori. Avendo la necessità di disporre di molte risorse logiche e, al tempo stesso, di una veloce unità di elaborazione, la scelta si è indirizzata verso le schede basate su FPGA/DSP e, in particolar modo, sulla Orsys MicroLine C6713Compact [16]. In Fig. 1.4 è rappresentata la parte più significativa dell'architettura di tale scheda, quella cioè che coinvolge i principali componenti.

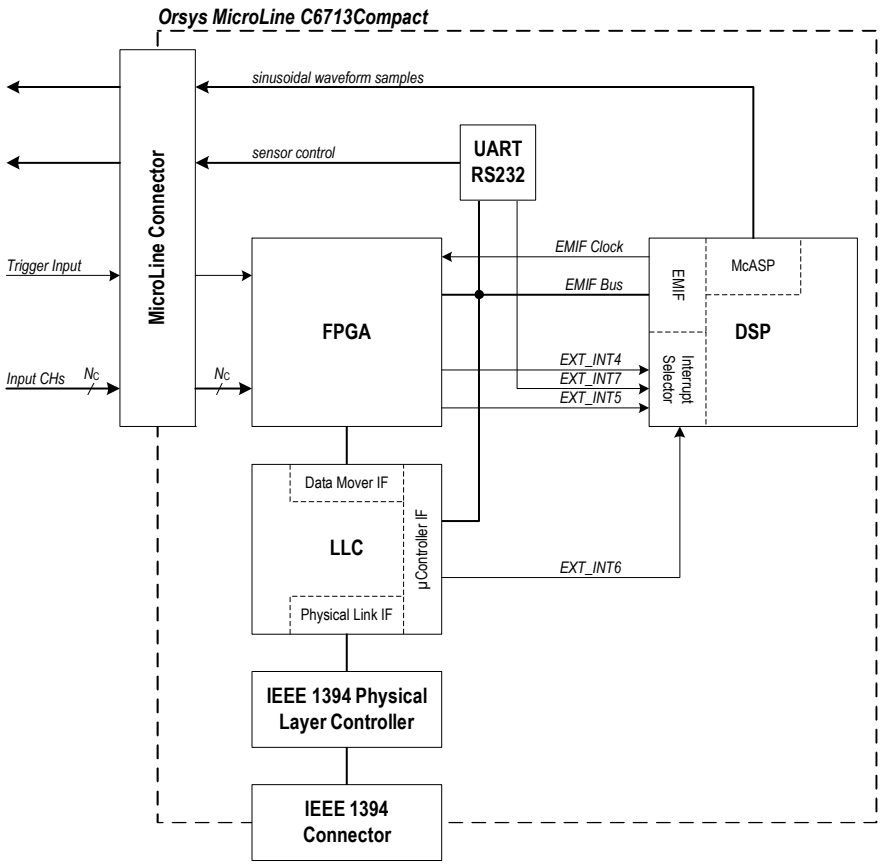


Fig. 1.4 Parte dell'architettura della scheda Orsys MicroLine C6713Compact relativa ai suoi principali componenti

Tali componenti sono:

- una FPGA Xilinx Virtex-II XC2V1000 da 1 M gate [17];
- un DSP Texas Instruments TMS320C6713 a 32 bit in virgola mobile e 225 MHz di core clock [18];
- un General Purpose Link-Layer Controller (LLC) Texas Instruments TSB12LV32 [18];
- una interfaccia seriale IEEE 1394a a 400 Mbps [19];
- una interfaccia UART RS232 Texas Instruments TL16C550 [18].

Il DSP è connesso all'FPGA per mezzo del bus EMIF (External Memory InterFace). La stessa EMIF fornisce anche il segnale di clock a 90 MHz.

L'LLC gestisce le comunicazioni sul bus IEEE 1394 per mezzo della DataMover InterFace e della µController InterFace. Su tale bus è possibile condurre simultaneamente due tipi di transazioni: *isocrone* e *asincrone*. Le transazioni isocrone sono appositamente pensate per tutti quei casi in cui, tra due o più nodi, è necessario sostenere un flusso dati rigorosamente costante nel tempo, anche per lunghi periodi. Viceversa, le transazioni asincrone sono pensate per i casi in cui non si hanno particolari esigenze di velocità o di temporizzazione, ed offrono il vantaggio di essere molto più facili da gestire rispetto alle precedenti, sia ad alto che a basso livello. In particolare, le transazioni isocrone

sono portate a termine dalla DataMover IF, le asincrone dalla µController IF. Si osservi inoltre che la prima è direttamente connessa all'FPGA, mentre la seconda è connessa al DSP tramite il bus EMIF.

Dal momento che le possibili frequenze della sinusoide richiesta dalle applicazioni AO sono tutte comprese all'interno della banda audio, per generare la sequenza dei campioni si è scelto di utilizzare una delle due periferiche Multichannel Audio Serial Port (McASP) di cui il DSP dispone internamente.

Infine, per il controllo dei sensori si è previsto l'utilizzo dell'interfaccia UART RS232, già presente sulla scheda e connessa al DSP tramite il bus EMIF.

Il progetto del contatore multicanale è quindi consistito nell'implementazione su FPGA del counter array e nella programmazione del DSP per le elaborazioni real-time e per la gestione delle varie periferiche.

Tramite il connettore MicroLine, gli N_C segnali provenienti dal sensore sono mandati in ingresso all'FPGA, che provvede ad acquisire e a contare gli impulsi ricevuti. Allo scadere di ogni finestra di integrazione, l'FPGA è dunque in grado di fornire in uscita un *pacchetto* di risultati di conteggio relativi agli N_C ingressi. Tale pacchetto è inviato sia al DSP sia alla DataMover IF. In questo modo, mentre l'LLC trasmette i risultati di conteggio al server Linux per mezzo del link isocrono, il DSP è in grado di elaborarli in real-time. Dopo aver elaborato un certo numero di pacchetti dovuto alla particolare applicazione svolta, il DSP invia i dati processati alla µController IF, e l'LLC a sua volta li trasmette al server Linux per mezzo del link asincrono.

Inoltre, per mezzo del link asincrono, il server è in grado di inviare al DSP tutti i parametri ed i comandi relativi all'acquisizione, al controllo dei sensori e alla generazione della sinusoide. È infatti il DSP che, tramite il bus EMIF, provvede a impostare correttamente l'FPGA, ad avviare o bloccare il processo di conteggio, e a controllare i sensori attraverso l'interfaccia UART.

Il parallelismo con l'architettura di Fig. 1.3 risulta adesso evidente.

Riguardo allo sviluppo del progetto nel tempo, in una prima fase sono stati raggiunti i seguenti obiettivi:

- $N_C = 60$;
- $f_{INmax} = 45$ MHz;
- $T_W = 20$ µs $\div$ 163 ms con step lineari di 10 µs;
- $L = 16$;
- modalità di acquisizione continua;
- accumulo real-time;
- gestione periferica UART RS232;
- sviluppo applicazione server Linux;

- sviluppo applicazione client LabView.

Inoltre, sono state separatamente sviluppate e testate le funzioni relative alla generazione dei campioni della sinusoide e alla gestione della periferica McASP.

Successivamente, grazie ad ulteriori sviluppi riguardanti sia il design dell'FPGA sia il firmware del DSP, è stato raggiunto per T_{Wmin} il valore richiesto di 10 µs. La principale intuizione che ha portato ad un tale miglioramento è stata quella di provare a far impostare automaticamente al sistema $L = 8$ ogni volta che, dall'esterno, l'utente selezionava $T_{\mathrm{W}} = 10$ µs. Tali risultati, recentemente pubblicati in [20], hanno quindi consentito di soddisfare appieno le specifiche FTI. Si osservi infatti che, per $T_{\mathrm{W}} = 10$ µs, se il pulse-rate di ingresso dovuto al sensore SPADA non supera i 20 MHz, ciascun risultato di conteggio non supera certamente il valore di 200, e quindi è tranquillamente esprimibile utilizzando solo 8 bit.

1.6 Il superamento delle specifiche astronomiche: obiettivi prefissati e risultati ottenuti nel progetto di un nuovo contatore multicanale

Lo scopo di questa tesi è stato quello di utilizzare l'esperienza acquisita dal Dipartimento di Ingegneria dell'Informazione nell'ambito del progetto COFIN-PRIN appena descritto, al fine di ottenere un contatore multicanale configurabile, ad alte prestazioni, che superi le specifiche richieste dall'applicazione astronomica.

Gli obiettivi da raggiungere sono stati pertanto così prefissati:

- sfruttare al meglio le risorse disponibili sulla scheda in modo da massimizzare N_{C}, massimizzare il range di T_{W} e minimizzare il suo step di variazione;
- implementare la modalità di acquisizione triggerata, con D ed N_{W} impostabili dall'utente;
- modificare sia l'applicazione server che l'applicazione client in modo da consentire all'utente di poter salvare su hard-disk solo i risultati relativi ad un certo insieme di canali definibile a piacere.

Inoltre, dagli ultimi risultati ottenuti nel progetto precedente, è apparso evidente che, per ottimizzare le prestazioni, è importante rendere il parametro L variabile. Una tecnica di questo tipo, come visto nel Paragrafo 1.3, non è stata ancora utilizzata in nessuno dei migliori contatori multicanale presenti attualmente sul mercato.

Per tale motivo, è stato dunque necessario:

- studiare in modo del tutto generale la dipendenza di T_{Wmin} da L, determinando le ipotesi che un contatore multicanale basato sull'architettura

di Fig. 1.4 deve verificare affinché una riduzione di L porti effettivamente ad una riduzione di T_{Wmin};

- fare in modo che il nostro sistema soddisfacesse tali ipotesi;
- utilizzare le informazioni ottenute dallo studio per ottimizzare T_{Wmin}.

In particolare, lo studio ha evidenziato che, riducendo L, le ipotesi che il sistema deve soddisfare per ottenere una conseguente riduzione di T_{Wmin} sono esclusivamente legate al massimo bit-rate di ingresso sostenibile dal DSP durante le elaborazioni real-time.

Si è deciso dunque di fare $L = 23, 16, 8$ e di consentire all'utente di scegliere liberamente uno dei tre valori. Per raggiungere questo obiettivo e per soddisfare al tempo stesso le specifiche sullo step di T_{W} e sulla modalità di acquisizione triggerata, l'FPGA è stata completamente ridisegnata. In particolare, l'architettura è stata organizzata in modo tale da rendere N_{C} l'unico parametro di sintesi. Di conseguenza, utilizzando l'ambiente di sviluppo Xilinx ISE Foundation [17], dopo un opportuno processo di ottimizzazione è stato possibile raggiungere $N_{\mathrm{C}} = 64$, occupando il 97 % delle risorse disponibili.

Il firmware del DSP è stato conseguentemente ottimizzato in modo da controllare le nuove funzioni disponibili sull'FPGA e, soprattutto, in modo da verificare, per $N_{\mathrm{C}} = 64$, le ipotesi sul bit-rate derivanti dal precedente studio.

Infine, le applicazioni server e client sono state opportunamente modificate, sia per soddisfare la specifica sulla selezione dei canali, sia per consentire all'utente di impostare i nuovi parametri messi a disposizione dal sistema.

L'elenco seguente riporta in dettaglio i principali obiettivi raggiunti, confrontando i risultati ottenuti con quelli relativi al precedente lavoro:

- il numero di canali N_{C} è stato aumentato da 60 a 64;
- i possibili valori di L, precedentemente limitati a 16, 8 e gestiti automaticamente, sono stati estesi a 23, 16, 8 e resi selezionabili a piacere;
- con $N_{\mathrm{C}} = 64$ e $L = 23$, le prove sperimentali hanno fornito un valore di T_{Wmin} pari a 24 µs;
- con $N_{\mathrm{C}} = 64$ e $L = 16, 8$ le prove sperimentali hanno fornito un valore di T_{Wmin} rispettivamente pari a 14 µs e 8 µs, contro i corrispondenti 20 µs e 10 µs precedentemente raggiunti con solo 60 canali;
- il valore di T_{Wmax} è stato aumentato da 163 ms a 186 ms;
- lo step di variazione di T_{W} è stato ridotto da 10 µs a 11 ns;
- è stata implementata la modalità di acquisizione triggerata con $D = 0 \div 728$ µs in step lineari di 11 ns, e $N_{\mathrm{W}} = 0 \div 255, 510, 1020$ a seconda del valore di L, e rispettivamente in step di 1, 2 o 4 unità (in particolare, impostando $N_{\mathrm{W}} = 0$ il sistema si comporta come se fosse $N_{\mathrm{W}} = \infty$).

Si osservi che i valori di T_{Wmin} appena riportati si riferiscono al funzionamento del contatore quando il DSP svolge le funzioni di accumulo in real-time. Nel caso in cui l'utente non sia interessato all'accumulo ma desideri semplicemente inviare i risultati di conteggio sul bus IEEE 1394 disattivando l'elaborazione real-time, per $L = 23,\ 16,\ 8$ si è ottenuto rispettivamente $T_{\mathrm{Wmin}} = 10.5$ µs, 5.4 µs, 2.7 µs.

Inoltre, per confrontare le prestazioni del nostro contatore con quelle offerte da PicoHarp 300 relativamente al solo conteggio di particelle, abbiamo sintetizzato il sistema con $N_{\mathrm{C}} = 2$. In tal caso, disattivando l'elaborazione real-time, per $L = 23,\ 16,\ 8$ si è ottenuto rispettivamente $T_{\mathrm{Wmin}} = 378$ ns, 189 ns, 88 ns, mentre PicoHarp 300, con l'unico $L = 9$, fornisce $T_{\mathrm{Wmin}} = 260$ ns.

Infine, mantenendo $N_{\mathrm{C}} = 2$ e attivando la funzione di accumulo, per $L = 23$, 16, 8 si è ottenuto rispettivamente $T_{\mathrm{Wmin}} = 1.4$ µs, 744 ns, 378 ns.

Si osservi che, fissati T_{Wmax} e f_{INmax}, per (1.1) L_{C} è determinato. Scegliendo $L = L_{\mathrm{C}}$, per ogni finestra di integrazione e per ogni pulse-rate di ingresso gestibili dal sistema, non si avrà mai la saturazione dei contatori dell'array. Viceversa, scegliendo un $L < L_{\mathrm{C}}$, esisteranno sicuramente delle possibili condizioni di funzionamento che causeranno la saturazione dei contatori. Nel nostro caso, essendo $f_{\mathrm{INmax}} = 45$ MHz e $T_{\mathrm{Wmax}} = 186$ ms, si ha $L_{\mathrm{C}} = 23$. Consentendo la scelta $L = L_{\mathrm{C}} = 23$, si consente all'utente di poter contare, senza alcun rischio di saturazione, gli impulsi provenienti dall'esterno al massimo pulse-rate, per la più lunga finestra di integrazione. Viceversa, nella versione precedente, tale possibilità non esisteva. Infatti, come osservato nel Paragrafo 1.4, essendo $f_{\mathrm{INmax}} = 45$ MHz e $T_{\mathrm{Wmax}} = 163$ ms, si aveva $L_{\mathrm{C}} = 21$, mentre invece L poteva assumere solo i valori 16, 8.

Capitolo 2. Il parametro L

2.1 Ottimizzazione di T_{Wmin}: formattazione dei risultati in funzione di L

Una volta stabilite le risorse hardware, per quanto visto precedentemente, è interessante capire se sia possibile ridurre T_{Wmin} variando L e, in tal caso, quali siano le tecniche da utilizzare.

Nel Paragrafo 1.3 si è osservato che, una volta determinata la piattaforma hardware, T_{Wmin} è limitato sia dal modo con cui la banda di comunicazione fisicamente disponibile tra counter array e unità di real-time processing è utilizzata, sia dal massimo bit-rate che tale unità è in grado di sostenere nello svolgere la particolare elaborazione richiesta.

Di conseguenza, il primo problema da risolvere è quello di ottimizzare l'utilizzo della banda di comunicazione sfruttando la possibilità di variare L.

Nel Paragrafo 1.5 è stata descritta la piattaforma hardware scelta per realizzare il contatore multicanale. In particolare, l'FPGA, il DSP e l'LLC sono connessi tra loro per mezzo del bus EMIF, che è largo 32 b. Inoltre, anche i dati che l'FPGA è in grado di scambiare direttamente con la DataMover IF dell'LLC sono espressi su 32 b. Dunque, ogni trasferimento tra un dispositivo e un altro coinvolge sempre parole a 32 b. Quanti più dati si riesce a includere in ciascuna parola, tanto migliore sarà l'utilizzo della banda.

Si supponga noto il valore di L_{C}, che, per la (1.1), è legato alle massime condizioni di utilizzo rappresentate da T_{Wmax} e f_{INmax}. Si supponga inoltre che tale valore soddisfi la relazione $16 < L_{\mathrm{C}} \leq 32$.

Focalizzando l'attenzione sui trasferimenti tra FPGA e DSP, in virtù di tale ipotesi l'utilizzo della banda migliora se, a seconda del valore prescelto di $L \leq L_{\mathrm{C}}$, è possibile formattare più di un risultato di conteggio in una singola parola a 32 b.

Infatti, se $16 < L \leq L_{\mathrm{C}}$, una parola a 32 b può contenere un solo risultato di conteggio, ossia l'unico formato utilizzabile è 1×32 b.

Invece, se $8 < L \leq 16$ o $L \leq 8$, una parola a 32 b può contenere rispettivamente 2 o 4 risultati di conteggio consecutivi, ossia è possibile utilizzare un formato 2×16 b o 4×8 b.

In particolare,

- per $16 < L \leq L_{\mathrm{C}}$ è disponibile il solo formato 1×32 b;

- per $8 < L \leq 16$ sono disponibili entrambi i formati 1×32 b e 2×16 b;
- per $L \leq 8$ sono disponibili i tre formati 1×32 b, 2×16 b e 4×8 b.

Dunque, a seconda del formato utilizzato, l'FPGA può comporre, per ciascun canale di ingresso, una parola a 32 b con 1, 2 o 4 risultati di conteggio consecutivi e inviare al DSP un pacchetto di N_{C} parole rispettivamente ogni T_{W}, $2\,T_{\mathrm{W}}$ o $4\,T_{\mathrm{W}}$ secondi.

In questo modo, a parità di massimo bit-rate sostenibile in ingresso dal DSP, per ciascun L è possibile scegliere tra i formati compatibili quello che consente di raggiungere valori di T_{W} più piccoli.

Tuttavia, anche il massimo bit-rate del DSP dipende dal formato utilizzato. Infatti, il tempo necessario al DSP per processare ciascun pacchetto di N_{C} parole a 32 b aumenta in base alla quantità di risultati che esso contiene.

Di conseguenza, non è detto che la formattazione appena descritta consenta veramente di raggiungere valori di T_{W} più piccoli, e dunque di ridurre T_{Wmin}. Anzi, è necessario capire come il bit-rate del DSP deve variare rispetto al formato prescelto in modo da garantire una qualche riduzione di T_{Wmin}.

A tal fine, per ciascuno dei tre formati, si indichino rispettivamente con B_{32}, B_{16} e B_8 i massimi bit-rate di ingresso che il DSP è in grado di sostenere in real-time. Per quanto appena osservato, risulta:

$$B_{32} > B_{16} > B_8\,. \tag{2.1}$$

Fissata la piattaforma hardware, i valori di B_{32}, B_{16} e B_8 sono noti non appena N_{C} e il particolare algoritmo real-time sono determinati.

Per sostenere l'elaborazione real-time, il DSP deve essere in grado di processare ciascun pacchetto ricevuto dall'FPGA prima che il successivo sia pronto.

