POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Corso di Laurea Specialistica in
Ingegneria Informatica

Sviluppo di un'architettura basata su FPGA per il controllo attivo del rumore, con funzionalitá hardware in the loop

Relatore: Prof. Fabio Salice
Correlatore: Prof. Luigi Piroddi

Tesi di laurea di: Matteo Ferrario
matr. 754708

Anno Accademico 2011 / 12

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Master of Science in
Computer Engineering

Development of an FPGA based module for active noise control, with hardware in the loop framework

Supervisor: Prof. Fabio Salice
Assistant Supervisor: Prof. Luigi Piroddi

Master Graduation Thesis by: Matteo Ferrario
Student Id. number 754708

Academic Year 2011 / 12

alla mia famiglia

Sommario

Utilizzando il termine calcolo pervasivo (dall'inglese ubiquitous computing) si riassume la presenza di sistemi elettronici digitali nella maggior parte degli oggetti utilizzati della nostra quotidianitá.
Una presenza tuttavia nascosta, che ci porta a non comprendere come il dispositivo agisce ma a beneficiare passivamente della sua funzionalitá, secondo un'ottica di integrazione intelligente nell'ambiente circostante.
Il testing delle funzionalitá di tali dispositivi é reso ostico dalla difficoltá intrinseca della progettazione di un componente hardware: una volta realizzato il prototipo si perde ogni traccia del comportamento dei singoli blocchi di cui é composto e si deve dedurre il corretto funzionamento o malfunzionamento tramite la verifica della risposta del dispositivo a stimoli selezionati.
Non esiste il concetto di variabili e debug runtime come nel software.
Risulterebbe quindi utile avere a disposizione un framework software per l'invio automatizzato di stimoli di input ad un generico sistema elettronico digitale il quale, contornato da un blocco hardware di trasmissione, possa riceverli, utilizzarli ed inviare i dati di output al fine di verificare la correttezza delle funzionalitá. Questo elaborato si pone l'obiettivo di sviluppare un ambiente che possa realizzare il sistema descritto tramite la tecnica hardware in the loop (HIL). L'ambiente progettato sará utilizzato per la verifica di un dispositivo hardware FPGA sul quale é implementato un algoritmo per la cancellazione del rumore (ANC).
Compito della simulazione HIL é finalizzare diversi tipi di prove sul dispositivo, al fine di stressarlo con input di natura pura (prodotti virtualmente dal computer) e con input complessi che si avvicinino a quelli presenti in un contesto di funzionamento reale. A seguito di queste prove sará possibile validare il dispositivo in anello chiuso e procedere alla sua messa in opera nell'ambiente proprio di lavoro.

Abstract

By using the term ubiquitous computing we mean the pervasiveness of embedded systems in many fields of our lives. Ubiquitous computing also gives the name to the third wave in computing systems: first wave was composed by mainframes, machines which their usage was shared among different users; second wave was the personal computing era, which connected each user to its own workstation. Third and last wave until nowadays, it's the age of calm technology, where more and more computing systems work and help into the background of our lives.
Embedded systems production is thus increasing with an exponential growth, to fullfill the market demand of new technologies in our society. A particular aspect of these systems is the testing which comes after the design of a new component. There's an intrinsic difficulty in this process, rising from the particular nature of hardware realization: once the board is routed, it's no more possible to inspect internal signals used in simulation, and the correct behavior must be inferred only by reading outputs. There's no concept of variable, debugging or at least not as easy as in the software.
It could be useful to develop a software framework to send inputs to a generic embedded system which, completed with an hardware block for trasmission, could receive them, elaborate them and send back results to test the correctness of the architecture. In this work a testing framework for embedded systems is developed, by realizing an hardware in the loop simulation (HIL). This new system will be used to verify an FPGA with implemented an active noise control algorithm. The purpose for this simulation is to complete different kind of tests on the device, to stress it with simple inputs (generated by the computer) and then complex inputs close to the ones sampled in a working environment.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Obiettivo e motivazioni

Il progetto realizzato in questo elaborato tratta tematiche inerenti ad algoritmi di controllo attivo del rumore (ANC), la loro esecuzione su sistemi elettronici digitali e la successiva validazione tramite simulazioni hardware in the loop (HIL).
Partendo da un problema di ANC ci si pone come obiettivo la creazione di un prototipo hardware che possa implementare un algoritmo di controllo adattivo, specifico per la sua risoluzione.
Viene presentata una panoramica del problema.
Le ventole presenti nei computer desktop o portatili di utilizzo comune sono necessarie ad evitare il surriscaldamento e la conseguente fusione dei componenti elettronici all'interno, operanti ad alta frequenza. Esse hanno dimensioni standard di pochi centimetri e, con una portata contenuta, producono un rumore tollerabile per una persona che accede ed utilizza il dispositivo per lavorare. Al contrario, le ventole installate nei computer per l'elaborazione del segnale digitale terrestre producono un rumore intenso, proporzionale alla loro velocità e quindi al surriscaldamento dei processori utilizzati.
Considerando una stanza dedita all'elaborazione del sistema DVB-T, la potenza di calcolo richiesta necessita l'installazione di più server, organizzati in strutture verticali denominate rack: è facilmente immaginabile come il rumore sonoro possa diventare fastidioso per chiunque debba rimanere del tempo in quella stanza.
E' stata quindi analizzata la situazione secondo un'ottica del controllo attivo del rumore: ovvero implementare un algoritmo di ANC per la riduzione del rumore acustico prodotto dalle ventole del rack.
Il passaggio successivo è stato scegliere una piattaforma di calcolo su cui codificare la funzionalità stabilita. Dopo un'analisi delle prestazioni, dell'ingombro e della potenza consumata, il dispositivo scelto per il progetto è una scheda riprogrammabile del tipo Field Programmable Gate Array (FPGA). Altra chiave