Per tal motivo, scegliendo il formato 1×32 b, dato che l'FPGA invia al DSP un pacchetto di N_{C} parole a 32 b ogni T_{W} secondi, deve essere necessariamente

$$\frac{32N_{\mathrm{C}}}{T_{\mathrm{W}}} \leq B_{32}\,, \tag{2.2}$$

ossia

$$T_{\mathrm{W}} \geq T_{\mathrm{W32}} = \frac{32N_{\mathrm{C}}}{B_{32}}\,. \tag{2.3}$$

In altre parole, usando il formato 1×32 b, il sistema consente di raggiungere un valore di T_{Wmin} pari a T_{W32}.

Analogamente, scegliendo il formato 2×16 b, dato che l'FPGA invia i pacchetti ogni $2\,T_{\mathrm{W}}$ secondi, deve essere

$$\frac{32N_{\mathrm{C}}}{2T_{\mathrm{W}}} \leq B_{16}\,, \tag{2.4}$$

ossia

$$T_{\mathrm{W}} \geq T_{\mathrm{W16}} = \frac{32N_{\mathrm{C}}}{2B_{16}}\,. \tag{2.5}$$

In questo caso, il sistema consente di raggiungere un valore di T_{Wmin} pari a T_{W16}. Confrontando la (2.5) con la (2.3), affinché il formato 2×16 b porti veramente ad una riduzione di T_{Wmin} deve essere ovviamente $T_{\mathrm{W16}} < T_{\mathrm{W32}}$, ossia $B_{32} < 2B_{16}$.

Infine, scegliendo il formato 4×8 b, i pacchetti sono inviati ogni $4T_{\mathrm{W}}$ secondi, e deve essere

$$\frac{32N_{\mathrm{C}}}{4T_{\mathrm{W}}} \leq B_{8}\,, \tag{2.6}$$

ossia

$$T_{\mathrm{W}} \geq T_{\mathrm{W8}} = \frac{32N_{\mathrm{C}}}{4B_{8}}\,. \tag{2.7}$$

Utilizzando questo formato, il sistema consente di raggiungere un valore di T_{Wmin} pari a T_{W8}. Ancora una volta, confrontando la (2.7) con la (2.5), affinché il formato 4×8 b porti veramente ad una riduzione di T_{Wmin} deve essere $T_{\mathrm{W8}} < T_{\mathrm{W16}}$, ossia $B_{16} < 2B_{8}$.

In conclusione, riunendo le precedenti considerazioni, la formattazione dei risultati di conteggio porta ad un'effettiva riduzione di T_{Wmin} se e solo se

$$B_{32} < 2B_{16} < 4B_{8}\,. \tag{2.8}$$

Le riduzioni di T_{Wmin}, a causa di (2.1), sono comunque vincolate a

$$\frac{T_{\mathrm{W16}}}{T_{\mathrm{W32}}} > \frac{1}{2} \text{ e } \frac{T_{\mathrm{W8}}}{T_{\mathrm{W16}}} > \frac{1}{2}\,. \tag{2.9}$$

Di conseguenza, nello sviluppo del nuovo contatore multicanale si è dovuto

- riprogettare l'FPGA per consentire la formattazione dei risultati;
- ottimizzare i tempi di esecuzione dell'algoritmo real-time per rendere
  - B_{32} la più grande possibile, in modo da ottenere un valore di T_{W32} il più piccolo possibile;
  - B_{16} il più vicino possibile a B_{32}, e B_{8} il più vicino possibile a B_{16}, in modo che, non solo la (2.8) risulti verificata, ma che i rapporti

$$\frac{T_{\mathrm{W16}}}{T_{\mathrm{W32}}} \text{ e } \frac{T_{\mathrm{W8}}}{T_{\mathrm{W16}}}$$

siano il più possibile vicini al loro estremo inferiore teorico di 1/2 imposto dalla (2.9).

2.2 Grafico per l'analisi dei formati disponibili, con o senza saturazione, in funzione delle condizioni di utilizzo

Per fornire all'utente la massima profondità di conteggio ottenibile in base al formato utilizzato, le possibili scelte di L devono essere $L = L_C$, 16, 8. Scegliendo uno di tali valori, il sistema adotterà in modo automatico rispettivamente il formato 1×32 b, 2×16 b o 4×8 b.

Supponendo verificata la (2.8), è interessante capire se, una volta scelto L, le condizioni di utilizzo causino o meno la saturazione dei contatori dell'array.

Per quanto visto precedentemente,

- se $T_W < T_{W8}$ il DSP non è in grado di elaborare in real-time i risultati di conteggio provenienti dall'FPGA;
- se $T_{W8} \leq T_W < T_{W16}$ è utilizzabile il solo formato 4×8 b;
- se $T_{W16} \leq T_W < T_{W32}$ sono utilizzabili solo i formati 4×8 b e 2×16 b;
- se $T_W \geq T_{W32}$ sono utilizzabili i tre formati 4×8 b, 2×16 b e 1×32 b.

Detto f_{IN} il pulse-rate di ingresso del counter array, si consideri il piano (T_W, f_{IN}). Un punto nel piano rappresenta una precisa condizione di utilizzo.

Supponendo di lavorare a destra di T_{W8}, si considerino le seguenti curve:

$$\begin{cases} \gamma_C : & T_W \cdot f_{IN} = 2^{L_C} \\ \gamma_{16} : & T_W \cdot f_{IN} = 2^{16} \\ \gamma_8 : & T_W \cdot f_{IN} = 2^8 \end{cases} \tag{2.10}$$

Tali curve sono rappresentate nel grafico di Fig. 2.1, disegnato su scale logaritmiche. Inoltre, sullo stesso grafico sono riportate le rette $T_W = T_{W8}$, $T_W = T_{W16}$ e $T_W = T_{W32}$ (i valori di T_{W8}, T_{W16} e T_{W32} utilizzati non sono casuali, ma sono quelli relativi al prototipo realizzato, riportati nel Paragrafo 1.6 e discussi nel Paragrafo 5.3).

La curva γ_C rappresenta le coppie (T_W, f_{IN}) che producono un risultato di conteggio esattamente di L_C bit. Analogamente, le curve γ_{16} e γ_8 rappresentano le coppie che producono risultati esattamente di 16 e 8 bit.

In particolare, si osservi che una coppia al di sopra di γ_C produce un risultato più lungo di L_C bit, mentre una coppia al di sotto di γ_C produce un risultato più corto di L_C bit. Di conseguenza, lavorando al di sopra di γ_C i contatori dell'array saturerano sicuramente. Invece, lavorando al di sotto di γ_C è necessario distinguere tre casi:

- se si è scelto $L = L_C$ e dunque il formato 1×32 b, deve essere ovviamente $T_W \geq T_{W32}$ e, sotto tale ipotesi, non si avrà mai saturazione;
- se si è scelto $L = 16$ e dunque il formato 2×16 b, deve essere $T_W \geq T_{W16}$ e, in tal caso, lavorando al di sopra della curva γ_{16} si avrà saturazione, al di sotto no;
- analogamente, se si è scelto $L = 8$ e dunque il formato 4×8 b, deve essere $T_W \geq T_{W8}$ e, in tal caso, lavorando al di sopra della curva γ_8 si avrà saturazione, al di sotto no.

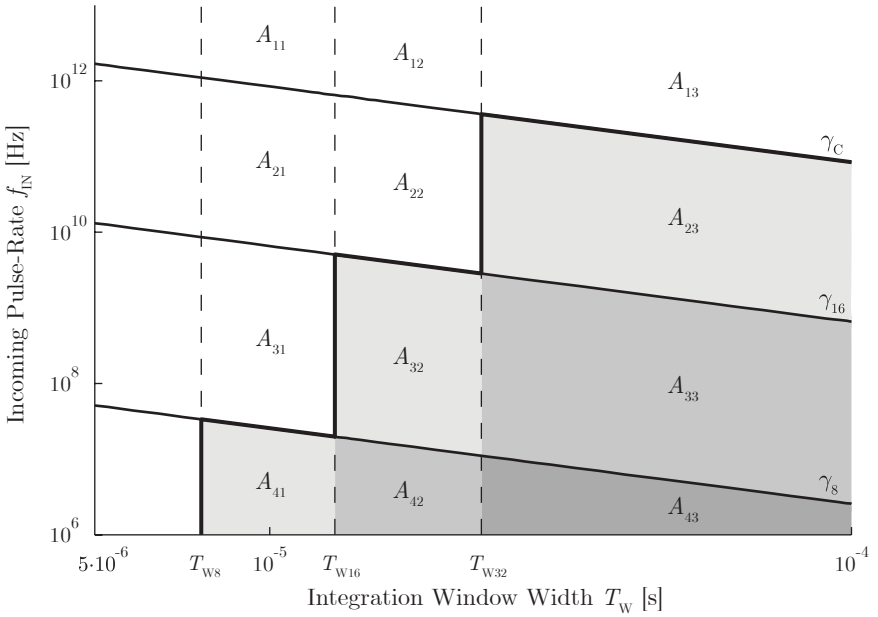


Fig. 2.1 Il piano (T_W, f_{IN}): partizione in funzione dei formati utilizzabili e della saturazione

In definitiva, come indicato nel grafico di Fig. 2.1, il semipiano a destra di T_{W8} risulta diviso nei 12 settori A_{jk} ($j = 1,\ldots,4$; $k = 1,\ldots,3$). Per ciascuno di essi, la Tab. 2.1 riassume i formati che consentono real-time processing e, per ciascun formato, se vi sia o meno saturazione.

A_{jk}	1	2	3
1	4×8 b – with saturation	2×16 b – with saturation 4×8 b – with saturation	1×32 b – with saturation 2×16 b – with saturation 4×8 b – with saturation
2	4×8 b – with saturation	2×16 b – with saturation 4×8 b – with saturation	1×32 b – *without* saturation 2×16 b – with saturation 4×8 b – with saturation
3	4×8 b – with saturation	2×16 b – *without* saturation 4×8 b – with saturation	1×32 b – *without* saturation 2×16 b – *without* saturation 4×8 b – with saturation
4	4×8 b – *without* saturation	2×16 b – *without* saturation 4×8 b – *without* saturation	1×32 b – *without* saturation 2×16 b – *without* saturation 4×8 b – *without* saturation

Tab. 2.1 Formati utilizzabili, con o senza saturazione, per ciascun settore A_{jk} di figura Fig. 2.1

I settori al di sopra della linea marcata di Fig. 2.1, riportata per comodità anche in Tab. 2.1, sono quelli in cui si ha sempre saturazione, indipendentemente dal formato scelto.

Viceversa, i settori al di sotto di tale linea, sono quelli in cui esiste almeno un formato che evita la saturazione. In particolare, tali settori sono colorati con un'opportuna scala di grigi che si intensifica all'aumentare del numero di formati non saturanti.

2.3 Grafico per l'analisi delle prestazioni del contatore

Il grafico di Fig. 2.2 semplicemente estende il grafico di Fig. 2.1, in modo da includere anche T_{Wmax} (come per il grafico di Fig. 2.2, i valori utilizzati non sono casuali, ma sono quelli relativi al prototipo realizzato, riportati nel Paragrafo 1.6 e discussi nel Paragrafo 5.3).

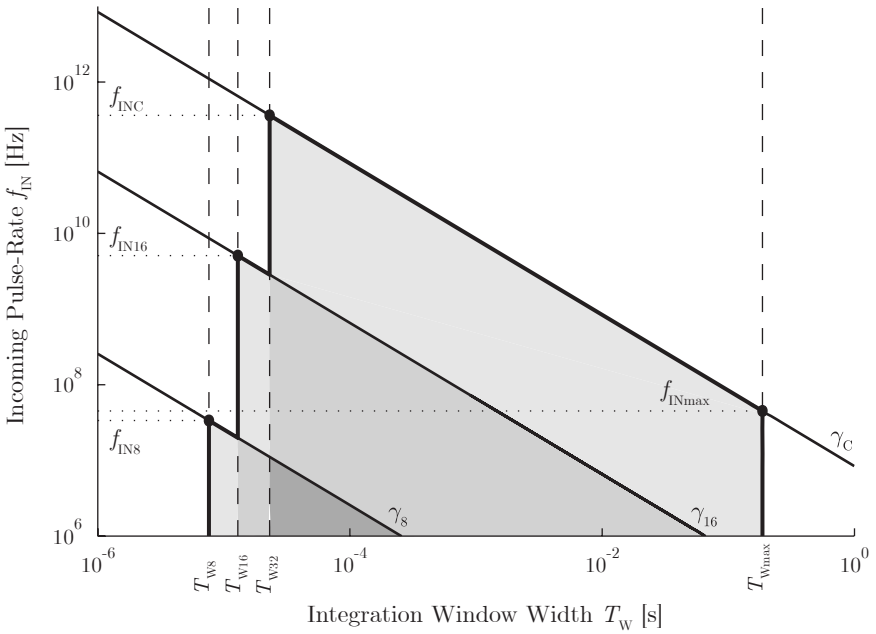


Fig. 2.2 Grafico per l'analisi delle prestazioni del contatore

Osservando tale grafico è facile determinare:

- per un dato f_{IN}, la più piccola finestra di integrazione T_{W} raggiungibile senza saturazione ed il relativo formato da utilizzare;
- per una data T_{W}, il più grande pulse-rate f_{IN} raggiungibile senza saturazione ed il relativo formato da utilizzare.

Si consideri ad esempio il primo problema. Siano f_{INC}, f_{IN16} ed f_{IN8} le ordinate dei punti di γ_{C} , γ_{16} e γ_{8} di ascisse rispettivamente T_{W32}, T_{W16} e T_{W8}. Dalla (2.10) e dalle definizioni di T_{W32}, T_{W16} e T_{W8} si ha:

$$\begin{cases} f_{\mathrm{INC}} = \dfrac{2^{L_{\mathrm{C}}}}{T_{\mathrm{W32}}} = \dfrac{2^{L_{\mathrm{C}}}}{32} \cdot \dfrac{B_{32}}{N_{\mathrm{C}}} \\ f_{\mathrm{IN16}} = \dfrac{2^{16}}{T_{\mathrm{W16}}} = \dfrac{2^{16}}{16} \cdot \dfrac{B_{16}}{N_{\mathrm{C}}} \\ f_{\mathrm{IN8}} = \dfrac{2^{8}}{T_{\mathrm{W8}}} = \dfrac{2^{8}}{8} \cdot \dfrac{B_{8}}{N_{\mathrm{C}}} \end{cases} \tag{2.11}$$

Si conclude immediatamente che:

- se $f_{\mathrm{IN}} \leq f_{\mathrm{IN8}}$, la più piccola finestra di integrazione raggiungibile senza saturazione coincide con la minima finestra gestibile dal sistema, ossia con T_{W8}, e l'unico formato utilizzabile è ovviamente 4×8 b;
- se $f_{\mathrm{IN8}} < f_{\mathrm{IN}} \leq f_{\mathrm{IN16}}$, la più piccola finestra raggiungibile senza saturazione è T_{W16} e l'unico formato utilizzabile è 2×16 b;
- se $f_{\mathrm{IN16}} < f_{\mathrm{IN}} \leq f_{\mathrm{INC}}$, la più piccola finestra raggiungibile senza saturazione è T_{W32} e l'unico formato utilizzabile è 1×32 b;
- infine, se $f_{\mathrm{IN}} > f_{\mathrm{INC}}$, qualunque finestra e qualunque formato produrranno la saturazione dei contatori dell'array.

Si osservi infine che nel grafico è riportata anche la retta $f_{\mathrm{IN}} = f_{\mathrm{INmax}}$. Ovviamente, l'utilizzo del contatore ha senso solo per $f_{\mathrm{IN}} \leq f_{\mathrm{INmax}}$. Di conseguenza, i punti del piano che rappresentano una vera condizione di funzionamento sono quelli che giacciono al di sotto di tale retta ed hanno un'ascissa compresa tra T_{W8} e T_{Wmax}.

Capitolo 3. L'architettura dell'FPGA

3.1 Il top-level

livello gerarchico: top

Come mostrato in Fig. 1.4, l'FPGA risulta connessa direttamente al connettore MicroLine, alla DataMover IF dell'LLC, e al bus EMIF del DSP. Inoltre, riceve in ingresso il segnale di clock a 90 MHz proveniente dalla stessa EMIF, ed è in grado di interrompere il DSP utilizzando due delle sue diverse linee di interrupt esterne.

Al fine di gestire le operazioni di conteggio, e di consentire la comunicazione sia verso la DataMover IF che verso il DSP, l'architettura interna dell'FPGA, al più alto livello gerarchico, è stata organizzata nei moduli rappresentati in Tav. A.1 e schematizzati in Fig. 3.1.

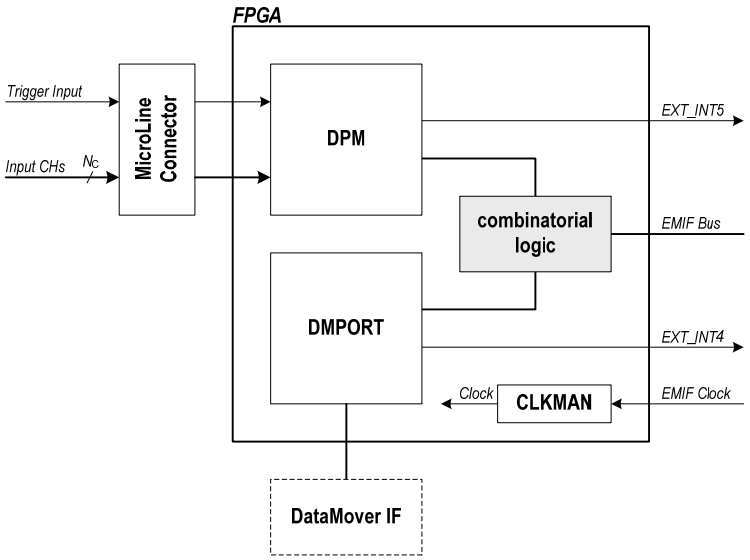


Fig. 3.1 Architettura interna dell'FPGA al più alto livello gerarchico

Il *Data Processing Module* (DPM) ha lo scopo di ricevere dal DSP tutti i parametri di funzionamento, di acquisire e contare gli impulsi provenienti dagli N_C canali di ingresso, di scandire le finestre di integrazione T_W secondo la prescelta modalità di acquisizione, di formattare i risultati di conteggio e, infine, di segnalare al DSP la presenza di un pacchetto di risultati pronto per essere elaborato.

Il *DataMover Port IP Core* (DMPORT) è invece una *intellectual property* fornita da Orsys [16]. Il suo scopo è quello di interfacciarsi direttamente alla DataMover IF dell'LLC, così da facilitare la gestione delle transazioni isocrone in uscita sul bus IEEE 1394. Infatti, visto dagli altri moduli interni all'FPGA, il DMPORT può essere pensato come un buffer FIFO dual-port, contenente

1024 parole da 32 b. Una volta che il DSP ha configurato opportunamente l'LLC, è possibile avviare una qualunque trasmissione sul link isocrono andando semplicemente a scrivere i dati che si vogliono trasmettere all'interno di tale buffer.

Sia il DPM che il DMPORT offrono al DSP un insieme di registri di interfaccia, dedicati a operazioni sia di setup, sia di comunicazione. In particolare, entrambi i moduli presentano un bus dati in ingresso a 32 b, un bus dati in uscita a 32 b, un bus indirizzi, e opportuni segnali di abilitazione alla scrittura e alla lettura. Infine, ciascuno di essi è in grado di interrompere il DSP utilizzando le due linee di interrupt (CPU_EXT_INT5_PAD e CPU_EXT_INT4_PAD) a cui l'FPGA è connessa.

La logica racchiusa nell'area ombreggiata gestisce opportunamente i segnali del bus EMIF, consentendo al DSP di comunicare separatamente sia con il DPM che con il DMPORT.

Il bus EMIF, visto dall'FPGA, è costituito da un bus dati a 32 b bidirezionale (CPU_D_PAD[31..0]), da un bus indirizzi a 20 b (CPU_A_PAD[21..2]), da un segnale di *chip-enable* (CPU_CE2n_PAD), e da altri segnali di abilitazione per le operazioni di scrittura (CPU_AWEn_PAD) e di lettura (CPU_AREn_PAD, CPU_AOEn_PAD). Tutti questi segnali sono sincroni rispetto al clock EMIF, e le temporizzazioni relative alle operazioni di lettura e di scrittura sono impostabili dall'utente esprimendole in numero di cicli di clock. Inoltre, i 20 b del bus indirizzi corrispondono solo ad una parte dei 32 b [31..0] di indirizzamento utilizzati normalmente dal DSP e, più precisamente, al sottoinsieme [21..2].

In particolare, lo spazio di indirizzamento del DSP è suddiviso, a livello logico, in molti sottospazi consecutivi, ciascuno dei quali è riservato alla comunicazione verso opportune periferiche, sia interne che esterne. Il sottospazio che si estende dall'indirizzo $\mathsf{A000\,0000_h}$ all'indirizzo $\mathsf{AFFF\,FFFF_h}$, denominato *EMIF CE2 Space*, è dedicato alla comunicazione verso l'FPGA. Ogni volta che il DSP accede ad una locazione mappata all'interno del CE2 Space, l'EMIF provvede non solo a gestire opportunamente i segnali relativi a dati, indirizzi e abilitazioni, ma ad attivare anche il segnale di chip-enable. Dunque, da lato FPGA, è possibile sapere se il DSP vuole instaurare una comunicazione con il DPM o il DMPORT semplicemente osservando il valore logico di tale segnale.

Inoltre, dato che il bus indirizzi EMIF è largo 20 b, all'interno dell'FPGA si potranno indirizzare fino a 2^{20} locazioni. Per i nostri scopi, 2^{19} indirizzi sia per il DPM che per il DMPORT sono più che sufficienti. Si è pensato quindi di utilizzare il bit più significativo di tale bus per dividere in due parti uguali le 2^{20} locazioni possibili: la prima riservata al DPM (CPU_A_PAD[21] = 0), la seconda riservata al DMPORT (CPU_A_PAD[21] = 1). In questo modo, la logica racchiusa nell'area ombreggiata, necessaria a collegare il DSP al DPM o al DMPORT, risulta estremamente semplice.

In particolare, tale logica, quando il segnale di chip-enable è attivo, provvede a inoltrare i segnali di abilitazione o al DPM o al DMPORT, a seconda del

valore di CPU_A_PAD[21]. Inoltre, al fine di gestire le richieste di lettura provenienti dal DSP, tale logica controlla anche un multiplexer ed una porta tristate, in modo da instradare opportunamente i dati provenienti dal bus di uscita di ciascun modulo verso il bus dati EMIF bidirezionale.

Infine, in Tav. A.1, così come in Fig. 3.1, è riportato anche il modulo *Clock Manager*. Tale modulo rappresenta l'istanza di uno dei molti gestori di clock di cui l'FPGA dispone. Il suo compito è quello di agganciare il segnale di clock proveniente dal DSP (CPU_ECLKOUT_PAD) e di garantirne la distribuzione all'interno di tutta l'FPGA, riducendo il più possibile i problemi di skew.

3.2 Il Data Processing Module (DPM)

livello gerarchico: top/DPM

L'architettura interna del DPM è schematizzata in Fig. 3.2.

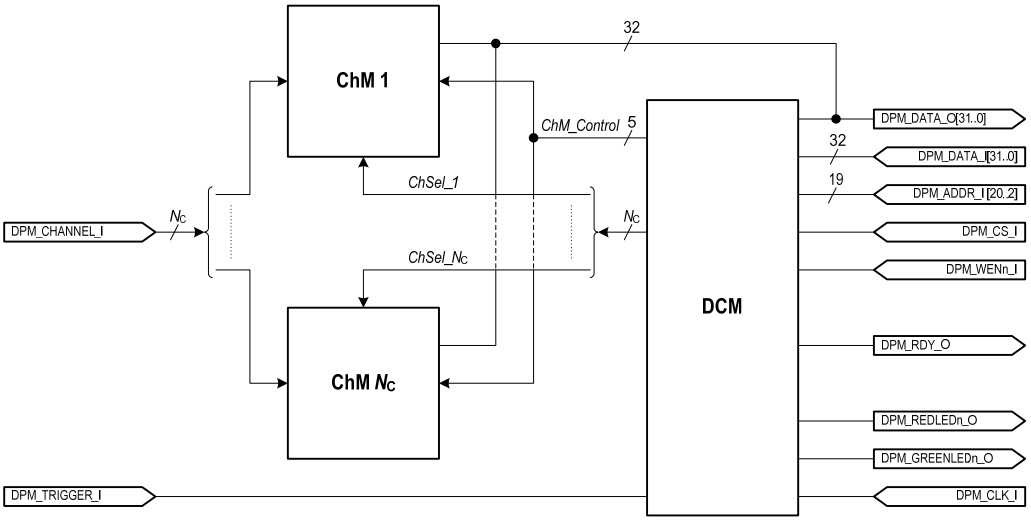


Fig. 3.2 Architettura del Data Processing Module

Il DPM è anch'esso organizzato in moduli: un DPM Control Module (DCM) ed N_C Channel Module (ChM).

Ciascun ChM ha il compito di acquisire e contare gli impulsi provenienti dal canale a cui è collegato, di formattare i risultati di conteggio in un'unica parola a 32 b, e di fornire in uscita tale parola.

Tutti i ChM operano in sincronia, e le loro operazioni sono controllate dal DCM per mezzo dei segnali ChM_Control. Il DCM genera opportunamente tali segnali a partire dalle impostazioni di acquisizione e dai comandi di start/stop ricevuti dal DSP.

Ogni volta che i ChM hanno completato la loro parola di uscita a 32 b, il DCM interrompe immediatamente il DSP, inviando un impulso sulla porta DPM_RDY_O. In tal modo, il DSP può accedere a ciascun ChM e leggere i risultati di conteggio.

Si osservi infatti che ciascun ChM è connesso al bus dati di uscita del DPM (DPM_DATA_O[31..0]). A ciascuno di essi è assegnato un preciso indirizzo tra quelli riservati al DPM. Quando il DSP vuole accedere al j-esimo ChM, il DCM ne decodifica l'indirizzo e attiva il corrispondente segnale ChSel_j. In questo modo, il ChM selezionato viene abilitato a inviare sul bus dati la propria parola di uscita.

Il DCM controlla inoltre i segnali DPM_REDLEDn_O e DPM_GREENLEDn_O, attivandoli in base al suo stato di funzionamento. In questo modo, dato che tali segnali sono direttamente collegati ai led presenti sulla scheda Orsys, dall'esterno è possibile conoscere lo stato di funzionamento corrente del counter array. In particolare, tale informazione risulta di fondamentale importanza sia in fase di test, sia in fase di debug.

Inoltre si osservi che, avendo scelto di organizzare il DPM in un modulo di controllo (il DCM) e in N_C moduli operativi (i ChM), è stato possibile rendere il progetto di tali moduli sostanzialmente indipendente dal numero di canali N_C. In particolare, ciò che varia con N_C è soltanto la logica interna al DCM dedicata alla decodifica degli indirizzi dei vari ChM e alla generazione dei segnali ChSel_j.

Infine, dal momento che il generico ChM necessita di essere istanziato N_C volte all'interno del DPM, per poter sfruttare al meglio le risorse logiche messe a disposizione dall'FPGA e massimizzare N_C, si è cercato di rendere tale modulo il più semplice possibile.

3.3 Il Channel Module (ChM)

livello gerarchico: top/DPM/ChM

L'architettura del j-esimo ChM, riportata in dettaglio in Tav. A.2, è schematizzata in Fig. 3.3.

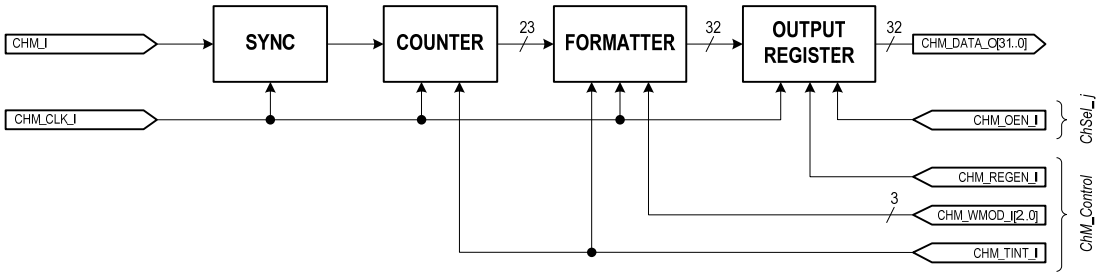


Fig. 3.3 Architettura del j-esimo ChM

Come si nota, ciascun ChM è formato da quattro componenti: un sincronizzatore, un contatore, un formatter ed un registro di uscita a 32 b.

In particolare, i segnali di controllo provenienti dal DCM, precedentemente indicati con ChM_Control, sono connessi a CHM_TINT_I, CHM_WMOD_I[2..0] e CHM_REGEN_I, mentre il segnale di selezione ChSel_j è connesso a CHM_OEN_I.

3.3.1 Il sincronizzatore

Il sincronizzatore ha il compito di acquisire il segnale di ingresso, agganciando le sue transizioni positive e generando, per ciascuna di esse, un impulso sincrono di durata pari ad un ciclo di clock.

In particolare, si è scelto di implementare tale sincronizzatore utilizzando la semplice soluzione convenzionale a due flip-flop riportata in Fig. 3.4, in cui il primo di essi è resettato in modo asincrono dal secondo.

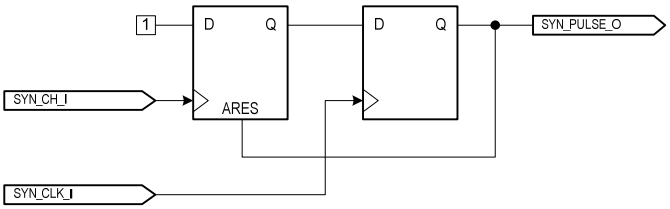


Fig. 3.4 Architettura del sincronizzatore

Analizzando tale circuito, è possibile osservare che il pulse-rate di uscita del sincronizzatore può raggiungere al massimo la metà della frequenza di clock f_{CLK} di sistema. Di conseguenza, affinché il counter array riconosca correttamente gli impulsi provenienti dai suoi canali di ingresso, è necessario che il loro pulse-rate f_{IN} soddisfi la relazione

$$f_{\mathrm{IN}} \leq f_{\mathrm{INmax}} = \frac{f_{\mathrm{CLK}}}{2}. \tag{3.1}$$

Per questo motivo, essendo $f_{\mathrm{CLK}} = 90$ MHz, si ha $f_{\mathrm{INmax}} = 45$ MHz.

3.3.2 Il contatore

Il successivo contatore, oltre ovviamente a ricevere in ingresso il segnale di clock e a fornire in uscita il risultato di conteggio, riceve anche un segnale di abilitazione (PCNT_EN_I) ed un segnale di reset sincrono (PCNT_SRES_I). Il primo coincide con il segnale di uscita del sincronizzatore, il secondo con il segnale CHM_TINT_I proveniente dal DCM.

Il DCM infatti, allo scadere di ogni finestra di integrazione, invia sulla porta CHM_TINT_I un impulso di durata pari ad un ciclo di clock. In questo modo, ogni T_{W} secondi, il contatore è in grado di ricominciare il conteggio dei fronti in salita rilevati dal sincronizzatore.

In particolare, il contatore è stato progettato in modo da rendere l'abilitazione prioritaria rispetto al reset. In altre parole, quando PCNT_EN_I e PCNT_SRES_I sono contemporaneamente attivi, il contatore porta la sua uscita a 1 anziché a 0. In questo modo, è automaticamente evitato il rischio di "perdere" i fronti che, per pura casualità, possono presentarsi esattamente in concomitanza dello scadere di una finestra di integrazione.

Come si è appena accennato, il DCM ha il compito di scandire le finestre di integrazione. Al fine di minimizzare lo step e, al tempo stesso, di estendere il range di variazione di T_{W}, durante il progetto del DCM si è scelto di esprimere tale valore direttamente in numero di cicli di clock, utilizzando una rappresentazione a 24 b. In questo modo, lo step di variazione di T_{W} coincide con il periodo di clock $1/f_{\mathrm{CLK}} \simeq 11$ ns, ed è inoltre possibile raggiungere il valore di T_{Wmax} dato da

$$T_{\mathrm{Wmax}} = \frac{2^{24}}{f_{\mathrm{CLK}}} \simeq 186 \text{ ms}\,. \tag{3.2}$$

Sostituendo nella relazione (1.1) le espressioni di T_{Wmax} e f_{INmax} ricavate in (3.1) e in (3.2), si ottiene $L_{\mathrm{C}} = 23$. Dunque, il contatore del generico ChM è stato progettato in modo da avere una profondità di conteggio pari a 23 b.

Avendo fissato $L_{\mathrm{C}} = 23$, in base alle considerazioni svolte nei Paragrafi 2.1 e 2.2 relative all'ottimizzazione di T_{Wmin}, il sistema deve consentire la scelta di $L = 23,\ 16,\ 8$. A tali valori corrispondono rispettivamente i formati 1×32 b, 2×16 b e 4×8 b.

Una volta che l'utente ha scelto uno dei valori di L, il DCM ha il compito di controllare opportunamente il formatter di ciascun ChM in modo che, allo scadere rispettivamente di 1, 2 o 4 finestre di integrazione consecutive, quest'ultimo sia in grado di fornire la parola a 32 b da scrivere nel successivo registro di uscita. In particolare, tale operazione di scrittura viene gestita dallo stesso DCM per mezzo del segnale di abilitazione CHM_REGEN_I.

3.3.3 Il formatter ed il registro di uscita

L'architettura del formatter è rappresentata in Fig. 3.5.

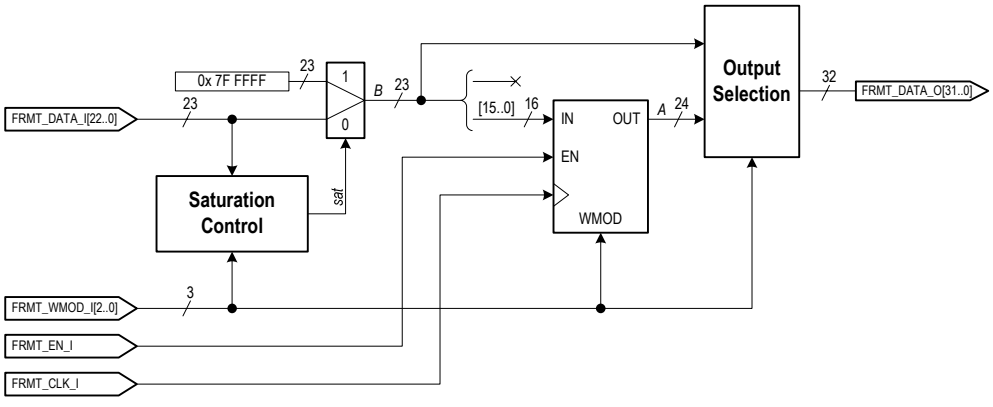


Fig. 3.5 Architettura del formatter

Il formatter è costituito da tre componenti: una rete combinatoria dedicata al controllo di saturazione, una rete combinatoria di uscita, ed un registro a 24 b con particolari funzioni di scrittura. Ciascuno di essi è controllato dai segnali FRMT_WMOD_I[2..0], che, come mostrato in Tav. A.2, coincidono con i segnali

CHM_WMOD_I[2..0] provenienti dal DCM. Inoltre, il registro a 24 b riceve anche un segnale di abilitazione FRMT_EN_I coincidente con CHM_TINT_I.

Facendo riferimento alla Fig. 3.5, a seconda del formato prescelto, le operazioni svolte dal formatter sono le seguenti:

- se $L = 23$,
  - durante ogni finestra di integrazione si ha:
    - FRMT_DATA_O[31..23] = 0000 0000 0_b;
    - FRMT_DATA_O[22..0] = B[22..0].

  In generale, la rete per il controllo della saturazione osserva continuamente il risultato fornito dal contatore, e non appena tale risultato eccede il massimo valore rappresentabile in base al formato scelto, commuta il multiplexer in posizione 1, in modo da fornire a tutti i circuiti a valle un risultato saturato. Tuttavia, per $L = 23$ non si ha mai saturazione. Dunque, in questo caso, tale rete non compie alcuna operazione, e mantiene il multiplexer in posizione 0, con B = FRMT_DATA_I.

  In questo modo, l'intero formatter risulta trasparente rispetto ai dati di uscita a 23 b del contatore. Inoltre, per completare la parola a 32 b, la rete combinatoria di uscita estende semplicemente tali dati aggiungendo nove zeri nelle posizioni più significative.

  Di conseguenza, per poter campionare nel registro di uscita i risultati provenienti dal contatore, il DCM, allo scadere di ogni finestra, invia su CHM_REGEN_I lo stesso impulso normalmente inviato su CHM_TINT_I.
- se $L = 16$,
  - allo scadere della prima finestra,
    - A[15..0] $\leftarrow$ B[15..0];
  - durante la seconda finestra,
    - FRMT_DATA_O[31..16] = B[15..0];
    - FRMT_DATA_O[15..0] = A[15..0].

  In altre parole, allo scadere della prima finestra, grazie all'impulso su CHM_TINT_I, il registro interno al formatter campiona nei suoi 16 b meno significativi di uscita A[15..0], i 16 b di ingresso B[15..0], ossia il primo risultato di conteggio da includere nella parola a 32 b. Successivamente, durante la seconda finestra, la rete combinatoria di uscita compone la parola a 32 b utilizzando tale risultato per i 16 b meno significativi, e instradando B[15..0] nei rimanenti 16 b.

  Inoltre, in questo caso, la rete per il controllo della saturazione è attiva, e osserva continuamente i bit FRMT_DATA_I[22..16]: non appena trova uno solo di essi uguale a 1, commuta immediatamente il multiplexer.

  Infine, allo scadere della seconda finestra, il DCM abilita la scrittura nel registro di uscita inviando su CHM_REGEN_I lo stesso impulso che invia su CHM_TINT_I. In questo modo, il registro di uscita preleva dal re-

gistro interno il primo risultato precedentemente campionato e, allo stesso tempo, campiona direttamente il secondo.

- se $L = 8$,
  - allo scadere della prima finestra,
    - A[7..0] ← B[7..0];
  - allo scadere della seconda finestra,
    - A[15..8] ← B[7..0];
  - allo scadere della terza finestra,
    - A[23..16] ← B[7..0];
  - durante la quarta finestra,
    - FRMT_DATA_O[31..24] = B[7..0];
    - FRMT_DATA_O[23..0] = A[23..0].

  Analogamente al caso precedente, allo scadere di ciascuna delle prime tre finestre, il registro interno campiona gli 8 b meno significativi del segnale proveniente dal multiplexer, memorizzandone il valore rispettivamente in A[7..0], A[15..8] e A[23..16]. Allo scadere della terza finestra, tale registro contiene dunque i tre risultati di conteggio già opportunamente ordinati. Successivamente, durante la quarta finestra, la rete combinatoria di uscita compone la parola a 32 b utilizzando tali risultati per i 24 b meno significativi, e instradando B[7..0] nei rimanenti 8 b.

  Anche in questo caso, la rete per il controllo della saturazione è attiva, ma stavolta osserva continuamente i bit FRMT_DATA_I[22..8].

  Infine, allo scadere della quarta finestra, il DCM invia su CHM_REGEN_I lo stesso impulso che invia su CHM_TINT_I. In questo modo, il registro di uscita preleva dal registro interno i primi tre risultati precedentemente campionati e, allo stesso tempo, campiona direttamente il quarto.