di lettura per questa scelta è la generalizzazione del progetto: la realizzazione di un algoritmo su FPGA può andare oltre i fini previsti in questo progetto, permettendo la riconfigurazione della scheda per altri problemi analoghi di controllo del rumore.
L'ultimo obiettivo che viene proposto prevede di creare un sistema hardware in the loop in cui inserire la piattaforma descritta, con lo scopo di condurre delle simulazioni di test e validazione.
Questo problema è interessante sia in termini di supporto al design hardware, sia per la genericità dell'architettura.
La validazione di un dispositivo hardware è un procedimento lungo e complesso, data la quantità di problematiche che possono sorgere nel momento in cui si pone l'oggetto nell'ambiente di lavoro: errori nella struttura interna dell'hardware, tempistiche e sincronizzazione; malfunzionamento dei sensori esterni, della loro alimentazione e interfacciamento; cattivo collegamento delle tracce con disturbi dovuti a fenomeni parassiti.
Con l'esecuzione di una simulazione HIL si porta il sistema ad un livello virtuale, in cui viene testata solo la struttura interna dell'hardware in risposta ad ingressi di natura digitale preprogrammati. Il vantaggio è quello di escludere la parte sensoriale dal testing e focalizzarlo sulla parte core. Il secondo giovamento per la realizzazione di questo framework è in termini di riutilizzo dell'implementazione: progettando un sistema completo di collegamento per simulazioni HIL, sarà possibile utilizzarlo non solo nella validazione di questo specifico dispositivo, ma anche su altri hardware riconfigurando gli input da fornire.

Il progetto PIANO

PIANO è un progetto di ricerca iniziato nel febbraio 2011 in collaborazione con il Politecnico di Milano, con acronimo PIAttaforme a basso costo per il coNtrOllo attivo del rumore.
La partecipazione a questo progetto ha fornito gli elementi chiave per la stesura di questo elaborato, nonchè la dotazione degli strumenti necessari per la sua realizzazione.

1.1 Descrizione generale

Per avere una panoramica generale del lavoro svolto si procede alla descrizione del sistema suddiviso per le funzionalità offerte. Questa visione iniziale sarà successivamente aggiornata nel terzo capitolo, arricchendo di dettagli i componenti costitutivi.
Il problema originale prevede di ridurre il rumore prodotto da due ventole tra-

mite tecniche di controllo attivo del rumore. L'hardware di implementazione scelto è una scheda FPGA, che conferisce al progetto garanzie di flessibilità e contenimento delle dimensioni nonché un costo contenuto data la sua genericità di fabbricazione. La scheda dovrà essere in grado di prelevare il rumore originale tramite un microfono, elaborarlo opportunamente tramite tecniche ANC e stabilire in output da speaker un segnale in controfase che possa cancellare il rumore oltre una zona dello spazio.
Partendo da questo concetto vengono inizialmente implementati i blocchi funzionali VHDL per la parte algoritmica del sistema; il campionamento e la gestione dei segnali provenienti dai sensori e diretti agli attuatori (microfoni e speaker) sono effettuati tramite specifici blocchi di conversione dac/adc, scritti in codice hardware VHDL.
Il secondo passaggio è creare un ambiente di testing dell'algoritmo, per inserirlo in un contesto virtuale di utilizzo, ovvero fornire in ingresso alla board dei segnali selezionati ad hoc per la verifica funzionale. Questo processo viene chiamato hardware in the loop, e prevede la comunicazione dei segnali di stimolo da un agente esterno quale un computer o un altro dispositivo: il passaggio chiave è assicurare un link di collegamento stabile tra la scheda ed il master della simulazione, che permetta l'invio dei segnali di test.
Prima di chiudere l'anello per la simulazione, l'algoritmo originale deve essere revisionato, in quanto il segnale audio non deve essere campionato dai microfoni e condotto agli altoparlanti ma deve essere prelevato dal reparto realizzato appositamente per la trasmissione dati HIL.
Dall'altro capo della trasmissione il computer funge da master del processo: il suo compito è fornire le parole chiave di comando alla scheda, inviare i dati di input per la simulazione e collezionare i risultati prodotti dall'hardware, presentandoli graficamente in maniera intuitiva. Per esporre queste funzionalità il terzo ed ultimo passaggio è il coding di una piattaforma software per la gestione dei dati di simulazione e per la trasmissione.

1.1.1 Stato dell'arte

L'implementazione specifica di una piattaforma HIL per il controllo di un algoritmo ANC implementato su hardware è un'innovazione in ambito scientifico: cenni ad applicazioni simili non sono stati trovati in letteratura, facendo risalire l'ultima ricerca a Gennaio 2012.
Sono invece presenti numerosi articoli riguardanti aspetti connessi all'applicazione:

- Realizzazione di algoritmi di controllo attivo del rumore su scheda FPGA
- Utilizzo di simulazioni HIL per testare le funzionalità di sistemi embedded

- Impiego di schede FPGA come master di simulazione per HIL (ovvero programmate per verificare l'unità sotto test).