Da ultimo, si osservi che, come descritto nel Paragrafo 3.2, tutti i registri di uscita dei ChM si affacciano direttamente sul bus dati DPM_DATA_O[31..0]. Questo avviene perché tali registri presentano un'uscita di tipo tristate: soltanto il registro selezionato dal DCM per mezzo del relativo segnale CHM_OEN_I è abilitato a imporre sul bus la propria parola a 32 b.

3.4 Il DPM Control Module (DCM)

livello gerarchico: top/DPM/DCM

L'architettura del DCM è schematizzata in Fig. 3.6 e riportata in dettaglio in Tav. A.3 e in Tav. A.4.

Come si può notare, il DCM offre al DSP un set di registri di interfaccia (IF REGs), la gestione dei quali è affidata alla rete di decodifica degli indirizzi (ADDR DEC) rappresentata in dettaglio nell'area ombreggiata di Tav. A.4.

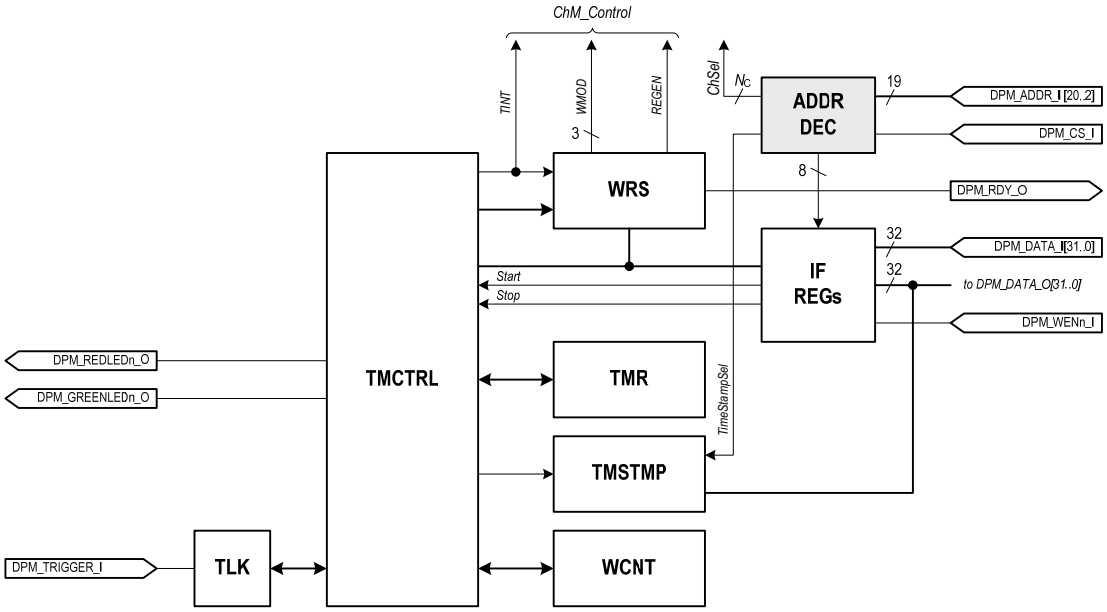


Fig. 3.6 Architettura del DCM

3.4.1 La rete di decodifica degli indirizzi

In stretta analogia con la rete combinatoria di Fig. 3.1, il primo compito di ADDR DEC è quello di dividere i 2^{19} indirizzi riservati al DPM in due parti uguali, utilizzando, a tale scopo, il più significativo dei 19 b di indirizzamento disponibili in ingresso, ossia DPM_ADDR_I[20]. In particolare, la parte che si ottiene per DPM_ADDR_I[20] = 0, denominata *System Space*, è dedicata ai registri interni del DCM, mentre l'altra, ottenuta per DPM_ADDR_I[20] = 1 e denominata *Channel Space*, è dedicata all'indirizzamento dei vari ChM.

Inoltre, per poter effettivamente selezionare uno dei registri interni al DCM o il registro di uscita di un qualunque ChM, tale rete dispone di due decoder, uno per ogni parte.

Riguardo al System Space, dal momento che i parametri di acquisizione che il DSP deve comunicare al DCM sono relativamente pochi, si è pensato di indirizzare all'interno di tale spazio soltanto 8 locazioni, utilizzando un semplice decoder 3 a 8 (DPM_SYSDEC).

Viceversa, per il Channel Space, dovendo indirizzare gli N_C ChM, è stato ovviamente utilizzato un decoder a N_C uscite (DPM_CHDEC). In particolare, tale decoder è stato descritto in modo completamente parametrico, cosicché in fase di sintesi, una volta fissato N_C, la sua implementazione possa avvenire in modo automatico.

In definitiva, come si può osservare in Tav. A.4, le N_C uscite del DPM_CHDEC (ossia i segnali ChSel) raggiungono i vari ChM, mentre le 8 uscite del DPM_SYSDEC raggiungono i registri di interfaccia del DCM.

3.4.2 I registri di interfaccia

L'architettura del generico registro di interfaccia è mostrata in Fig. 3.7a.

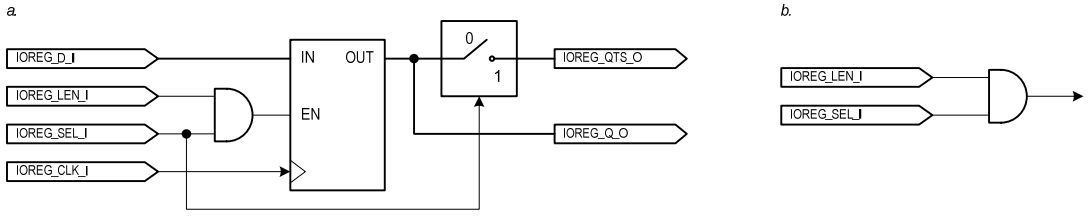


Fig. 3.7 Registri (a) e pseudo-registri (b) di interfaccia utilizzati all'interno del DCM

I registri di interfaccia, oltre alla normale uscita IOREG_Q_O, presentano anche l'uscita tristate IOREG_QTS_O. La prima è utilizzata per il trasferimento dei dati verso i dispositivi interni al DCM, la seconda per la gestione delle richieste di lettura provenienti dal DSP.

Infatti, la porta tristate è controllata dal segnale di selezione IOREG_SEL_I, proveniente da una delle 8 linee di uscita del DPM_SYSDEC. In questo modo, dal momento che tutte le uscite IOREG_QTS_O dei vari registri di interfaccia sono connesse al bus dati di uscita del DPM, quando il DSP vuole effettuare una lettura, il dato contenuto nel registro selezionato viene direttamente instradato verso il bus dati EMIF.

Inoltre, lo stesso segnale IOREG_SEL_I gioca un ruolo di fondamentale importanza per le operazioni di scrittura. Infatti, tutti i registri di interfaccia ricevono sull'ingresso IOREG_D_I i dati provenienti dal bus EMIF, e sull'ingresso IOREG_LEN_I l'impulso di abilitazione alla scrittura. Tuttavia, grazie alla presenza della porta AND, solo quello selezionato dal DSP memorizzerà effettivamente il dato inviato.

Un simile meccanismo è stato sfruttato anche per la gestione dei comandi di start e di stop che il DSP deve inviare al DCM. Infatti, come si vedrà in seguito, al fine di controllare l'avvio e l'arresto delle operazioni di acquisizione e di conteggio, è sufficiente inviare ai dispositivi interni al DCM un impulso rispettivamente sui segnali Start o Stop mostrati in Fig. 3.6.

Si è pensato dunque di mappare all'interno del System Space due *pseudo-registri* come quello di Fig. 3.7b: l'uscita del primo è stata connessa al segnale Start, quella del secondo al segnale Stop. Ovviamente, gli pseudo-registri, essendo solamente costituiti da una porta AND, non hanno niente a che vedere con i normali registri di interfaccia. Tuttavia, al DSP appaiono come tali.

In questo modo, effettuando ad esempio una qualunque operazione di scrittura sullo pseudo-registro di start, è possibile inoltrare l'impulso di abilitazione proveniente dal bus EMIF direttamente sul segnale Start, avviando le operazioni di acquisizione e di conteggio. Per il segnale Stop il discorso è analogo.

Si noti comunque che una tale soluzione, se da un lato risulta estremamente semplice, basandosi sull'utilizzo di una sola porta logica, dall'altro richiede il sacrificio di un'intera locazione dello spazio di indirizzamento per un'informazione che potrebbe essere contenuta addirittura in un unico bit. Nel nostro caso, la scelta di tale soluzione è stata resa possibile proprio a causa della grande ampiezza del System Space e delle poche locazioni necessarie per i registri di interfaccia.

Infine, osservando la Tav. A.4, la mappatura dei registri all'interno del System Space e il loro stesso dimensionamento potrebbero apparire ingiustificati. In realtà, come si vedrà in seguito, una tale scelta è dovuta alla volontà di rendere il nuovo contatore multicanale immediatamente utilizzabile al posto di quello descritto nel Paragrafo 1.5. Inoltre, è bene precisare che tale scelta è del tutto ininfluente a livello funzionale, e che può essere facilmente modificata o riadattata in base alle esigenze del sistema di misura in cui il contatore sarà utilizzato.

3.4.3 Il Timer Controller e il Write Selector

Come mostrato in Fig. 3.6, i segnali di Start e Stop raggiungono il *Timer Controller* (TMCTRL), che costituisce la principale macchina a stati dedicata al controllo di tutto l'array. Il suo compito è quello di scandire le finestre di integrazione secondo la modalità di acquisizione prescelta, di inviare per mezzo del segnale TINT gli impulsi di fine finestra a tutti i ChM, e di controllare l'attività della macchina a stati *Write Selector* (WRS).

Tale macchina, coerentemente con quanto visto nel Paragrafo 3.3, in base al valore di L e alla finestra di integrazione corrente, genera i segnali di controllo da inviare ai formatter (WMOD) e ai registri di uscita (REGEN). Inoltre, non appena tali registri contengono una nuova parola a 32 b, interrompe il DSP inviando un impulso su DPM_RDY_O.

3.4.4 Gli altri dispositivi controllati dal Timer Controller

Per svolgere le sue funzioni, il Timer Controller si serve di tre componenti: un timer a 24 b (TMR), un sincronizzatore (TLK) analogo a quello presente in ciascun ChM, ed un contatore a 8 b (WCNT). Il timer è utilizzato in entrambe le modalità di acquisizione, sia per scandire le finestre di integrazione T_{W}, sia per misurare il delay D. Viceversa, il sincronizzatore e il contatore sono utilizzati solo in modalità triggerata: il primo per agganciare il segnale di trigger proveniente dall'esterno, il secondo per contare i pacchetti inviati dall'array al DSP.

Si osservi infatti che, in modalità triggerata, formattando 2 o 4 risultati in una singola parola a 32 b, e sospendendo l'acquisizione dopo un certo numero N_{W} di finestre di integrazione, c'è il rischio che l'ultimo pacchetto di risultati rimanga incompleto. Questo accade ovviamente se N_{W} non è un multiplo né di 2, né di 4. Di conseguenza, per evitare di inviare al DSP parole contenenti bit

senza alcun significato, in modalità triggerata l'utente è chiamato a specificare non tanto il valore di N_{W}, quanto piuttosto il numero di pacchetti N_{PKT} che il sistema deve completare prima di sospendere l'acquisizione e tornare in attesa di trigger.

Ovviamente, N_{W} e N_{PKT} sono strettamente legati tra loro. Infatti, in base al formato selezionato, indicando con $F = 1, 2, 4$ il numero di risultati di conteggio inclusi in ciascuna parola a 32 b, si ha

$$N_{\mathrm{W}} = N_{\mathrm{PKT}} \cdot F \tag{3.3}$$

Avendo scelto di esprimere N_{PKT} su 8 b, a seconda del valore di F – e quindi del formato selezionato – N_{W} può raggiungere rispettivamente il valore di 255, 510 o 1020, con passi di 1, 2 o 4 unità.

Non appena il WCNT raggiunge il valore N_{PKT}, il Timer Controller sospende l'acquisizione, lo resetta e torna in attesa di trigger. In particolare, impostando $N_{\mathrm{PKT}} = 0$, il sistema si comporta come se fosse $N_{\mathrm{PKT}} = \infty$. In tal caso, infatti, dopo l'arrivo del primo trigger, l'acquisizione continua ininterrottamente fino all'arrivo del comando di stop.

3.4.5 Il Time Stamp

Infine, il Timer Controller gestisce anche un altro contatore, denominato *Time Stamp* (TMSTMP), anch'esso utilizzato per contare i pacchetti inviati dall'array al DSP. Tuttavia, a differenza di WCNT, tale contatore è utilizzato sempre, indipendentemente dalla modalità di acquisizione prescelta, e viene resettato solo all'arrivo del comando di start.

In questo modo, in un qualunque istante, il suo valore rappresenta il numero progressivo dell'ultimo pacchetto inviato al DSP a partire dall'inizio della sessione di lavoro corrente.

In particolare, tale contatore è mappato all'interno del Channel Space, immediatamente prima del ChM 1. La sua uscita tristate, connessa ovviamente al bus dati di uscita del DPM, è abilitata dal segnale TimeStampSel, proveniente dalla rete ADDR DEC.

In questo modo, il DSP, a seconda dell'applicazione real-time implementata, ogni volta che viene interrotto dal DCM, può leggere o un pacchetto di N_{C} parole, o uno di $N_{\mathrm{C}} + 1$ parole, in cui la prima di esse contiene il numero progressivo. Una tale informazione può servire ad esempio per svolgere operazioni di controllo di integrità sui dati di volta in volta ricevuti, in modo da segnalare all'utente, in caso di malfunzionamento, l'eventuale perdita di uno o più pacchetti.

3.5 Funzionamento del Timer Controller (TMCTRL)

livello gerarchico: top/DPM/DCM/TMCTRL

In Fig. 3.8 è riportato il diagramma degli stati del Timer Controller.

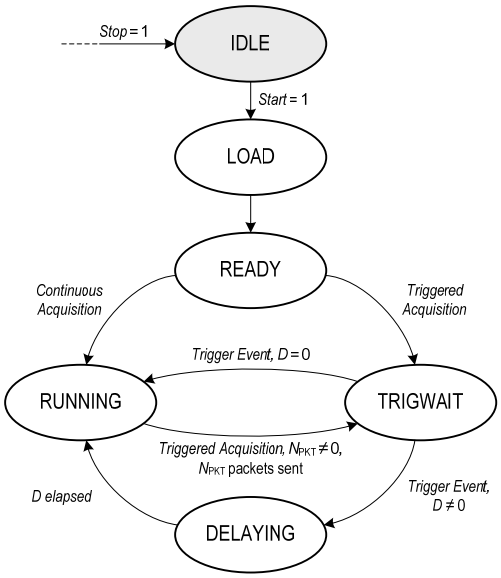


Fig. 3.8 Diagramma degli stati del Timer Controller (TMCTRL)

Come si può notare, lo stato di reset coincide con IDLE, ed il segnale di reset con Stop. In IDLE, il Timer Controller non esegue alcuna operazione, e tutti i dispositivi da esso controllati sono disattivati.

All'arrivo dell'impulso di Start, la macchina si porta nello stato LOAD. In tale stato, i parametri di acquisizione contenuti nei registri di interfaccia vengono trasferiti nei registri interni. I dispositivi controllati sono ancora disattivati.

Al clock successivo, la macchina entra nello stato READY, grazie al quale, in base ai parametri ricevuti, si predispone ad avviare le successive operazioni di acquisizione. I dispositivi controllati continuano ad essere disattivati.

Se l'utente ha scelto la modalità continua, al clock successivo la macchina si porta nello stato RUNNING, viceversa, nel caso di modalità triggerata, nello stato TRIGWAIT.

In particolare, in TRIGWAIT, il Timer Controller attiva il sincronizzatore TLK e si posiziona in attesa di trigger. All'arrivo del trigger, se l'utente ha scelto $D = 0$, la macchina entra nello stato RUNNING, altrimenti in DELAYING.

In DELAYING, il Timer Controller misura il delay D attivando il timer TMR. Trascorso tale ritardo, entra automaticamente in RUNNING.

In RUNNING, sono attivi il timer TMR, la macchina WRS e, in caso di acquisizione triggerata con $N_{\mathrm{PKT}} \neq 0$, anche il contatore WCNT. In particolare, ogni volta che TMR raggiunge il valore T_{W} scelto dall'utente, il Timer Controller

invia su TINT l'impulso di fine finestra che, come mostrato in Tav. A.3, raggiunge la macchina WRS, i vari ChM e lo stesso timer, causandone il reset.

Se l'acquisizione è continua, o triggerata con $N_{\mathrm{PKT}} = 0$, la scansione delle finestre si ripete ininterrottamente, fino all'arrivo di un impulso su Stop. Viceversa, se l'acquisizione è triggerata con $N_{\mathrm{PKT}} \neq 0$, ogni volta che il WCNT raggiunge N_{PKT} il Timer Controller torna in TRIGWAIT, sospendendo l'acquisizione.

Riguardo al Time Stamp, per gestire correttamente la numerazione progressiva dei pacchetti inviati al DSP, il Timer Controller lo mantiene in reset soltanto negli stati IDLE, LOAD e READY.

Inoltre, in tali stati, anche i led connessi alle porte DPM_GREENLEDn_O e DPM_REDLEDn_O sono mantenuti spenti. Viceversa, nello stato RUNNING il Timer Controller accende il led verde, in TRIGWAIT il rosso, in DELAYING entrambi.

3.6 Funzionamento del Write Selector (WRS)

livello gerarchico: top/DPM/DCM/WRS

Il diagramma degli stati del Write Selector è rappresentato in Fig. 3.9.

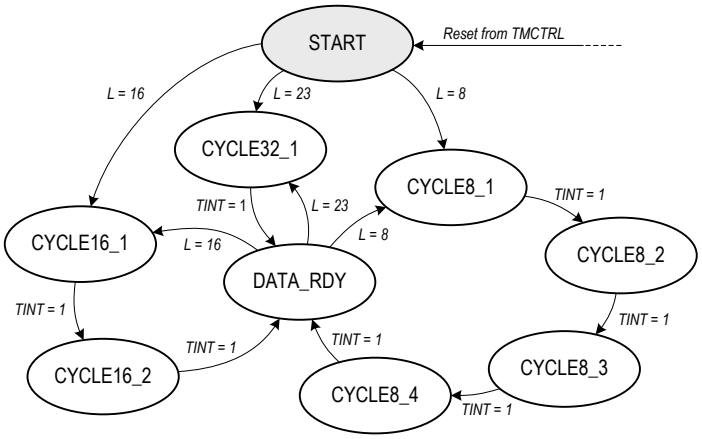


Fig. 3.9 Diagramma degli stati del Write Selector (WRS)

Lo stato di reset è START, ed il segnale di reset proviene direttamente dal Timer Controller. In tale stato, il Write Selector mantiene la propria uscita REGEN disattivata, in modo che i registri di uscita dei vari ChM non siano abilitati alla scrittura.

Non appena il Timer Controller entra in RUNNING, il reset viene rimosso e la macchina, a seconda del particolare valore di $L = 23$, 16, 8 scelto dall'utente, si pone rispettivamente in CYCLE32_1, CYCLE16_1 o CYCLE8_1.

Come si può vedere dal diagramma di Fig. 3.9, ciascuno di tali stati rappresenta l'inizio di un diverso ciclo di funzionamento: il primo è formato da

CYCLE32_1 e DATA_RDY, il secondo da CYCLE16_1, CYCLE16_2 e DATA_RDY, ed infine il terzo da CYCLE8_1, CYCLE8_2, CYCLE8_3, CYCLE8_4 e DATA_RDY.

In particolare, ciascun ciclo consente di pilotare i formatter dei vari ChM secondo una delle tre sequenze di operazioni descritte nel Paragrafo 3.3, così da poter scrivere nei registri di uscita a 32 b rispettivamente 1, 2 o 4 risultati di conteggio consecutivi. Inoltre, alla fine di ogni ciclo, viene inviato un impulso di interrupt al DSP per segnalare la presenza di un nuovo pacchetto.

Si consideri ad esempio il primo ciclo. In CYCLE32_1, il Write Selector, tramite i segnali WMOD, impone ai formatter di essere trasparenti rispetto al risultato a 23 b dei contatori, e di completare la parola di uscita a 32 b aggiungendo nove zeri nelle posizioni più significative. Inoltre, lo stesso Write Selector collega la propria uscita REGEN all'ingresso TINT, cosicché il successivo impulso di fine finestra proveniente dal Timer Controller raggiunga anche i registri di uscita, abilitandoli alla scrittura.

In seguito all'arrivo di tale impulso, la macchina entra nello stato DATA_RDY. REGEN viene disattivata, e viene inviato un impulso di interrupt sull'uscita DPM_RDY_O. Al clock successivo il ciclo ricomincia.

Considerando invece il secondo ciclo, in CYCLE16_1 il Write Selector impone ai formatter di memorizzare, allo scadere della finestra di integrazione corrente, il risultato di conteggio espresso su 16 b. REGEN è mantenuta disattivata, e all'arrivo dell'impulso su TINT, la macchina si porta nello stato CYCLE16_2.

In CYCLE16_2, il Write Selector impone ai formatter di comporre la parola di uscita utilizzando sia i 16 b memorizzati allo scadere della precedente finestra, sia i 16 b provenienti dai contatori. Come avviene in CYCLE32_1, l'uscita REGEN è collegata all'ingresso TINT, cosicché il successivo impulso di fine finestra abiliti alla scrittura i registri di uscita dei vari ChM. Infine, in seguito all'arrivo di tale impulso, la macchina entra in DATA_RDY, il DSP viene interrotto, e al clock successivo il ciclo ricomincia.

Il terzo ciclo è ovviamente analogo agli altri due. In CYCLE8_1, CYCLE8_2 e CYCLE8_3, il Write Selector controlla i formatter in modo che memorizzino, allo scadere di ciascuna finestra, il risultato di conteggio espresso su 8 b. Infine, in CYCLE8_4, i formatter compongono la parola di uscita utilizzando i tre risultati precedentemente memorizzati e gli 8 b provenienti dal contatore. REGEN viene connesso a TINT, e all'arrivo dell'impulso di fine finestra la macchina entra in DATA_RDY. Il DSP viene interrotto e il ciclo ricomincia.

Capitolo 4. Il firmware del DSP

4.1 Compiti del DSP

Come descritto nel Paragrafo 1.5, la comunicazione tra server Linux e DSP avviene per mezzo del link asincrono IEEE 1394 che, a bordo della scheda di sviluppo, è gestito dalla µController IF dell'LLC. In particolare, le informazioni che il server invia al DSP si possono così riassumere:

- parametri di acquisizione;
- comandi di start/stop;
- impostazioni dei sensori;
- parametri della sinusoide.

A sua volta, il DSP utilizza tali informazioni per svolgere i seguenti compiti:

- impostare il DPM (per le operazioni di acquisizione e conteggio) e il DMPORT (per la trasmissione dei pacchetti sul link isocrono);
- avviare o arrestare l'attività del DPM;
- impostare i sensori tramite la periferica UART;
- generare i campioni della sinusoide da inviare alla McASP.

Inoltre, dopo aver avviato l'acquisizione, il DSP deve ovviamente:

- ricevere i pacchetti provenienti dal DPM;
- elaborarli in tempo reale;
- inviare periodicamente al server Linux, per mezzo del link asincrono, i risultati ottenuti durante l'elaborazione.

In particolare, dalla descrizione svolta nel Capitolo 3, appare evidente che, per come è stata progettata l'architettura dell'FPGA, non è possibile trasferire un pacchetto di risultati dal DPM al DMPORT in modo diretto.

Infatti, sia il DPM che il DMPORT possono comunicare esclusivamente con il DSP, e non tra loro. Di conseguenza, il DSP ha il compito non solo di ricevere ed elaborare i pacchetti provenienti dal DPM, ma anche di rinviarli al DMPORT per consentire la loro trasmissione sul link isocrono.

È ovvio che, così facendo, il bus EMIF viene impegnato il doppio del necessario. Infatti, ogni pacchetto viene trasferito tra FPGA e DSP per ben due volte, prima in un senso e poi nell'altro.

Una simile soluzione può apparire estremamente penalizzante dal punto di vista delle prestazioni, e in particolar modo rispetto al valore di T_{Wmin}, dal momento che il DSP, prima di ricevere nuovi pacchetti dal DPM, non solo deve avere elaborato i precedenti, ma deve averli anche rinviati al DMPORT.

Tuttavia, come si vedrà in seguito, il rinvio avviene in background rispetto all'elaborazione, ossia contemporaneamente, in modo indipendente, e senza utilizzare le risorse di calcolo ad essa riservate. Inoltre, i tempi di elaborazione sono molto più lunghi rispetto a quelli di trasferimento, e dunque il rinvio dei pacchetti al DMPORT, pur impegnando il bus EMIF più del dovuto, non penalizza in alcun modo le prestazioni del sistema.

Viceversa, se in futuro si deciderà di sviluppare ulteriormente il contatore multicanale consentendo all'utente di poter disattivare l'elaborazione dei pacchetti in tempo reale, per ottimizzare le prestazioni del sistema sarà sicuramente necessario risolvere il problema del rinvio.

A tal proposito, una semplice soluzione può essere quella di modificare la rete combinatoria rappresentata nell'area ombreggiata di Fig. 3.1 e di Tav. A.1, in modo che, quando il DSP vuole leggere una qualunque locazione all'interno del Channel Space,

- sul bus indirizzi del DMPORT sia forzato l'indirizzo relativo al buffer FIFO di trasmissione;
- i segnali di abilitazione alla lettura provenienti dal bus EMIF e destinati al DPM raggiungano anche il DMPORT, abilitandolo invece alla scrittura.

Infatti, si osservi in Tav. A.1 che, quando il DSP accede al DPM in lettura, il multiplexer è in posizione 0 e la porta tristate è in posizione 1, e dunque i dati di uscita del DPM si trovano già in ingresso al DMPORT. Di conseguenza, modificando tale rete nel modo descritto, ogni volta che il DSP riceve un pacchetto dal DPM, tale pacchetto verrebbe direttamente trasferito anche nel buffer di trasmissione del DMPORT e dunque inviato sul link isocrono.

Ovviamente, perché tutto funzioni, il DSP dovrebbe continuare a svolgere le operazioni di lettura dei pacchetti anche nel caso in cui l'utente abbia disattivato l'elaborazione in tempo reale, cosa che, comunque, non costituirebbe alcun problema.

Infine, è bene precisare che i valori delle minime finestre di integrazione riportati nel seguito e relativi al funzionamento del contatore in assenza di elaborazione in tempo reale, sono stati misurati senza aver posto alcun rimedio al problema del rinvio dei pacchetti. Viceversa, implementando una soluzione come quella appena proposta, tali valori, peraltro già comparabili con quelli dei migliori contatori multicanale presenti attualmente sul mercato, possono, in teoria, addirittura dimezzare.

Riguardo alla gestione dei sensori, il DSP, in base ai parametri di impostazione ricevuti dal server, ha il compito di preparare un'opportuna stringa di caratteri e di inviarla alla periferica UART. In particolare, dal momento che il server può inviare tali parametri in un qualunque istante, il DSP deve essere in grado di gestire i sensori anche durante l'elaborazione in tempo reale dei pacchetti.

Riguardo invece alla generazione dei campioni della sinusoide, come accennato nel Paragrafo 1.5, in passato sono state separatamente sviluppate e testate con successo delle opportune routine, che tuttavia, a causa della loro complessità, non sono state ancora integrate nell'applicazione principale.

In particolare, tali routine consentivano all'utente di impostare, in un qualunque istante, tutti i parametri della sinusoide e, in base ad essi, generavano continuamente i campioni da inviare alla McASP. Di conseguenza, necessitavano di essere mantenute sempre in esecuzione, anche durante l'elaborazione dei pacchetti, sottraendo inevitabilmente a quest'ultima un discreto quantitativo di risorse di calcolo.

Viceversa, una soluzione più semplice, che non sottrae risorse all'elaborazione, ma che tuttavia non consente di variare i parametri della sinusoide in un qualunque istante, può essere quella di preparare, prima di avviare l'acquisizione, un'opportuna sequenza di campioni da scansionare ciclicamente.

Si osservi infatti che, detti A_0, f_0 e φ_0 rispettivamente l'ampiezza, la frequenza e la fase della sinusoide, ed f_S la frequenza di campionamento, il k-esimo campione ξ_k calcolato all'istante k/f_S è dato ovviamente da

$$\xi_k = A_0 \sin\left(2\pi f_0 \frac{k}{f_S} + \varphi_0\right) \qquad k = 0,\ 1,\ 2\ldots \tag{4.1}$$

Avendo scelto per f_S il valore standard di 48 kHz utilizzato nei sistemi audio, e dato che f_0, variando tra 1.5 kHz e 3 kHz con passi di 1 Hz (Paragrafo 1.4), può assumere solo valori interi, dalla (4.1) si può osservare che i campioni della sinusoide si ripetono sicuramente ogni 48000, qualunque siano i valori degli altri due parametri.

Dunque è possibile preparare una sequenza di 48000 – 1 campioni, e memorizzarla nella SRAM da 64 MB di cui la scheda Orsys dispone. In particolare, tale memoria, pur non essendo rappresentata in Fig. 1.4, risulta connessa al bus EMIF come tutte le altre periferiche esterne del DSP.

Così facendo, la sequenza ottenuta può essere scansionata ciclicamente in background, trasferendo automaticamente un campione dopo l'altro dalla memoria alla periferica McASP, senza sottrarre risorse all'elaborazione.

Infine si osservi che, almeno per le applicazioni astronomiche, il fatto che una soluzione simile non consenta di variare i parametri della sinusoide durante l'acquisizione, non rappresenta comunque un problema.

4.2 La comunicazione tra DSP e LLC

In generale, ogni volta che l'LLC vuole interagire con il DSP, invia una richiesta di interrupt sulla linea EXT_INT6 (vedi Fig. 1.4).

Ad esempio, l'LLC può interrompere il DSP dopo aver ricevuto in ingresso un nuovo dato sul link asincrono, oppure dopo aver completato, sullo stesso link, una transazione in uscita. Nel primo caso, l'interruzione consente di segnalare al DSP la presenza di un nuovo dato all'interno dei registri della µController IF, pronto per essere letto, mentre, nel secondo, di comunicare al DSP l'esito dell'ultima transazione asincrona in uscita.

Inoltre, ogni volta che il DSP riceve un segnale di interrupt, il suo compito è quello di sospendere immediatamente l'esecuzione principale e di avviare la corrispondente *Interrupt Service Routine* (ISR). Successivamente, solo dopo aver portato a termine l'ISR, il DSP può riprendere l'esecuzione dal punto in cui era stata precedentemente interrotta. Ovviamente, quanto più complesse sono le varie ISR, tanto maggiore sarà il tempo in cui l'esecuzione principale rimane sospesa.

Affinché un sistema real-time funzioni correttamente, è necessario che i tempi di elaborazione siano contenuti entro certi limiti ben precisi. Ad esempio, nel nostro caso è necessario che il DSP elabori ciascun pacchetto prima che il successivo sia pronto. Ovviamente, se durante l'elaborazione il DSP riceve una o più richieste di interruzione, la durata delle ISR può essere tale da impedire a quest'ultimo di rispettare la scadenza temporale (*deadline*) rappresentata dall'istante di arrivo del pacchetto successivo.

Si osservi in particolare che, per evitare tale problema, non sempre è possibile disabilitare le interruzioni. Infatti, nel caso ad esempio dell'LLC, una scelta del genere renderebbe impossibile la comunicazione tra DSP e server, causando il blocco delle transazioni asincrone sul bus IEEE 1394. Di conseguenza, è estremamente necessario rendere le ISR le più semplici e veloci possibile.

A tal fine, per gestire le richieste di interruzione provenienti dall'LLC si è scelto di utilizzare un meccanismo basato su una coda di *callback*.

All'interno del programma principale del DSP è stato definito un opportuno set di funzioni, ciascuna delle quali svolge una delle possibili operazioni necessarie a soddisfare le richieste di interrupt provenienti dall'LLC.

Una coda di callback è semplicemente una coda di puntatori a funzione. Quando arriva un interrupt dall'LLC, il DSP avvia un'apposita ISR che legge il codice di interruzione e, in base ad esso, inserisce nella coda di callback un puntatore a quella particolare funzione del set precedente, capace di eseguire l'operazione richiesta dall'LLC.

Successivamente, tale funzione non viene chiamata, e la ISR ritorna immediatamente, consentendo al DSP di riprendere l'esecuzione dal punto in cui era stata precedentemente interrotta.

Di conseguenza, il programma principale ha il compito di controllare periodicamente la coda di callback e, nel caso in cui non sia vuota, di estrarre il primo puntatore e di chiamare la funzione da esso puntata.

Ovviamente, il vantaggio di un simile meccanismo è quello sia di poter utilizzare una ISR semplice e veloce, in modo da sospendere il meno possibile l'esecuzione principale, sia di poter rimandare la chiamata delle funzioni di comunicazione verso l'LLC a istanti più favorevoli e, soprattutto, noti a priori.

Tuttavia, è necessario che il programma principale sia tale da garantire che il controllo della coda di callback si ripeta con una certa regolarità, altrimenti il meccanismo descritto fallirebbe, con le stesse conseguenze che si avrebbero disabilitando le interruzioni.

4.3 Pacchetti e risultati di elaborazione: il trasferimento tra FPGA, DSP e LLC

Il trasferimento dei pacchetti tra DPM e DSP avviene in background, grazie all'*Enhanced DMA* (EDMA) di cui il DSP dispone.

In particolare, si è scelto di utilizzare una tecnica di *Ping-Pong buffering*. Nella memoria interna del DSP sono stati definiti due buffer (Ping e Pong), capaci di contenere M pacchetti ciascuno. L'idea è quella di fare in modo che, mentre un buffer sta ricevendo i nuovi pacchetti dal DPM, l'altro venga elaborato dal DSP, e viceversa.

A tal fine, le richieste di interruzione provenienti dal DPM raggiungono direttamente l'EDMA, che, a sua volta, provvede a trasferire i nuovi pacchetti all'interno di uno dei due buffer.

In particolare, dopo aver riempito il buffer Ping, l'EDMA comincia automaticamente a riempire Pong, e quando anche Pong è stato riempito, il procedimento ricomincia da capo, sovrascrivendo i contenuti precedenti. Tale procedimento si ripete continuamente, fino all'arresto dell'acquisizione.

Inoltre, l'EDMA, ogni volta che uno dei due buffer è stato riempito, interrompe immediatamente il DSP. La relativa ISR determina quale dei due buffer è pronto per essere elaborato e, dei due flag appositamente predisposti per contenere l'una o l'altra delle due informazioni, attiva il corrispondente.

Di conseguenza, analogamente a quanto avviene per la gestione della coda di callback, il programma principale ha il compito di controllare periodicamente tali flag e, nel caso ve ne sia uno attivo, di resettarlo, di cominciare

l'elaborazione del buffer corrispondente e, al tempo stesso, di avviare le operazioni di rinvio in background dei suoi pacchetti al DMPORT.

Il rinvio dei pacchetti dal DSP al DMPORT avviene grazie al *Quick DMA* (QDMA). Quest'ultimo, in particolare, non è un ulteriore dispositivo di cui il DSP dispone, ma è semplicemente una parte dello stesso EDMA.

In generale, l'EDMA può gestire in parallelo molti trasferimenti, anche complessi, ciascuno dei quali è avviato da un opportuno evento di trigger. Ad esempio, nel caso del riempimento dei due buffer, l'evento di trigger coincide con la richiesta di interruzione proveniente dal DPM.

I parametri relativi a ciascun trasferimento (quali ad esempio l'indirizzo sorgente, l'indirizzo destinatario, il numero di parole da trasferire, la priorità, l'evento di trigger associato, etc.) sono contenuti nella memoria interna dell'EDMA, in appositi *descrittori*.

Ovviamente, il programmatore, se desidera utilizzare l'EDMA per un qualunque trasferimento, anche semplice, deve comunque specificare completamente uno di tali descrittori, definendone tutti i parametri. Inoltre, se desidera che l'avvio di tale trasferimento sia controllato dal DSP, deve generare, quando lo ritiene opportuno, un apposito evento di trigger da inviare all'EDMA.

Tutte queste operazioni, a seconda del caso, possono risultare troppo costose in termini di tempo. Per tal motivo, all'interno dell'EDMA esiste un particolare descrittore, il QDMA, appositamente pensato per eseguire trasferimenti semplici e immediati. Infatti, i parametri di cui il QDMA necessita sono molto pochi rispetto a quelli di un qualunque altro descrittore e inoltre, dopo aver specificato l'ultimo di essi, il trasferimento inizia immediatamente, senza la necessità di generare un apposito evento di trigger.

In particolare, per rinviare i pacchetti al DMPORT, il DSP deve comunicare al QDMA solamente:

- l'indirizzo sorgente, ossia quello della prima locazione del buffer da rinviare;
- l'indirizzo destinazione, ossia quello del buffer di trasmissione del DMPORT;
- il numero di volte che l'indirizzo sorgente deve essere incrementato, ossia $N_C \cdot M - 1$.

Appena comunicati tali dati, il DSP comincia l'elaborazione del buffer, e il QDMA provvede contemporaneamente al rinvio.

Riguardo invece all'elaborazione dei pacchetti, i risultati ottenuti di volta in volta per ciascun canale sono memorizzati in un apposito Output Buffer, allocato nella memoria interna del DSP e contenente N_C parole a 32 b. In particolare, quando il DSP ha elaborato i buffer Ping e Pong per un certo numero di volte dettato dalla particolare applicazione real-time implementata, l'Output

Buffer viene trasferito alla µController IF, per essere così trasmesso al server Linux sul link asincrono. In particolare, anche tale trasferimento avviene in background, sempre grazie al QDMA.

In Fig. 4.1 sono dunque riassunti sia i trasferimenti che coinvolgono i soli pacchetti, sia quelli relativi ai risultati dell'elaborazione.

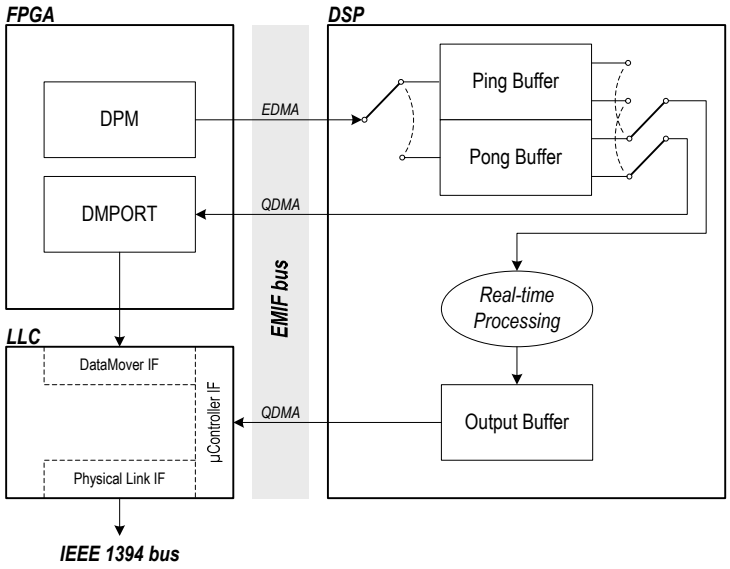


Fig. 4.1 Il trasferimento dei pacchetti e dei risultati di elaborazione tra FPGA, DSP ed LLC

Ovviamente, dal momento che il bus EMIF è unico, tali trasferimenti avverranno in modo interlacciato.