Vengono quindi presentati alcuni articoli per ciascuna categoria. Il progetto [8] descrive l'utilizzo di un framework per finalizzare simulazione HIL su controller digitali codificati su FPGA: in particolare il framework permette una rapida prototipazione del controller su un sistema RAPTOR2000 e la successiva validazione tramite matlab/Simulink con input da computer. I risultati mostrano come l'approccio sia efficace ma il tempo necessario per la simulazione aumenta con la complessità del design, mantenendo gli input costanti. Da questo si evince come durante la scrittura di codice vhdl per prototipazioni fpga si debba mantenere la struttura il più semplice possibile, almeno per la fase di testing delle componenti principali. Altresì interessante è notare la tecnica utilizzata per la trasmissione degli stimoli: l'interfacciamento è possibile con un chip PLX9054 che connette il bus PCI con un protocollo locale a 32 bit per la board.
Similmente anche nell'elaborato [4] si propone un collegamento usb tra computer e fpga per testare l'efficacia del codice vhdl come in testbench; in questo caso viene utilizzato un chip alternativo prodotto dalla FTDI, il quale permette una connessione tramite protocollo proprietario fino a 12Mbit/s.
Dalle conclusioni si comprende la potenzialità di questo dispositivo per predisporre simulazioni HIL general purpose su FPGA interfacciate con il pc sullo standard USB 2.0 .
L'articolo [7] analizza un sistema basato su fpga per creare un'applicazione di augmented audio: si implementa un algoritmo ANC all'interno di una scheda per diminuire il rumore esterno percepito con una coppia di cuffie. I risultati hanno mostrato come una piattaforma FPGA possa essere un elemento chiave nel processare segnali audio complessi in ambienti audio ad alta fedeltà; con la scheda a loro disposizione, una Spartan 3 della Xilinx, è stato possibile implementare filtri per la convoluzione fino al 256 esimo tap. Un'implementazione interessante dell'algoritmo LMS viene trattata in [5]: la versione rivisitata permette un utilizzo di gate minore comparata alla versione standard; i risultati mostrano come l'algoritmo implementato possa essere eseguito fino ad una velocità di 50MHz, producendo performance comparabili oltre che una maggiore velocità di esecuzione.

1.2 Contenuti

Questa tesi tratta in maniera completa gli aspetti di concettualizzazione, design e realizzazione del progetto presentato.
In questo capitolo è contenuta una panoramica del lavoro svolto: vengono quindi elencate le motivazioni e gli obbiettivi che si propone, il contesto e le

tematiche in cui è stato sviluppato, gli spunti dalla letteratura per realizzarlo.
Nel secondo capitolo viene presentato da un punto di vista teorico l'insieme delle nozioni richieste per il completamento del progetto. In particolare viene inizialmente trattata la teoria di ANC per la riduzione del rumore acustico, le sue origini e su quale principio si basa.
In seguito è discusso l'hardware utilizzato per l'implementazione, costituito da una scheda FPGA, prestando attenzione agli aspetti legati alla storia di questi dispositivi e ai loro punti di forza. L'ultima sezione descrive la tecnica HIL per il testing hardware: partendo da alcuni cenni storici si definisce da che problemi è stata innescata e qual'è il suo approccio risolutivo.
Il terzo capitolo dell'elaborato contiene i dettagli per concretare il progetto. Nella prima sezione viene elaborata la parte algoritmica del sistema, ovvero la revisione degli algoritmi ANC e la loro codifica mediante blocchi hardware VHDL.
Continuando nel capitolo viene presentato il procedimento completo per realizzare la simulazione HIL. Partendo dal collegamento realizzato tra scheda e computer, viene analizzato il chip che instaura la connessione e il protocollo scelto per i segnali sul canale. In seguito viene trattata l'implementazione VHDL del blocco di tramissione : in questa parte per ciascun reparto (bufferizzazione dati, trasmissione, test) vengono descritti i segnali per l'interfacciamento I/O ed i processi critici delle macchine a stati.
Completata la parte di hardware viene trattata l'interfaccia software che regola e gestisce la simulazione HIL. In questa sezione è descritto il design, l'interfaccia grafica e le routine del progetto software in piattaforma .NET. Tra le funzionalità chiave troviamo: la comunicazione via USB, la simulazione degli ingressi HIL e la presentazione dei risultati.
Nell'ultimo capitolo vengono elencate le prove effettuate e i risultati del progetto, contornati da possibili sviluppi futuri dell'applicazione.

Capitolo 2

Richiami e definizioni di base

Questo secondo capitolo di introduzione ha lo scopo di dare al lettore un'infarinatura di base delle tematiche in cui si inserisce la tesi. Vengono quindi trattate le nozioni teoriche necessarie per comprendere i tre aspetti principali del progetto, ovvero:

- Controllo attivo del rumore, algoritmi;
- Sistemi elettronici digitali, FPGA;
- Simulazioni tramite tecniche di tipo hardware in the loop (HIL).

2.1 Controllo attivo del rumore

Il controllo del rumore è un aspetto rilevante in molte applicazioni del progresso scientifico. Parallelamente con l'introduzione di innovazioni, è stato necessario introdurre dei dispositivi per l'attenuazione del rumore: ne è un caso esemplare l'automobile e l'invenzione delle marmitte per il motore.
Ci sono due tipi di controllo del rumore: passivo e attivo. Nel controllo passivo l'onda sonora del rumore viene assorbita da strutture che non generano potenza al loro interno, ma si limitano a diminuire l'intensità del suono. Fanno parte di questa categoria dispositivi come isolanti, barriere, silenziatori, marmitte, ecc. Il loro utilizzo è semplice ma hanno una grande limitazione, l'ingombro: per attenuare onde a bassa frequenza occorrono infatti dei dispositivi di grandi dimensioni e conseguentemente di costo maggiore.
Il controllo attivo del rumore (ANC) prevede l'utilizzo di sorgenti di rumore secondarie che permettano la riduzione del rumore iniziale (primario), basandosi sul principio della sovrapposizione di onde [6]. Questo principio, come mostrato nella figura 2.1, afferma che due onde sonore che si trovano nello stesso punto dello spazio risultano in un onda di intensità pari alla somma delle intensità delle singole: prendendo l'esempio chiave di due onde alla stessa

frequenza ma in opposizione di fase (rumore primario, rumore secondario di cancellazione), la loro somma risulta in un segnale di ampiezza nulla (rumore risultante). Partendo da questo concetto analizziamo i componenti fondamen-