In particolare, si è scelto di rendere i trasferimenti gestiti dall'EDMA fortemente prioritari sia rispetto a quelli gestiti dal QDMA, sia rispetto ad ogni altro trasferimento che coinvolge il bus EMIF.

In questo modo, si è cercato di garantire il corretto riempimento dei buffer Ping e Pong, evitando che – a causa del traffico sul bus dovuto al rinvio dei pacchetti, al trasferimento dell'Output Buffer, o all'invio delle impostazioni dei sensori da parte del server – alcuni pacchetti andassero persi.

4.4 Il ruolo del Ping-Pong buffering nell'ottimizzazione delle prestazioni

Al fine di evidenziare i vantaggi introdotti dall'utilizzo della tecnica di Ping-Pong buffering, è necessario fare alcune considerazioni sui tempi richiesti dal DSP per svolgere le varie attività.

Si indichi dunque con $T_{\mathrm{PKT}} = T_{\mathrm{W}} \cdot F$ il tempo di vita di ciascun pacchetto generato dal DPM, e con T_{PRC} il tempo necessario al DSP per elaborarlo. Ovviamente, sia T_{PKT} che T_{PRC} dipendono dal formato scelto.

Affinché il DSP sia in grado di elaborare in tempo reale i pacchetti provenienti dal DPM, indipendentemente da qualunque tecnica di buffering eventualmente utilizzata deve essere verificata la relazione

$$T_{\mathrm{PRC}} \leq T_{\mathrm{PKT}} \,. \tag{4.2}$$

Tuttavia, non è detto che tale relazione sia sufficiente.

Infatti, il DSP,

- prima di iniziare l'elaborazione di un buffer, deve gestire la richiesta di interruzione proveniente dall'EDMA;
- durante l'elaborazione, può essere interrotto più volte dall'LLC per inserire un puntatore nella coda di callback;
- alla fine dell'elaborazione, prima di poter cominciare la successiva, deve controllare la coda di callback e, se non è vuota, estrarre il primo puntatore ed eseguire la funzione da esso puntata.

Per gestire la richiesta di interruzione provenienti dall'EDMA, il DSP spenderà quindi un tempo T_{EDMA}, e per gestire la coda di callback un tempo complessivo τ_{LLC}.

In particolare, si osservi che il valore di τ_{LLC} dipende

- dalla normale attività del DSP, che periodicamente invia i risultati di elaborazione alla µController IF, e quindi al server Linux;
- dalle richieste che l'utente, e quindi il server, invia al DSP durante l'acquisizione, come ad esempio quelle relative alla gestione dei sensori.

Si supponga inizialmente che, durante l'acquisizione, il DSP non riceva alcuna richiesta dal server. Sotto tale ipotesi, nel peggiore dei casi il valore τ^0_{LLC} di τ_{LLC} è dato semplicemente dalla somma tra il tempo necessario a inserire nella coda di callback il puntatore alla funzione di notifica dell'esito dell'ultima transazione sul link asincrono, e il tempo necessario a estrarre tale puntatore ed eseguire la funzione puntata.

In assenza di una qualunque tecnica di buffering, il che equivale a considerare $M = 1$, affinché il sistema funzioni in tempo reale, deve quindi essere

$$T_{\mathrm{EDMA}} + T_{\mathrm{PRC}} + \tau^0_{\mathrm{LLC}} \leq T_{\mathrm{PKT}} \,. \tag{4.3}$$

Viceversa, con la tecnica di Ping-Pong buffering, la relazione (4.3) diviene

$$T_{\mathrm{EDMA}} + M \cdot T_{\mathrm{PRC}} + \tau^0_{\mathrm{LLC}} \leq M \cdot T_{\mathrm{PKT}} \,. \tag{4.4}$$

Se $T_{\mathrm{PRC}} = T_{\mathrm{PKT}}$, le (4.3) e (4.4) sono equivalenti e impossibili: con o senza buffering il sistema non può funzionare in tempo reale.

Supponiamo quindi di essere riusciti a ottenere un T_{PRC} che verifichi la relazione fondamentale

$$T_{\mathrm{PRC}} < T_{\mathrm{PKT}} \,. \tag{4.5}$$

Per far funzionare il sistema senza una tecnica di buffering, è necessario che il T_{PRC} ottenuto verifichi anche la (4.3). Viceversa, utilizzando la tecnica di Ping-Pong buffering, per verificare la (4.4) è sufficiente scegliere

$$M \geq \frac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{T_{\mathrm{PKT}} - T_{\mathrm{PRC}}}. \tag{4.6}$$

Di conseguenza, affinché il sistema funzioni in tempo reale, a seconda del valore di F è necessario che sia verificata una delle seguenti relazioni:

$$\begin{cases} M \geq \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{1 \cdot T_{\mathrm{W}} - T_{\mathrm{PRC32}}} & \text{per } F = 1 \\ M \geq \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{2 \cdot T_{\mathrm{W}} - T_{\mathrm{PRC16}}} & \text{per } F = 2 \\ M \geq \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{4 \cdot T_{\mathrm{W}} - T_{\mathrm{PRC8}}} & \text{per } F = 4 \end{cases} \tag{4.7}$$

dove T_{PRC32}, T_{PRC16} e T_{PRC8} sono i tempi di elaborazione di un pacchetto di N_{C} parole a 32 b contenenti ciascuna rispettivamente 1, 2 o 4 risultati di conteggio consecutivi.

Tuttavia, una volta implementato il firmware del DSP, T_{PRC32}, T_{PRC16}, T_{PRC8}, T_{EDMA} e τ^0_{LLC} sono determinati, mentre invece T_{W} resta ovviamente variabile.

In particolare, per verificare la relazione fondamentale espressa dalla (4.5), a seconda del valore di F deve necessariamente essere:

$$\begin{cases} T_{\mathrm{W}} > \dfrac{T_{\mathrm{PRC32}}}{1} & \text{per } F = 1 \\ T_{\mathrm{W}} > \dfrac{T_{\mathrm{PRC16}}}{2} & \text{per } F = 2 \\ T_{\mathrm{W}} > \dfrac{T_{\mathrm{PRC8}}}{4} & \text{per } F = 4 \end{cases} \tag{4.8}$$

Si osservi quindi che, fissato F e scelto un T_{W} che verifica la (4.8), esiste sempre un valore di M che verifica la (4.7), consentendo il funzionamento del sistema in tempo reale.

Viceversa, fissato F e un qualunque valore di $M > 0$, esiste sempre un valore minimo T_{Wmin} di T_{W} tale che, per ogni $T_{\mathrm{W}} \geq T_{\mathrm{Wmin}}$ sia la (4.7) che la (4.8) sono verificate, e dunque il funzionamento del sistema è garantito.

In particolare, coerentemente con l'analisi generale svolta nel Paragrafo 2.1, per $F = 1$, 2, 4 si indichi tale valore minimo rispettivamente con T_{W32}, T_{W16} e T_{W8}. Dalla (4.7) si ha dunque:

$$\begin{cases} T_{\mathrm{W32}} = \dfrac{T_{\mathrm{PRC32}}}{1} + \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{M} & \text{per } F = 1 \\ T_{\mathrm{W16}} = \dfrac{T_{\mathrm{PRC16}}}{2} + \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{2M} & \text{per } F = 2 \\ T_{\mathrm{W8}} = \dfrac{T_{\mathrm{PRC8}}}{4} + \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{4M} & \text{per } F = 4 \end{cases} \tag{4.9}$$

Il grande vantaggio che si ha utilizzando la tecnica di Ping-Pong buffering risulta dunque evidente: quanto più M è grande, tanto più le minime finestre di integrazione diminuiscono, avvicinandosi ai rispettivi limiti teorici imposti dalla (4.8).

Inoltre, si osservi che la formattazione dei risultati di conteggio è vantaggiosa solo quando consente di ottenere $T_{\mathrm{W8}} < T_{\mathrm{W16}} < T_{\mathrm{W32}}$. A tal fine, utilizzando la (4.9) si ottiene facilmente che:

$$\begin{cases} T_{\mathrm{W16}} < T_{\mathrm{W32}} \quad \Leftrightarrow \quad T_{\mathrm{PRC16}} < 2 \cdot T_{\mathrm{PRC32}} + \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{M} \\ T_{\mathrm{W8}} < T_{\mathrm{W16}} \quad \Leftrightarrow \quad T_{\mathrm{PRC8}} < 2 \cdot T_{\mathrm{PRC16}} + \dfrac{T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}}{M} \end{cases} \tag{4.10}$$

In particolare, tenuto conto della (4.10), la formattazione è vantaggiosa per ogni M se e solo se i tempi di elaborazione verificano le condizioni:

$$\begin{cases} T_{\mathrm{PRC16}} \leq 2 \cdot T_{\mathrm{PRC32}} \\ T_{\mathrm{PRC8}} \leq 2 \cdot T_{\mathrm{PRC16}} \end{cases} \tag{4.11}$$

Come già accennato, in ogni caso si ha $T_{\mathrm{PRC32}} < T_{\mathrm{PRC16}} < T_{\mathrm{PRC8}}$.

In fase di ottimizzazione del firmware, lo scopo da raggiungere è quello di ridurre il più possibile le minime finestre di integrazione e di far sì che la formattazione risulti vantaggiosa. A tal fine, si deve dunque:

- minimizzare i tempi di elaborazione;
- fare in modo che tali tempi verifichino la (4.11);
- utilizzare il più grande M consentito dalle risorse hardware.

Nel nostro caso, in particolare, tali risorse sono rappresentate dai 256 kB della memoria interna del DSP, nella quale, per consentire un'elaborazione veloce, devono trovare spazio non solo Ping e Pong, ma anche il codice, l'Output Buffer e tutti gli altri dati.

Infine, per $F = 1, 2, 4$, è possibile determinare il massimo bit-rate di ingresso sostenibile dal DSP in tempo reale, indicato – come nell'analisi svolta nel Paragrafo 2.1 – rispettivamente con B_{32}, B_{16} e B_8. In generale, tale bit-rate è dato

dal numero di bit contenuti all'interno di ciascun buffer, diviso per il massimo tempo necessario a completarne l'elaborazione. Si ha quindi:

$$\begin{cases} B_{32} = \dfrac{M \cdot N_{\mathrm{C}} \cdot 32}{T_{\mathrm{EDMA}} + M \cdot T_{\mathrm{PRC32}} + \tau^0_{\mathrm{LLC}}} & \text{per } F = 1 \\ B_{16} = \dfrac{M \cdot N_{\mathrm{C}} \cdot 32}{T_{\mathrm{EDMA}} + M \cdot T_{\mathrm{PRC16}} + \tau^0_{\mathrm{LLC}}} & \text{per } F = 2 \\ B_{8} = \dfrac{M \cdot N_{\mathrm{C}} \cdot 32}{T_{\mathrm{EDMA}} + M \cdot T_{\mathrm{PRC8}} + \tau^0_{\mathrm{LLC}}} & \text{per } F = 4 \end{cases} \tag{4.12}$$

Ricavando dalla (4.12) le espressioni di T_{PRC32}, T_{PRC16} e T_{PRC8} in funzione dei rispettivi bit-rate, e sostituendo tali espressioni nella (4.10), si verifica che

$$\begin{cases} T_{\mathrm{W16}} < T_{\mathrm{W32}} & \Leftrightarrow \quad B_{32} < 2 \cdot B_{16} \\ T_{\mathrm{W8}} < T_{\mathrm{W16}} & \Leftrightarrow \quad B_{16} < 2 \cdot B_{8} \end{cases} \tag{4.13}$$

Di conseguenza, le condizioni riassunte nella (4.10), per le quali la formattazione risulta vantaggiosa, equivalgono a quelle determinate nell'analisi generale svolta nel Paragrafo 2.1.

Sin qui si è supposto che, durante l'acquisizione, il DSP non riceva alcuna richiesta dal server, e dunque, sotto tale ipotesi, nelle formule precedenti si è scelto di utilizzare il valore τ^0_{LLC} di τ_{LLC}.

Tuttavia, affinché il sistema sia effettivamente utilizzabile, è necessario consentire al server Linux di comunicare con il DSP anche durante l'acquisizione. Per tal motivo, al fine di prevedere correttamente le prestazioni del sistema, sarebbe necessario calcolare una nuova stima di τ_{LLC} tenendo conto anche delle richieste inviate dal server.

Si osservi che la frequenza delle richieste inviate dal server non può essere troppo elevata. Infatti, per evitare malfunzionamenti, e per consentire al DSP di soddisfare ciascuna richiesta in tempi brevi, è necessario garantire che la coda di callback si mantenga la più corta possibile.

A tal fine, dato che il DSP riceve periodicamente una richiesta dallo stesso LLC ogni volta che ha completato l'invio dell'Output Buffer, un valore ragionevole di tale frequenza può essere quello di 1 richiesta ogni $2 \cdot M \cdot T_{\mathrm{PRC}}$ secondi.

In tal caso, le nuove prestazioni del sistema si valutano con un nuovo τ^1_{LLC} dato dalla somma tra il tempo necessario a inserire due puntatori nella coda, e il tempo necessario ad estrarre un puntatore ed eseguire la funzione che richiede il maggior tempo di esecuzione.

Ad ogni modo, τ^1_{LLC} sarà molto vicino a τ^0_{LLC}. Di conseguenza, continuando a utilizzare τ^0_{LLC}, l'errore che si commetterà nella valutazione delle prestazioni fornite dalla (4.9) e dalla (4.12), sarà trascurabile.

Infine è bene osservare che, come avverrebbe utilizzando una qualunque altra tecnica di buffering, al crescere di M aumenta la latenza del sistema.

4.5 L'algoritmo real-time

Come accennato nel Paragrafo 1.5, l'algoritmo real-time accumula periodicamente, per ciascun canale, i risultati di conteggio provenienti dal DPM. I dati di volta in volta ottenuti vengono memorizzati dal DSP all'interno dell'Output Buffer.

Dopo aver accumulato i risultati contenuti in $P{\cdot}M$ pacchetti consecutivi, ossia dopo aver elaborato i buffer Ping e Pong complessivamente per P volte, il DSP invia il contenuto dell'Output Buffer al server Linux, sul link asincrono. Successivamente resetta tale buffer e ricomincia l'accumulo da capo.

Così facendo, è come se il DSP fornisse al server i risultati di conteggio relativi ad una nuova finestra di integrazione ampia T_{ACC} secondi, con

$$T_{\mathrm{ACC}} = P \cdot M \cdot T_{\mathrm{PKT}} = P \cdot M \cdot F \cdot T_{\mathrm{W}} \,. \tag{4.14}$$

In particolare, una volta che l'utente ha fissato T_{W} ed L (e dunque F), il DSP determina automaticamente P in modo da avere per T_{ACC} il più piccolo valore maggiore di 100 ms, compatibile con la (4.14).

Si osservi infine che la complessità dell'algoritmo, espressa in numero di istruzioni elementari, varia ovviamente a seconda del formato selezionato.

Infatti, se $L = 23$ (ossia $F = 1$), dal momento che ogni parola a 32 b contiene un unico risultato, è sufficiente sommare la i-esima parola di ciascun pacchetto alla i-esima parola dell'Output Buffer.

Viceversa, se $L = 16,\ 8$ (ossia $F = 2,\ 4$), per la i-esima parola di ciascun pacchetto si dovranno sommare, alla i-esima parola dell'Output Buffer, rispettivamente 2 o 4 risultati e, per ciascuna somma, saranno necessarie alcune operazioni di mascheramento e di shift a destra.

In definitiva, si può dimostrare che per ciascuna parola a 32 b proveniente dal DPM sono necessarie $3F - 2$ operazioni elementari, oppure, in modo ovviamente analogo, che per ciascun pacchetto ne sono necessarie $N_{\mathrm{C}}(3F - 2)$.

4.6 Il diagramma di flusso

Il diagramma di flusso del programma principale implementato sul DSP è rappresentato in Fig. 4.2.

Nella fase iniziale, il DSP imposta le sue periferiche: dapprima quelle interne, quali EMIF, EDMA e McASP (anche se ancora non è utilizzata); successivamente, grazie al bus EMIF, quelle esterne, quali LLC e UART.

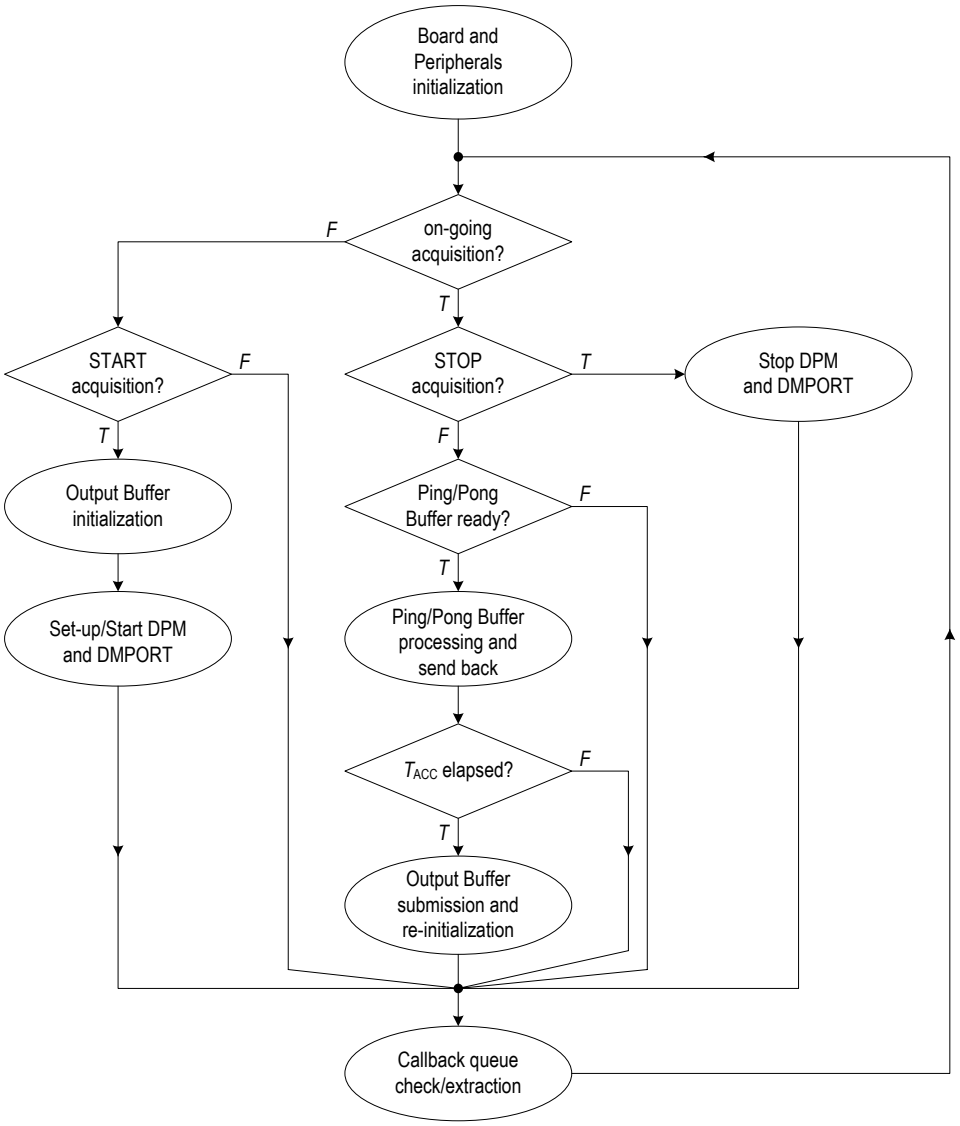


Fig. 4.2 Il diagramma di flusso del programma principale implementato sul DSP

Dopo la fase di inizializzazione, comincia il ciclo infinito. In particolare, si osservi che, qualunque sia il ramo percorso, l'ultima fase del ciclo consiste sempre nel controllo della coda di callback e nella chiamata di eventuali funzioni di comunicazione verso l'LLC rimaste in attesa.