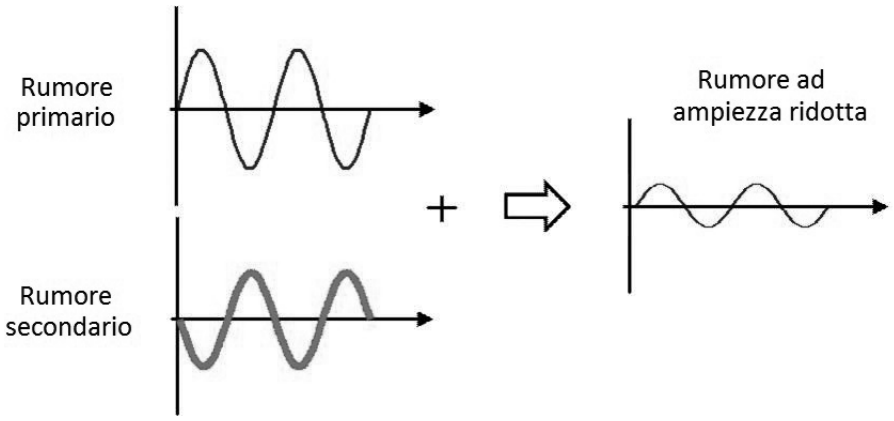

Figura 2.1: *Somma di onde sonore*

tali per la realizzazione di un sistema ANC.

- Microfoni (sensori): necessari per prelevare il segnale rumore. Devono essere posizionati in punti strategici dell'ambiente da controllare
- Speaker (attuatori) : riproducono il segnale secondario di cancellazione. Come i microfoni la loro posizione è di notevole impatto sulle prestazioni del sistema.
- DSP: dispositivo per il controllo del sistema. Si occupa della quantizzazione dei segnali, elaborazione secondo gli algoritmi ANC, e comando del segnale degli speaker.

La teoria dell' active noise control definisce gli algoritmi per la cancellazione del rumore in diversi ambienti, statici o dinamici, e permette di risolvere le difficoltà implementative legate alla complessità del singolo problema.

2.1.1 Tipologie di ANC

I sistemi ANC si differenziano per il tipo di rumore da controllare e per la configurazione di microfoni e speaker.

- Broadband feedforward ANC : è la configurazione utilizzata per la cancellazione di un segnale rumore ad ampio spettro; significa che il rumore non ha componenti dominanti ed è uniformemente distrubuito nelle frequenze. É composto da un microfono per il segnale di riferimento da

cancellare, uno speaker per la cancellazione e un microfono per l'errore residuo;

- Narrowband feedforward ANC: controlla tipicamente i sistemi con parti rotatorie in cui il rumore prodotto è periodico e centrato in alcune componenti dominanti. Il microfono di riferimento è a volte integrato con altri sensori per calcolare la frequenza dominante (es: velocità della ventola);
- Feedback ANC : sistema simile al Broadband ma risparmia il microfono di riferimento, pagando un prezzo sulla ridotta stabilità del sistema;
- Multiple channel ANC: questo sistema gestisce il funzionamento dell'algoritmo in ambienti in cui il suono può seguire diversi percorsi, oppure servono diversi punti di cancellazione;

Il problema definito da queste tecniche è il medesimo, ovvero l'identificazione del filtro che rappresenta il cammino compiuto dal suono, dal punto di registrazione al punto di emissione.
Una volta stabilito il filtro che descrive questo path è possibile convolvere il segnale registrato ed emettere dallo speaker un segnale il più vicino possibile a quello reale che ha compiuto il percorso fisico per ridurlo/cancellarlo.

2.1.2 Algoritmi: LMS

Il Least Mean Squares è uno degli algoritmi più diffusi per il procedimento della stima adattiva di un filtro.
L'algoritmo ha lo scopo di identificare un generico filtro, descritto dalla rispo-

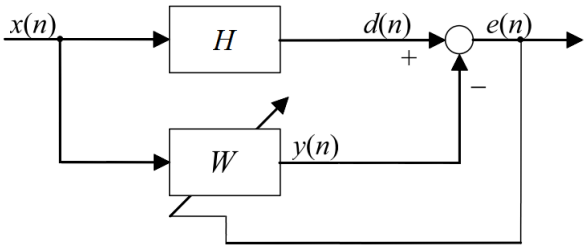

Figura 2.2: *Identificazione tramite minimi quadrati (LMS)*

sta all'impulso h(n) non nota a priori, tramite l'approssimazione in un filtro secondario, w(n). Il segnale d(n) è definito dalla convoluzione

$$d(n) = h(n) * x(n) \tag{2.1}$$

mentre y(n) è ottenuto applicando la convoluzione:

$$y(n) = \mathbf{w}(n)^T * \mathbf{x}(n) \tag{2.2}$$

utilizzando il vettore dei pesi, stimati dall'algoritmo e la finestra dei valori di x precedenti:

$$\mathbf{w}(n) = [w_1(n), \ldots, w_L(n)]^T \tag{2.3}$$

$$\mathbf{x}(n) = [x(n), \ldots, x(n - L + 1)]^T \tag{2.4}$$

Il procedimento per ottenere i coefficienti adattivi della risposta w(n) è molto semplice e parte della considerazione di minimizzare la funzione di costo:

$$J(n) = E|e(n)^2| \tag{2.5}$$

contenente l'errore residuo e(n), dato dalla differenza tra disturbo originale e risposta dell'algoritmo:

$$e(n) = d(n) - y(n) \tag{2.6}$$

Per raggiungere il minimo viene calcolata la derivata della funzione di costo e posta a 0, ottenendo la formula per il procedimento iterativo della convergenza dei pesi:

$$w_l(n + 1) = w_l(n) + \mu x(n - l)e(n) \quad \text{per} \quad l = 0, 1, \ldots, L - 1 \tag{2.7}$$

2.1.3 Algoritmi: FxLMS

In applicazioni reali di ANC non è possibile utilizzare la versione standard del LMS, perchè non prevede gli effetti introdotti dalla secondary path [11].

Il blocco S(z) è una modellizzazione teorica di una catena di trasformazioni legate al segnale y, partendo dal momento in cui viene elaborato dall'algoritmo fino a quando si trova nel punto di cancellazione. Ci sono diversi fattori che influenzano il segnale puro dell'algoritmo, tra i quali:

- convertitori DAC e ADC: producono inevitabilmente errore di quantizzazione, rumore;
- amplificatori : la risposta non è lineare e introduce distorsioni nel segnale;
- casse : diffusione non ottimale, risposta in frequenza;
- canale compreso fra lo speaker ed il punto di sovrapposizione acustica;

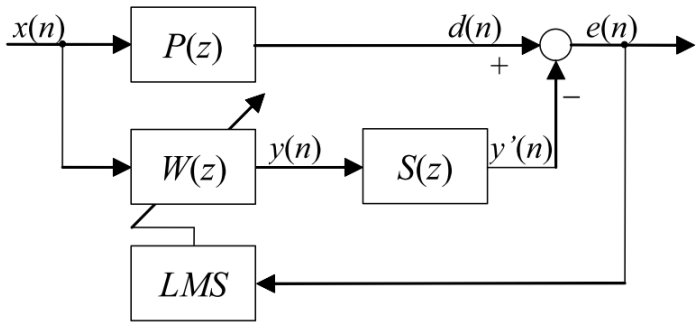

Figura 2.3: *Percorso secondario S(z)*

- canale compreso fra il punto in cui le due onde si incontrano ed il punto dove viene posto il microfono di errore;
- microfono: risposta in frequenza, disturbi.

Dalla quantità di fattori si può evincere come la sola stima del canale primario non sia sufficiente a produrre un segnale che possa cancellare fedelmente l'originale. E' possibile calcolare l'effetto del secondary path sull'errore finale.

Errore senza modellizzazione del secondary path

$$e(n) = d(n) - y(n) \text{ , con } y(n) = \mathbf{w}(n)^T * \mathbf{x}(n) \tag{2.8}$$

Errore con secondary path

$$e(n) = d(n) - y'(n) \text{ , con } y'(n) = s(n) * y(n) \tag{2.9}$$

L'aggiunta del secondary path nella stima prevede una complicazione dell'algoritmo iniziale LMS, che si adatta nella sua forma più completa e maggiormente utilizzata chiamata Filtered-X LMS.
Il primo passo è cercare di stimare il secondo canale S in modalità offline rispetto al percorso primario: ciò significa porsi nella condizione di rumore nullo sul path primario, e di rumore bianco in S .
In questa fase verrà quindi stimata la risposta all'impulso $\hat{\mathbf{s}}(n)$ con l'algoritmo classico LMS, introducendo rumore bianco x nel secondary path tramite lo speaker ed utilizzando l'errore e(n) all'uscita per l'aggiornamento dei pesi del filtro.
L'equazione per l'aggiornamento dei pesi è:

$$\hat{s}_l(n+1) = \hat{s}_l(n) + \mu x(n-l)e(n) \text{ , per } l = 0,..L-1 \tag{2.10}$$

Una volta ottenuta una versione approssimativa di $\mathbf{s}(n)$ con $\hat{\mathbf{s}}(n)$, è possibile procedere la stima del primary path P secondo lo schema FxLMS.

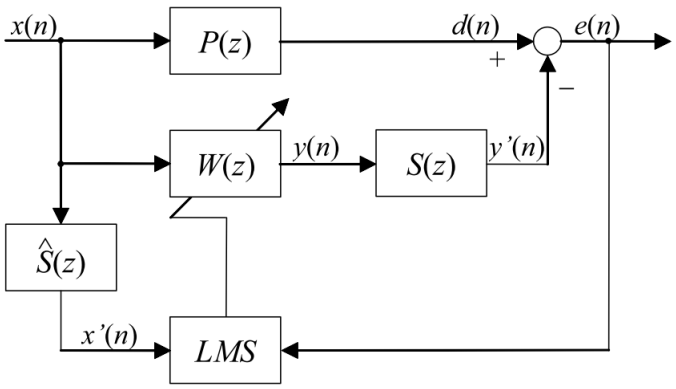

Figura 2.4: *Blocco FxLMS*

Rispetto all'algoritmo iniziale viene introdotto il blocco filtered x $\hat{S}(z)$ per il calcolo dei pesi del LMS. Riprendendo la formula generale per l'aggiornamento LMS:

$$w_l(n+1) = w_l(n) - \mu \frac{\partial e(n)}{\partial w_l(n)} e(n) \tag{2.11}$$

Il gradiente dell'errore per l'algoritmo FxLMS risulta:

$$\frac{\partial e(n)}{\partial w_l(n)} = -\mathbf{s}(n) * \mathbf{x}(n) = -\hat{\mathbf{s}}(n) * \mathbf{x}(n) = x'(n-l) \tag{2.12}$$

Ottenendo quindi lo schema rappresentato in figura con la nuova formula di aggiornamento dei pesi

$$w_l(n+1) = w_l(n) + \mu x'(n-l) e(n) \tag{2.13}$$

E' importante sottolineare come la presenza della stima di un canale aggiuntivo aumenta l'errore finale dell'algoritmo, ed in alcuni casi può introdurre stabilità; le condizioni di convergenza possono essere riassunte in:

- guadagno μ contenuto;
- l'errore in fase fra s(n) ed $\hat{s}(n)$ non eccede i 90°.