Per prima cosa, il DSP controlla se è in corso un'acquisizione. Se non lo è (ramo di sinistra), controlla se dal server è arrivata una richiesta di start. In tal caso, si predispone all'accumulo azzerando l'Output Buffer; successivamente, imposta il DPM e il DMPORT utilizzando i parametri di acquisizione ricevuti dall'utente e, infine, avvia le loro attività.

Viceversa, se è in corso un'acquisizione, il DSP controlla se è arrivata una richiesta di stop dal server: in tal caso (ramo di destra) provvede ad arrestare le operazioni del DPM e del DMPORT, altrimenti (ramo centrale) controlla se uno dei due buffer Ping e Pong è pronto per essere elaborato.

Se così è, il DSP fa partire il rinvio in background dei pacchetti al DMPORT e comincia l'elaborazione, memorizzando di volta in volta i risultati ottenuti all'interno dell'Output Buffer.

Infine, dopo aver completato l'elaborazione, il DSP controlla se è trascorso il tempo T_{ACC}, ossia se, a partire dall'ultimo invio dell'Output Buffer al server Linux, Ping e Pong sono stati elaborati complessivamente P volte. In caso affermativo, provvede immediatamente a inviare l'Output Buffer e, successivamente, a riazzerarlo, così da essere pronto a ricominciare in seguito le operazioni di accumulo.

Capitolo 5. Implementazione, test e conclusioni

5.1 Sintesi e implementazione dell'architettura dell'FPGA

L'architettura dell'FPGA è stata descritta interamente in VHDL, rendendo N_C il parametro principale. Per lo sviluppo del codice e per i test funzionali si è utilizzato l'ambiente Aldec ActiveHDL 6.2 [21], mentre per la sintesi e per l'implementazione, l'ambiente Xilinx ISE Foundation 8.1i [17].

In particolare, riguardo a sintesi ed implementazione, gli obiettivi da raggiungere sono stati così prefissati:

- massimizzare il numero di canali N_C, utilizzando al meglio le risorse logiche disponibili;
- fare in modo che i tempi di propagazione dei segnali all'interno dell'FPGA consentano rispettivamente
  - il funzionamento dell'array alla frequenza di clock di 90 MHz;
  - l'utilizzo delle più brevi temporizzazioni EMIF, sia in lettura che in scrittura, così da velocizzare la comunicazione tra FPGA e DSP.

Come si può notare, per raggiungere tali obiettivi è necessario ottimizzare sia l'utilizzo delle risorse logiche disponibili (*area goal*), sia la velocità delle connessioni interne (*speed goal*).

Tuttavia, ISE, così come ogni altro software di sintesi e di implementazione, mette a disposizione del progettista alcuni potenti algoritmi finalizzati esclusivamente all'ottimizzazione o dell'utilizzo delle risorse logiche disponibili, o della velocità delle connessioni interne.

Di conseguenza, dal momento che è preferibile ottenere un counter array ad alte prestazioni e con un N_C ridotto piuttosto che il contrario, si è scelto di utilizzare gli algoritmi che privilegiano la velocità delle connessioni, così da non penalizzare né la massima frequenza di clock (e quindi f_{INmax}), né i tempi necessari allo scambio dei dati tra FPGA e DSP.

Dopo diversi tentativi, è stato possibile determinare i settaggi ottimali per gli algoritmi di sintesi e di implementazione, riuscendo così ad ottenere i seguenti risultati:

- $N_C = 64$, occupando il 97% delle risorse logiche disponibili;
- massima frequenza di clock pari a 100 MHz;
- possibilità di utilizzare, sia per le operazioni di lettura che per quelle di scrittura, una temporizzazione EMIF che dura complessivamente 5 periodi di clock.

Per la fase di sintesi è stato utilizzato il più efficiente algoritmo di ottimizzazione della velocità messo a disposizione da ISE. In particolare, la scelta determinante è stata quella di attivare la funzione di *Register Balancing*.

Tale funzione consente all'algoritmo di sintesi di ottimizzare le reti combinatorie in modo da uniformare il più possibile i tempi di propagazione tra i vari registri. Così facendo, è possibile ridurre i cammini critici, rendendo possibile l'utilizzo di una frequenza di clock più elevata.

Riguardo invece alla successiva implementazione, ISE suddivide le principali operazioni da compiere in tre fasi consecutive, o processi: *Translate*, *Map* e *Place and Route*.

- Translate ha il compito di tradurre la rete logica prodotta dall'algoritmo di sintesi in una rete equivalente, composta esclusivamente da quelle primitive che gli algoritmi dei processi successivi sono in grado di riconoscere;
- Map, a partire dalle primitive della rete tradotta, ha il compito di produrre una nuova rete logica, mappando tali primitive all'interno di blocchi logici (CLB e IOB) analoghi a quelli effettivamente disponibili nell'FPGA;
- Place and Route ha il compito di assegnare a ciascun CLB e a ciascun IOB della rete mappata un preciso CLB e un preciso IOB presente all'interno dell'FPGA (*place*) e, successivamente, di determinare i collegamenti necessari tra di essi (*route*) per ottenere le funzionalità logiche richieste.

Per ottimizzare i processi di Map e Place and Route, l'utente può indicare alcuni limiti ben precisi (*constraints*) che il sistema, una volta implementato, deve rispettare per soddisfare le specifiche. In particolare, tra tutti i constraints utilizzati, i più significativi sono stati *Clock Period*, *Offset In Before*, *Offset Out After*, *Area Group* e *Use Low-Skew Lines*.

- Clock Period consente di specificare il minimo periodo di clock per il quale deve essere garantito il funzionamento del sistema;
- Offset In Before consente di specificare il massimo ritardo tollerabile nella propagazione dei segnali del bus EMIF, dai pin fino all'ingresso dei registri di interfaccia, in modo da garantire il rispetto della temporizzazione di scrittura;
- Offset Out After, analogamente al precedente, consente di specificare il massimo ritardo tollerabile nella propagazione dei segnali di uscita dei registri di interfaccia verso i pin del bus EMIF, in modo da rispettare la temporizzazione di lettura;
- Area Group, applicato ai vari ChM, consente di mantenere il più possibile compatte e raggruppate le risorse logiche dell'FPGA destinate a ciascuno di essi;

- Use Low-Skew Lines, applicato ai segnali di controllo dei vari ChM, fa sì che questi siano instradati sulle linee globali a basso ritardo che percorrono orizzontalmente tutta l'FPGA, cosicché i corrispondenti tempi di propagazione dal DCM a ciascun ChM siano il più possibile ridotti.

Relativamente al Map Process, oltre ad aver scelto l'algoritmo più efficiente per l'ottimizzazione della velocità, i settaggi che hanno consentito di ottenere i migliori risultati sono stati *Timing-driven Placement* e *Combinatorial Logic Optimization*.

- Timing-driven Placement consente all'algoritmo di mappare le primitive all'interno dei vari blocchi logici seguendo opportune strategie che, in base ai constraints specificati dall'utente, tentano di ridurre il più possibile i cammini critici;
- Combinatorial Logic Optimization consente all'algoritmo di analizzare preventivamente le reti combinatorie ed effettuare opportune modifiche che rendano più facile il raggiungimento dei vari constraints specificati dall'utente.

Infine, per il Place and Route Process si è utilizzato il più efficace algoritmo disponibile, scegliendo la modalità *Multi-Pass*.

Tale modalità, a differenza della tradizionale *Single-Pass*, consente di ripetere più volte e in modo automatico l'intero processo di Place and Route, utilizzando i risultati ottenuti da ciascuna iterazione come linee guida per le iterazioni successive. Così facendo, l'algoritmo tenta di migliorare, ad ogni iterazione, i risultati ottenuti in quelle precedenti.

Una volta completato il numero di iterazioni stabilito, per ciascuna di esse ISE fornisce la stima delle prestazioni del sistema. In questo modo l'utente può scegliere l'iterazione per la quale si prevedono le migliori prestazioni, ed utilizzarne i risultati per programmare l'FPGA.

In particolare, per ottenere delle prestazioni migliori rispetto a quelle che si avrebbero con la semplice modalità Single-Pass, Xilinx suggerisce di utilizzare almeno 5 iterazioni. Si è scelto dunque di utilizzarne 7.

In definitiva, per avere un'idea della complessità generale degli algoritmi utilizzati in fase di implementazione, si consideri che per completare i processi Translate e Map, e per effettuare un'iterazione di Place and Route, è necessario un tempo complessivo di circa $1^{\mathrm{h}}15^{\mathrm{m}}$ utilizzando una macchina Pentium M 1.73 GHz, 1 GB RAM.

La Tab. 5.1 riassume le statistiche di occupazione delle risorse disponibili sull'FPGA, sia per il progetto intero con $N_{\mathrm{C}} = 64$, che per ogni singolo modulo. I risultati sono espressi in numero di *Slice* e di *Look-Up-Table* (LUT), sia come valore assoluto che come percentuale del totale.

Design Module	Slices		LUTs		Equivalent Gates
	number	utilization %	number	utilization %	
Entire Design, N_C = 64	5,003	97.71	9,961	97.28	2,747,130
DCM	154	3.01	244	2.38	10,016
ChM Synchronizer	2	0.04	0	0.00	32
ChM Counter	14	0.27	27	0.26	357
ChM Formatter	64	1.25	122	1.19	924
ChM Output Register	16	0.31	0	0.00	355
DMPORT	327	6.39	403	3.94	275,141

Tab. 5.1 Statistische di occupazione delle risorse disponibili sull'FPGA

Si consideri infatti che ciascun CLB è a sua volta suddiviso in 4 Slice identiche, ciascuna delle quali contiene 2 LUT, 2 Flip-Flop e molte altre risorse logiche. In particolare, la nostra FPGA dispone di 1280 CLB e, di conseguenza, di 5120 Slice e 10240 LUT.

Si osservi che il numero totale di Slice e di LUT utilizzate è molto minore rispetto a quello che si otterrebbe sommando i valori del DMPORT, del DCM e di N_C ChM. Questo accade perché molte Slice e molte LUT sono condivise tra più moduli. Ad ogni modo, i valori riportati, anche se non possono essere direttamente sommati, consentono comunque di comparare le complessità dei vari moduli.

Inoltre, in Tab. 5.1, sia per il progetto intero che per ogni singolo modulo, è riportata un'ulteriore stima della complessità, espressa in numero di gate equivalenti. Si osservi in particolare che il numero di gate equivalenti del progetto intero è molto maggiore del numero di gate effettivamente disponibili all'interno dell'FPGA. Questo è indice di quanto gli algoritmi utilizzati, pur essendo finalizzati all'ottimizzazione della velocità delle connessioni, siano riusciti comunque a contenere l'occupazione delle risorse logiche disponibili.

Infine, per scoprire se fosse stato possibile ottenere un N_C maggiore, sono stati eseguiti alcuni cicli di sintesi e implementazione utilizzando gli algoritmi finalizzati all'ottimizzazione dell'occupazione delle risorse. Tuttavia, si è scoperto che non si sarebbero superati comunque i 64 canali.

Di conseguenza, i risultati ottenuti in fase di sintesi e di implementazione sono, con buona probabilità, il massimo che si può ottenere – per questo progetto – dall'FPGA disponibile sulla scheda Orsys utilizzata.

5.2 Ottimizzazione del firmware del DSP

Il firmware del DSP, scritto in linguaggio C++, è stato sviluppato utilizzando l'ambiente Texas Instruments Code Composer Studio 2.5 [18].

Le principali ottimizzazioni svolte in fase di implementazione hanno riguardato ovviamente le routine di elaborazione dei pacchetti provenienti dal DPM.

Come visto nei Paragrafi 4.3 e 4.6, non appena uno dei due buffer Ping e Pong è pronto, la prima operazione che il DSP compie è quella di attivare il QDMA per il rinvio dei pacchetti in esso contenuti al DMPORT.

Successivamente, il flusso di esecuzione percorre uno di tre possibili cammini (*paths*), a seconda del valore di $L = 23, 16, 8$ scelto dall'utente.

In particolare, ciascun cammino è costituito da due loop annidati: quello esterno scansiona gli M pacchetti contenuti nel buffer, quello interno scansiona le N_{C} parole contenute in ciascun pacchetto e, per ognuna di esse, esegue le operazioni necessarie ad accumulare i risultati di conteggio nell'Output Buffer.

Ovviamente, quanto più i loop sono veloci, tanto minore è il tempo di elaborazione, sia di un pacchetto che di un buffer intero.

Dal momento che ciascuna iterazione non necessita dei dati prodotti dalla precedente, per velocizzare l'esecuzione dei loop è possibile sfruttare le potenzialità di *software pipelining* messe a disposizione dal DSP, cosicché la generica iterazione $i + 1$ possa cominciare T_{OV} secondi prima della conclusione della precedente iterazione i.

A tal fine, è necessario consentire al compilatore di "srotolare" (*unroll*) il più possibile ciascun loop durante la compilazione.

Si consideri ad esempio un generico loop di n iterazioni, per ciascuna delle quali il DSP esegue una sequenza di p operazioni. Srotolare tale loop per un numero di volte $m \leq n$ significa semplicemente costruire un loop equivalente di n/m iterazioni, per ciascuna delle quali il DSP esegue una sequenza di $m{\cdot}p$ operazioni. Il vantaggio sta nel fatto che, grazie al pipelining, l'esecuzione di un'iterazione del nuovo loop risulta molto più veloce dell'esecuzione di m iterazioni del loop di partenza.

Di conseguenza, grazie alle direttive `#pragma MUST_ITERATE` e `#pragma UNROLL`, si è fatto in modo che il compilatore srotolasse entrambi i loop di ciascun cammino. In particolare, i loop esterni sono stati srotolati per M volte.

Si osservi inoltre che, srotolando il loop esterno, il pipelining fa sì che T_{PRC} dipenda dal numero di volte per cui tale loop è stato srotolato, ossia, nel nostro caso, da M. Si indichi infatti con T_{LOOP} il tempo necessario a completare il loop interno, e con T_{BUF} il tempo necessario a completare quello esterno, coincidente per definizione con $M{\cdot}\,T_{\mathrm{PRC}}$. Si può dimostrare facilmente che

$$T_{\mathrm{BUF}} = M \cdot T_{\mathrm{PRC}} = M \cdot T_{\mathrm{LOOP}} - \big(M - 1\big) \cdot T_{\mathrm{OV}}\,, \tag{5.1}$$

e dunque che

$$T_{\mathrm{PRC}} = T_{\mathrm{LOOP}} - \left(1 - \frac{1}{M}\right) \cdot T_{\mathrm{OV}}\,. \tag{5.2}$$

Tuttavia, tale dipendenza non crea problemi, anzi è vantaggiosa. Infatti, a parità di T_{LOOP} e T_{OV} (il cui valore è dovuto alla particolare architettura interna del DSP), al crescere di M, T_{PRC} diminuisce.

Di conseguenza, al fine di ottimizzare le prestazioni dell'intero sistema, valgono ancora le considerazioni svolte nel Paragrafo 4.4: fare M il più grande possibile e minimizzare T_{PRC} in modo da rendere vera la (4.11). Ovviamente, per ridurre T_{PRC} è opportuno agire soprattutto su T_{LOOP}, ed è quindi per tal motivo che sono stati srotolati anche i loop interni di ciascun cammino.

Srotolando i loop, la dimensione del codice compilato aumenta necessariamente. Per contenere dunque l'occupazione di memoria, avendo scelto di svolgere i loop esterni per M volte, si è scelto di svolgere quelli interni solo per 2 volte. In questo modo, la dimensione del codice srotolato e, di conseguenza, quella di tutto il codice compilato, dipende esclusivamente da M.

Il valore di M dovrà quindi essere scelto in modo da poter contenere nella memoria interna del DSP non solo Ping, Pong, l'Output Buffer e gli altri dati, ma anche tutto il codice, compresi i loop srotolati.

Cercando di sfruttare il più possibile i 256 kB disponibili, dopo aver impostato $N_{\mathrm{C}} = 64$, è stato possibile ottenere $M = 16$.

Implementando il firmware con tali valori, e utilizzando i due timer di cui il DSP dispone internamente, sono stati misurati per 1000 volte i valori di T_{PRC} e di T_{BUF}, in funzione del formato selezionato. I valori minimi, medi e massimi così ottenuti sono riportati in Tab. 5.2.

Data Format	Packet Processing Time [µs] T_{PRC} @ N_C = 64			Buffer Processing Time [µs] T_{BUF} @ M = 16		
	Min	Avg	Max	Min	Avg	Max
F = 1	T_{PRC32}			T_{BUF32}		
	22.075	22.100	22.123	353.195	353.601	353.963
F = 2	T_{PRC16}			T_{BUF16}		
	24.850	24.919	25.225	397.596	398.708	397.596
F = 4	T_{PRC8}			T_{BUF8}		
	29.394	29.493	29.753	470.300	471.891	476.046

Tab. 5.2 Tempi di elaborazione del DSP per $N_{\mathrm{C}} = 64$ e $M = 16$

Considerando i valori medi di T_{PRC}, si può notare che la (4.11) è verificata.

Inoltre, sempre utilizzando i timer, è stato misurato per 1000 volte il tempo complessivo $T_{\mathrm{EDMA}} + \tau^0_{\mathrm{LLC}}$, ottenendo un valor medio di 15.817 µs.

Sostituendo infine nella (4.9) e nella (4.12) i tempi medi di elaborazione fin qui trovati, è stato possibile determinare, per ogni formato disponibile, i valori nominali delle minime finestre di integrazione e dei massimi bit-rate sostenibili in ingresso dal DSP in tempo reale. Si è ottenuto così

$$\begin{cases} T_{\mathrm{W32}} \simeq 23.090\ \mu\mathrm{s} \\ T_{\mathrm{W16}} \simeq 12.950\ \mu\mathrm{s} \\ T_{\mathrm{W8}} \simeq 7.618\ \mu\mathrm{s} \end{cases} \quad \text{e} \quad \begin{cases} B_{32} \simeq 88.702\ \mathrm{Mbps} \\ B_{16} \simeq 79.050\ \mathrm{Mbps} \\ B_{8} \simeq 67.188\ \mathrm{Mbps} \end{cases} \tag{5.3}$$

5.3 Test sperimentali e conclusioni

Il sistema, una volta implementato, è stato finalmente testato collegando gli N_{C} ingressi ad un generatore di forme d'onda.

In particolare, sono stati misurati i valori attuali delle minime finestre di integrazione, ottenendo rispettivamente

$$\begin{cases} \tilde{T}_{\mathrm{W32}} \simeq 24\ \mu\mathrm{s} \\ \tilde{T}_{\mathrm{W16}} \simeq 14\ \mu\mathrm{s} \\ \tilde{T}_{\mathrm{W8}} \simeq 8\ \mu\mathrm{s} \end{cases} \tag{5.4}$$

Tali valori risultano molto vicini a quelli nominali, e dunque il modello teorico utilizzato per determinarli risulta valido.

Inoltre, sostituendo tali valori nella (2.11) è possibile calcolare i valori attuali dei pulse-rate di ingresso f_{IN16} e f_{IN8} rappresentati nel grafico di Fig. 2.2. Si ha dunque

$$\begin{cases} \tilde{f}_{\mathrm{IN16}} \simeq 4.7\ \mathrm{GHz} \\ \tilde{f}_{\mathrm{IN8}} \simeq 32\ \mathrm{MHz} \end{cases} \tag{5.5}$$

Si osservi in particolare che è inutile calcolare anche il valore attuale di f_{INC}, dal momento che deve essere $f_{\mathrm{IN}} < f_{\mathrm{INmax}} = 45$ MHz.

Con tali risultati, e ricordando inoltre che $L_{\mathrm{C}} = 23$, $T_{\mathrm{Wmax}} \simeq 186$ ms, il grafico di Fig. 2.2 descrive completamente le prestazioni del sistema, racchiudendo in sé tutte le principali informazioni da fornire a potenziali utilizzatori.

Inoltre, per poter confrontare le prestazioni del sistema con quelle degli altri contatori multicanale, per $N_{\mathrm{C}} = 32, 16, 8, 4, 2$ sono stati misurati i valori attuali delle minime finestre di integrazione che si ottengono.

Per ciascuno di tali N_{C}, è stato modificato, di volta in volta, soltanto il firmware del DSP. In particolare, è bastato far sì che l'EDMA, ad ogni interruzione ricevuta dal DPM, trasferisse semplicemente un pacchetto più piccolo, formato soltanto dai primi N_{C} canali consecutivi.

Infine, tenendo presente le considerazioni svolte nel Paragrafo 4.1, per $N_{\mathrm{C}} = 64, 2$ sono stati misurati i valori attuali delle minime finestre di integrazione che si ottengono disattivando l'accumulo in tempo reale.

I risultati ottenuti sono riportati in Tab. 5.3.

Real-time Accumulation	N_C	M	Minimum Integration Window Width – Actual Values		
			T_{W8}	T_{W16}	T_{W32}
enabled	64	16	8 µs	14 µs	24 µs
	32	32	4 µs	7 µs	12 µs
	16	55	2 µs	4 µs	6 µs
	8	90	1 µs	2 µs	3 µs
	4	93	789 ns	1.4 µs	2.1 µs
	2	95	378 ns	744 ns	1.4 µs
disabled	64	16	2.7 µs	5.4 µs	10.5 µs
	2	95	88 ns	189 ns	378 ns

Tab. 5.3 Minime finestre di integrazione in funzione di N_C, M e dell'accumulo in tempo reale

Per prima cosa si osservi che, diminuendo N_C da 64 a 32, è stato possibile aumentare proporzionalmente M, da 16 a 32. Questo ha fatto sì che la dimensione dei buffer Ping e Pong sia rimasta uguale, mentre quella del codice srotolato sia raddoppiata. Viceversa, diminuendo N_C da 32 a 16, non è stato possibile aumentare M in modo proporzionale, dal momento che il codice avrebbe avuto una dimensione troppo grande per essere contenuto nella memoria interna del DSP. Lo stesso si può dire per tutti gli altri casi.

Riguardo alle prestazioni con accumulo real-time, è significativo notare che con 8 canali si riesce a raggiungere il "traguardo" di 1 µs, e che con solo 2 canali ci si avvicina notevolmente alle prestazioni di PicoHarp 300 riassunte nel Paragrafo 1.3.

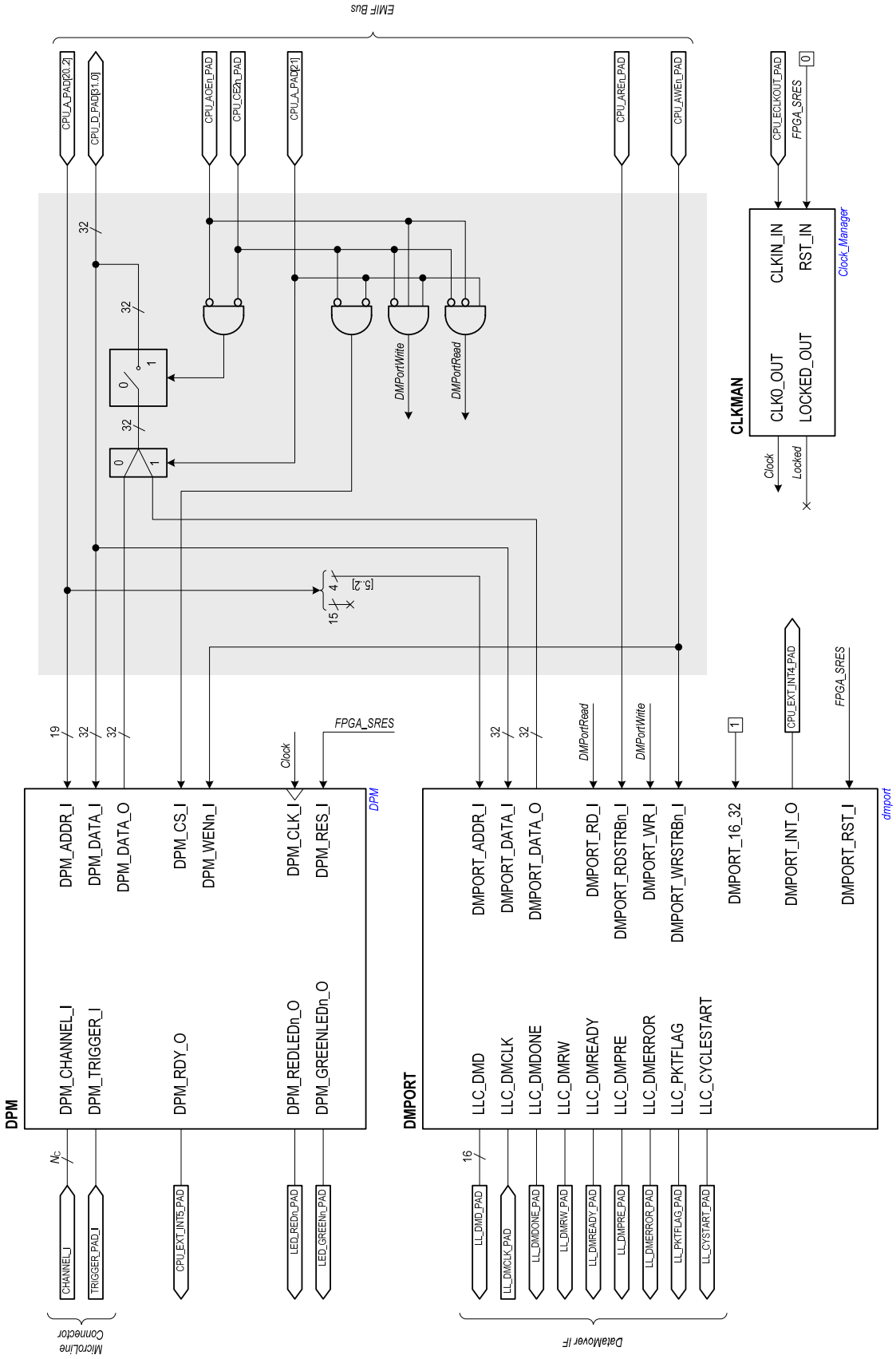
Riguardo invece alle prestazioni senza accumulo real-time, i risultati sono ancora più significativi, sia perché superano quelli di PicoHarp 300 (relative, ovviamente, al solo utilizzo come contatore), sia perché sono stati misurati senza aver posto alcun rimedio al problema del rinvio dei pacchetti da DSP a DMPORT. In particolare, come visto nel Paragrafo 4.1, risolvendo tale problema, questi risultati potrebbero addirittura dimezzare.

I principali sviluppi futuri dovranno quindi prevedere:

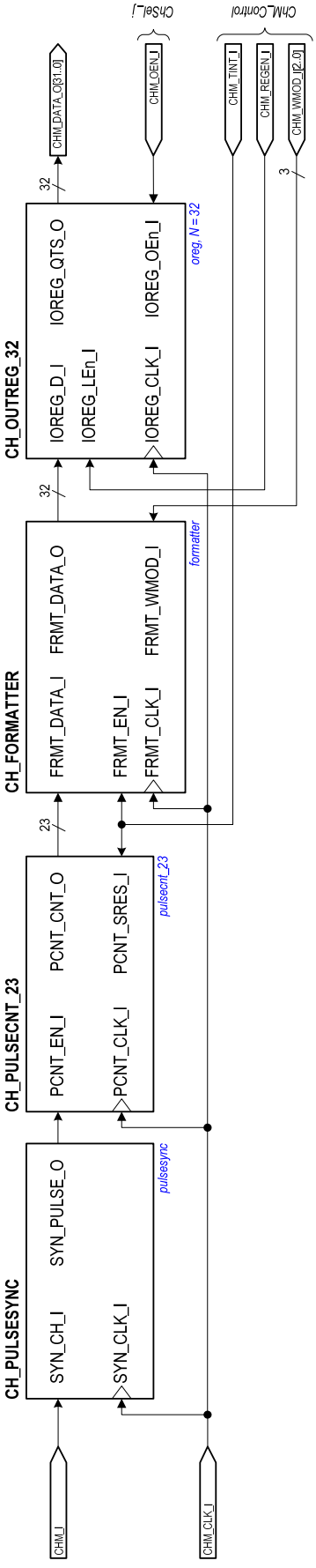
- il trasferimento diretto dei pacchetti dal DPM al DMPORT;
- la possibilità di disattivare l'elaborazione real-time svolta dal DSP;
- la possibilità di far svolgere l'elaborazione real-time al PC di controllo (ossia al server Linux), che riceve continuamente i risultati di conteggio attraverso il link isocrono.

In particolare, tale ultima possibilità si è già rivelata vincente per molti dei contatori miluticanale presenti attualmente sul mercato.

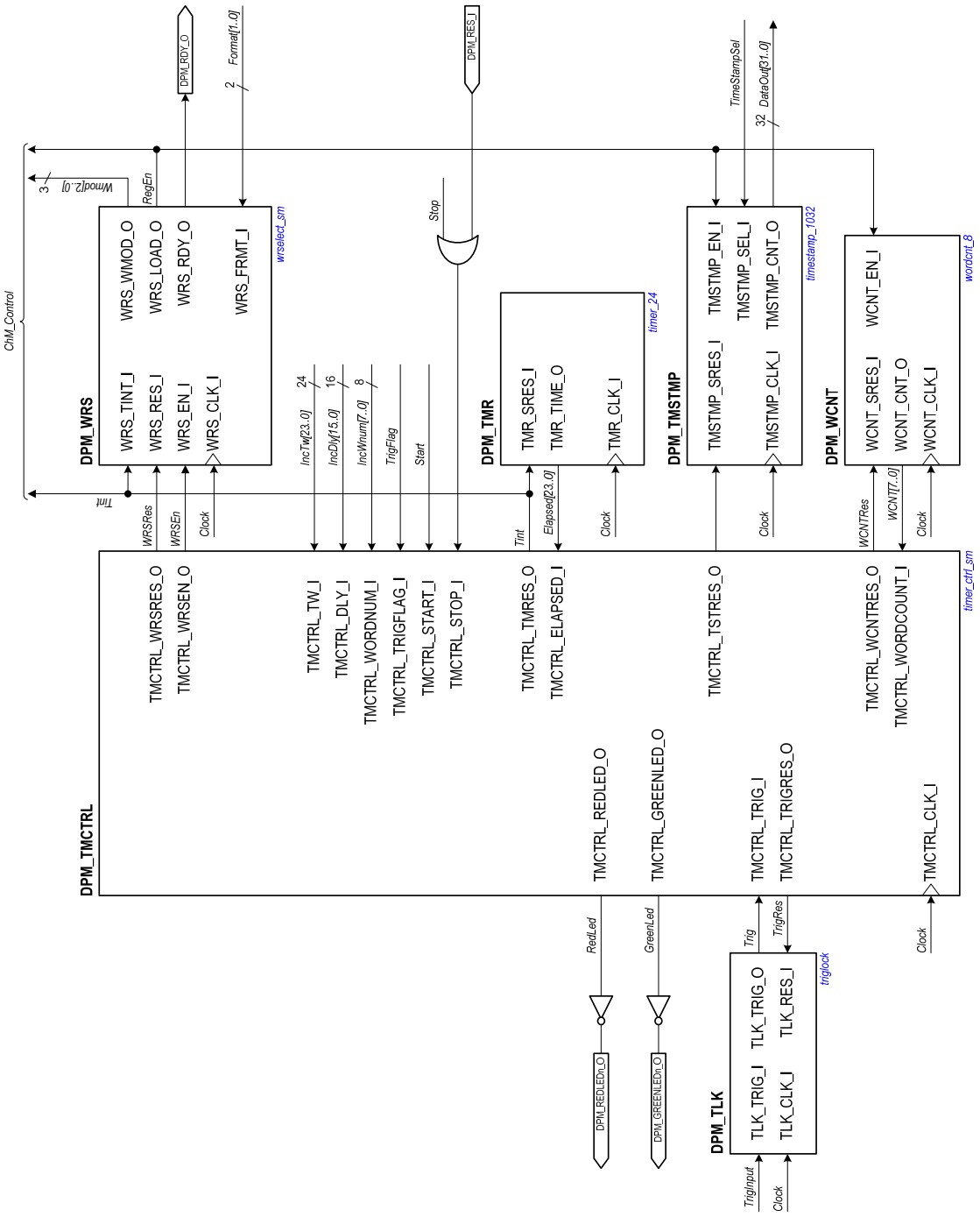
Appendice A. Tavole FPGA



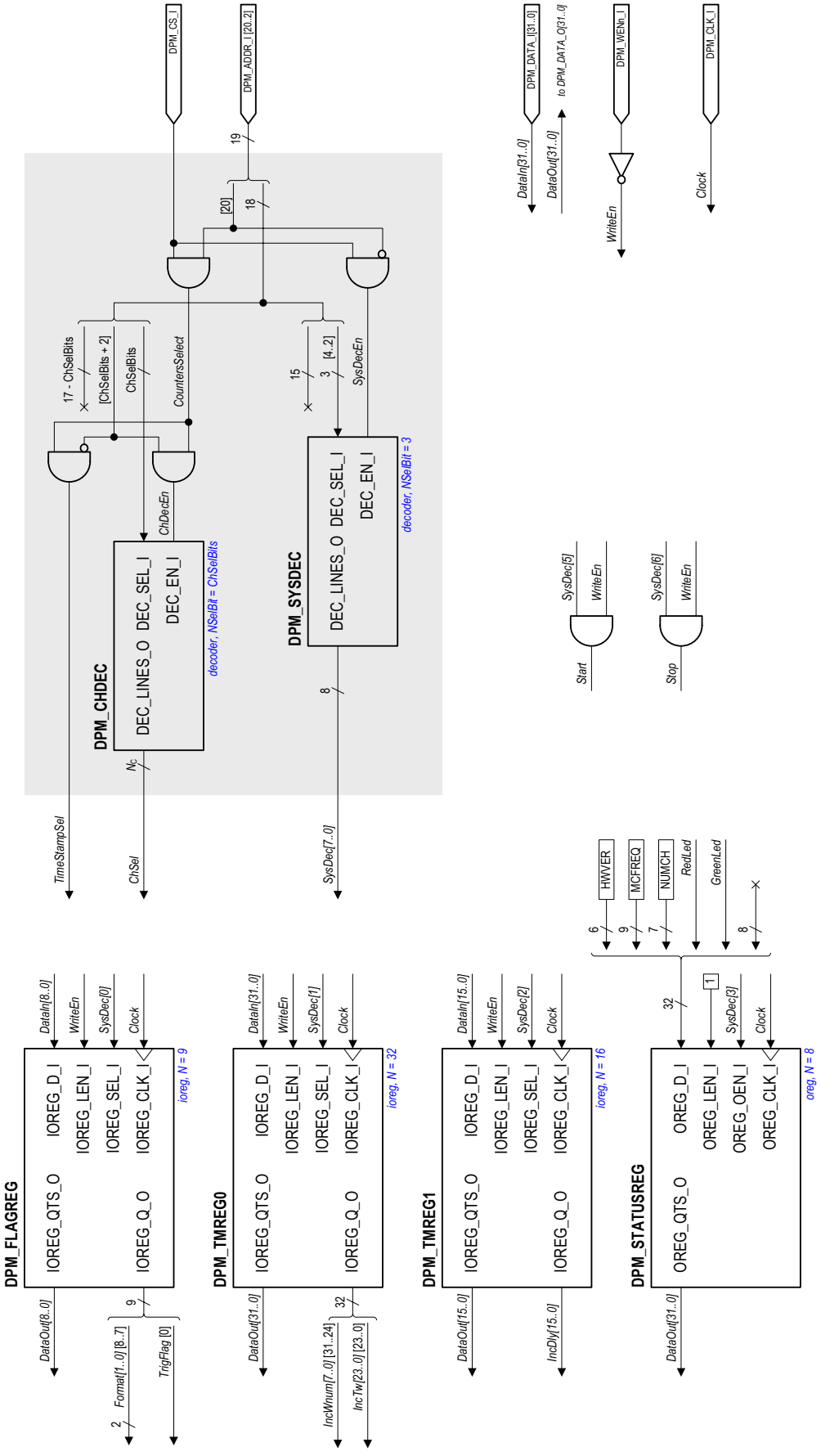
Tav. A.1 Architettura interna dell'FPGA al più alto livello gerarchico



Tav. A.2 Architettura del generico Channel Module (ChM)



Tav. A.3 Architettura del DCM – parte 1



Tav. A.4 Architettura del DCM – parte 2

Bibliografia

[1] PicoQuant GmbH, D-12489 Berlin, Germany, www.picoquant.com

[2] Ortec, Oak Ridge, TN 37830, USA, www.ortec-online.com

[3] Addi-Data GmbH, D-77833 Ottersweier, Germany, www.addi-data.com

[4] KineticSystems, Lockport, IL 60441, USA, www.kscorp.com

[5] SeaBird Electronics Inc., Bellevue, WA 98005, USA, www.seabird.com

[6] Microsoft Corporation, www.microsoft.com/visualc

[7] National Instruments Corporation, www.ni.com/labview

[8] European Southern Observatory, www.eso.org

[9] Project infos at www.shef.ac.uk/physics/people/vdhillon/ultracam

[10] Project infos at www.eso.org/projects/aot/macao_vlti

[11] Project infos at www.eso.org/instruments/sinfoni

[12] F. Zappa, S. Tisa, S. Cova, P. Maccagnani, D. Calia, R. Saletti, R. Roncella, G. Bonanno, M. Belluso, "Single-photon avalanche diode arrays for fast transients and adaptive optics," *IEEE Trans. on Instrumentation and Measurement*, vol. 55, no. 1, pp. 365-374, Feb. 2006.

[13] C. Niclass, M. Sergio, E. Charbon, "A Single Photon Avalanche Diode Array Fabricated in Deep-Submicron CMOS Technology," *Proc. of Design, Automation and Test in Europe*, vol. 1, pp. 1-6, Mar. 2006.

[14] A. Tosi, S. Cova, F. Zappa, M. A. Itzler, R. Ben-Michael, "InGaAs/InP Single Photon Avalanche Diode Design and Characterization," *Proc. of the 36th European Solid-State Device Research Conference*, pp. 335-338, Sept. 2006.

[15] D. Mosconi, D. Stoppa, L. Pancheri, L. Gonzo, A. Simoni, "CMOS Single-Photon Avalanche Diode Array for Time-Resolved Fluorescence Detection," *Proc. of the 32nd European Solid-State Circuits Conference*, pp. 564-567, Sept. 2006.

[16] Orsys Orth Systems GmbH, D-88677 Markdorf, Germany, www.orsys.de

[17] Xilinx Inc., San Jose, CA 95124-3400, USA, www.xilinx.com

[18] Texas Instruments Inc., Dallas, TX 75243-4136, USA, www.ti.com

[19] IEEE 1394-1995, "IEEE standard for high performance serial bus (ANSI)," *Institute of Electrical and Electronical Engineers*, 445 Hoes Lane, Piscataway, NJ 08855.

[20] D. Audino, F. Baronti, R. Roncella, R. Saletti, S. Tisa, F. Zappa, M. Belluso, G. Bonanno, "60-channel 10 µs Time-Resolution Counter Array for Long-Term Continuous Event Counting," *IEEE Trans. on Nuclear Science*, in press.

[21] Aldec Inc., Henderson, NV 89074, USA, www.aldec.com