2.1.4 Algoritmi: FxLMS multicanale

Spesso il problema della riduzione del rumore risulta più complicato del caso in cui il problema sia circoscritto a condotti ristretti dato che i sistemi, tipicamente, presentano una risposta complessa e multimodale all'eccitazione primaria a cui sono soggetti. Per ottenere un'attenuazione globale del disturbo, tutti i modi strutturali ed acustici che si vogliono ridurre devono essere osservabili e controllabili. E' quindi necessario utilizzare sistemi di controllo automatico del rumore multicanale. Gli algoritmi proposti in precedenza per il caso di singolo canale possono essere tutti ricondotti al caso multicanale con l'unico sforzo di riscrivere le equazioni in maniera adeguata.
La versione multicanale più generale prevede J sensori di input, M punti in cui cancellare il disturbo, K speaker per la cancellazione.
Come si può notare dall'immagine lo schema è identico a quello del FxLMS

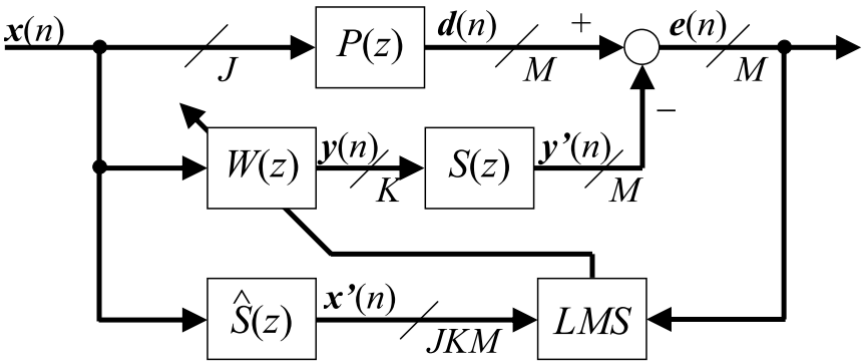

Figura 2.5: *Blocco FxLMS multicanale*

ma ora S(z), P(z) e W(z) rappresentano matrici di funzioni di trasferimento. Le equazioni sono le medesime del caso FxLMS, ma adattate a lavorare su matrici per comporre tutti i modi del suono, in tutti i punti in cui è necessaria la cancellazione. Di seguito l'elaborazione approfondita del caso 2x2x1.

Configurazione multicanale 2x2x1

Si analizza ora la configurazione con 2 canali, 2 punti di cancellazione ed un disturbo (che sarà poi utilizzata nel sistema da implementare).

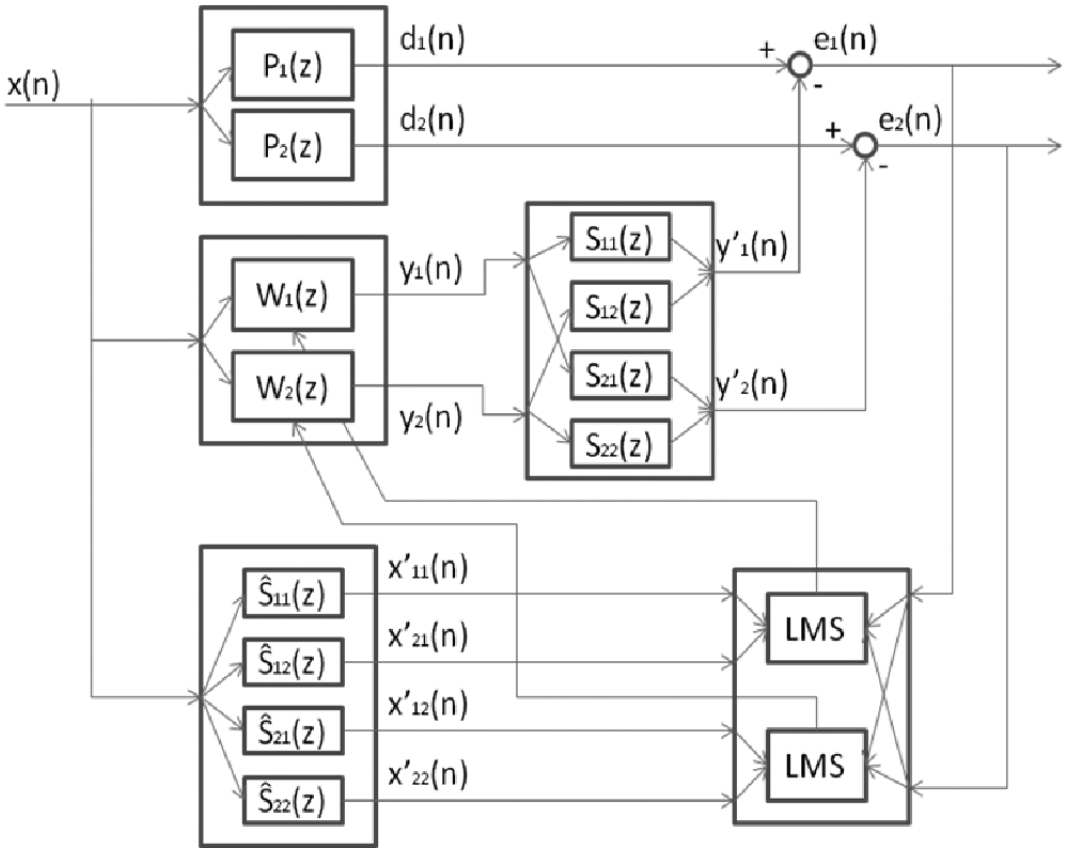

Figura 2.6: *FxLMS 2x2x1*

Lo schema mostra le matrici delle risposte all'impulso che determinano tutti i percorsi che segue il segnale dalla sorgente ai due punti di cancellazione.
Le equazioni che regolano l'algoritmo sono:

$$\textbf{Errore residuo} \quad e_1(n) = d_1(n) - y_1'(n) \qquad e_2(n) = d_2(n) - y_2'(n) \tag{2.14}$$

$$\textbf{Disturbi} \quad d_1(n) = x(n) * p_1(n) \qquad d_2(n) = x(n) * p_2(n) \tag{2.15}$$

$$\textbf{Segnale nullificatore} \quad \begin{aligned} y_1'(n) &= s_{11}(n) * y_1(n) + s_{12}(n) * y_2(n) \\ y_2'(n) &= s_{21}(n) * y_1(n) + s_{22}(n) * y_2(n) \end{aligned} \tag{2.16}$$

$$\textbf{Filtraggio x} \quad \begin{aligned} \mathbf{x'_{11}}(n) &= s_{11}(n) * \mathbf{x}(n) \qquad \mathbf{x'_{12}}(n) = s_{21}(n) * \mathbf{x}(n) \\ \mathbf{x'_{21}}(n) &= s_{12}(n) * \mathbf{x}(n) \qquad \mathbf{x'_{22}}(n) = s_{22}(n) * \mathbf{x}(n) \end{aligned} \tag{2.17}$$

Le equazioni per l'aggiornamento dei pesi sono simili a quelle del FxLMS a singolo canale, modificate per tenere conto dell'effetto congiunto dei due percorsi del segnale audio:

$$\begin{aligned} \mathbf{w_1}(n+1) &= \mathbf{w_1}(n) + \mu[\mathbf{x'_{11}}(n)e_1(n) + \mathbf{x'_{12}}(n)e_2(n)] \\ \mathbf{w_2}(n+1) &= \mathbf{w_2}(n) + \mu[\mathbf{x'_{21}}(n)e_1(n) + \mathbf{x'_{22}}(n)e_2(n)] \end{aligned} \tag{2.18}$$

2.2 Hardware FPGA per sistemi digitali

Per la realizzazione del sistema si ha a disposizione una scheda HDR 60 della Lattice, la cui peculiarità è di avere risorse ottimizzate per l'elaborazione video. Questa piattaforma di sviluppo contiene una FPGA ECP3 della Lattice, la quale sarà il nucleo per la logica del sistema e conterrà l'algoritmo FxLMS per la riduzione del rumore. La famiglia Lattice ECP3 è particolarmente adatta alla nostra applicazione in quanto offre ottime capacità computazionali per DSP:

- É predisposta con architetture per operazioni di tipo Multiply&Accumulate, specifiche per la convoluzione dei segnali;
- Integra un'alta memoria al suo interno (149K LUT) per la bufferizzazione dei risultati;
- Presenta il consumo più basso per prestazioni offerte tra quelle presenti sul mercato.

2.2.1 Schede FPGA

Una scheda FPGA, *Field Programmable Gate Array* , è un circuito integrato progettato per essere configurato ripetutamente dall'utente, o dal progettista, in seguito alla sua manifatturazione (da li il termine Field Programmable, programmabile sul campo).
La configurazione di una FPGA è solitamente specificata utilizzando un linguaggio di descrizione hardware (HDL), similmente a quello utilizzato per i dispositivi application specific (ASIC): inizialmente per la realizzazione di un ASIC occorreva descrivere il diagramma del circuito, ora questa pratica sta diventando sempre più rara.
Un dispositivo FPGA oggigiorno può implementare qualsiasi funzionalità logica che possa essere implementata su un circuito specifico ASIC.
Essa però offre notevoli vantaggi a fronte di una applicazione special purpose: possibilità di aggiornare la configurazione della funzionalità dopo la realizzazione; abbassamento dei costi non ricorrenti di ingegnerizzazione[10]; costo finale per unità prodotta minore.
Le board contengono dei componenti logici programmabili chiamati logic blocks e una gerarchia di connessioni riconfigurabili che permettono l'interconnessione dei blocchi logici in diverse configurazioni. I blocchi logici possono essere codificati per sintetizzare sia funzioni combinatorie complesse, che semplici operazioni di logica binaria; ogni blocco contiene anche elementi di memoria, tipicamente flip flop, al fine di permettere la realizzazione di blocchi sequenziali.

Sviluppi moderni

Il recente trend di sviluppo è stato portare il livello di genericità dell'architettura FPGA un passo oltre, aggiungendo ai blocchi logici e alle interconnessioni già esistenti dei microprocessori embedded e periferiche avanzate, per creare un sistema completo.
I primi esempi di architetture ibride possono essere trovati nelle board Xilinx Virtex-II PRO e Virtex-4, le quali includono uno o più processori PowerPC interconnessi nell'architettura tradizionale.
Dal 2010 viene introdotta sul mercato una piattaforma di processing estendibile, integrante nella logica un microcontroller ARM(processore a 32 bit con memoria e IO): con l'aggiunta di un tale dispositivo di calcolo è possibile implementare funzioni sempre più complesse, a calcolo seriale e parallelo, che portarono alla realizzazione dei più moderni sistemi embedded.

Architettura

L'architettura standard [12] di una board FPGA è costituita da un array di blocchi logici, interfacce di input-output e canali per l'istradamento delle interconnesioni. Durante l'implementazione di un nuovo design hardware, è

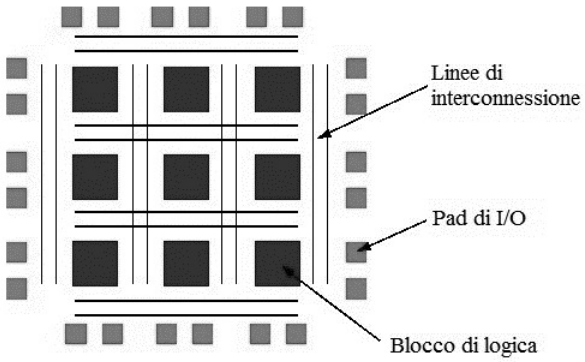

Figura 2.7: *Architettura FPGA*

buona norma che lo schema circuitale sia mappato su FPGA utilizzando il numero adeguato di risorse: se da un lato il numero di blocchi funzionali e degli I/O può essere facilmente determinato in base al design, dall'altro lato il numero delle piste per l'instradamento dei segnali può variare di molto anche con la stessa logica. Dato che le tracce non utilizzate incrementano il costo (e diminuiscono le performance) , durante la manifattura di una scheda si cerca di fornire un numero bilanciato di connessioni, in maniera tale da essere sufficienti per la maggior parte dei design in termini di collegamenti inter-LUT e IO.

La struttura generica di un blocco logico, unità fondamentale di calcolo per

una board, è composta da un set di LUT a 4 input, un Full Adder e un flip flop D per l'output.

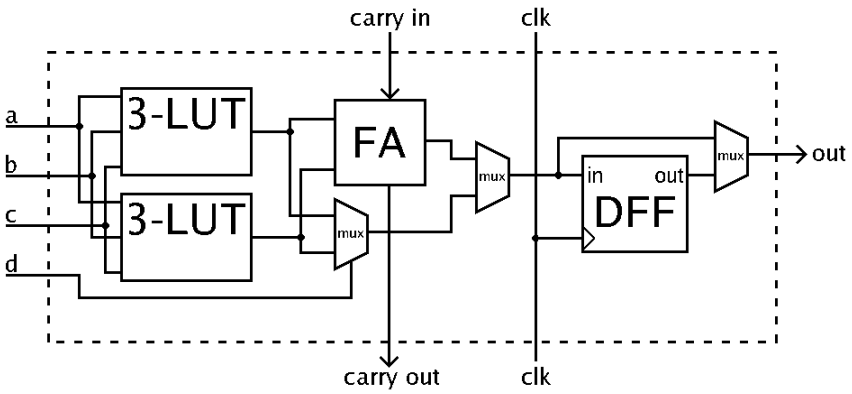

Figura 2.8: *Blocco di logica*

Un blocco di logica come quello mostrato in figura presenta due modalità di operazione: logica e aritmetica. In modalità logica i segnali di input vengono combinati nelle LUT di ingresso tramite il mux sinistra; in modalità aritmetica gli output vengono inseriti nell'unità di calcolo full adder. La scelta di quale modalità utilizzare è selezionata dal multiplexer presente al centro.
L'output può essere di tipo sincrono o asincrono, in base alla programmazione dell'ultimo multiplexer sulla destra: nel caso asincrono verrà presa l'uscita diretta del blocco di logica, nel caso sincrono l'uscita del flip flop.

Programmazione e design

Per definire il comportamento funzionale di una board FPGA, il programmatore deve fornire un codice descrittivo dell'hardware (in formato VHDL-Verilog), o uno schema a blocchi. La descrizione tramite HDL è più adatta per progettare strutture di grandi dimensioni in quanto dà la possibilità di istanziare dinamicamente set di componenti (specificando architetture generiche) al posto di porli singolarmente nel design.
Una volta scritto il codice, viene generata una netlist mappata sulla tecnologia FPGA tramite tool di design automatizzato: questo schema riassume l'insieme dei componenti necessari e il loro collegamento in un formato comprensibile al calcolatore.
Il processo di place&route riconfigura la netlist per l'architettura FPGA in cui verrà implementato: per questo motivo è eseguito tramite i software proprietari delle stesse case produttrici della board.
Alla fine di questo processo si ha a disposizione l'insieme dei blocchi fisici che verranno utilizzati, in particolare delle specifiche tempistiche di hold e setup per

ciascun componente. L'architettura viene quindi simulata a livello di timing per verificare che i vincoli temporali necessari per il corretto processamento dei segnali siano soddisfatti.
Completato e validato il processo di design lo step finale consiste nel generare un file binario per riconfigurare la scheda FPGA, attraverso la programmazione con interfaccia seriale (JTAG) della sua memoria interna.

2.3 Sistemi e simulazioni HIL

L'acronimo Hardware-In-the-Loop (HIL) definisce la realizzazione di un sistema hardware/software per la simulazione e la validazione di dispositivi elettronici di tipo special purpose.
Il suo scopo è inviare al componente hardware che si sta analizzando degli input virtuali creati a computer, che riproducano gli stimoli dei sensori come in un ambiente reale di utilizzo. Una simulazione quindi che permetta di verificare tutte le funzionalità in anello chiuso, ma in condizioni ideali: i segnali di input vengono scelti e trasmessi da un componente software e non prelevati dai sensori; allo stesso modo gli output hardware sono inviati al computer al posto di essere rilevati dal comportamento degli attuatori del sistema.
La funzionalità viene di conseguenza verificata isolando il nucleo del sistema da possibili malfunzionamenti esterni, dati dal reparto di trasduzione dei segnali dell'ambiente di funzionamento reale, disturbi nei sensori o negli attuatori.

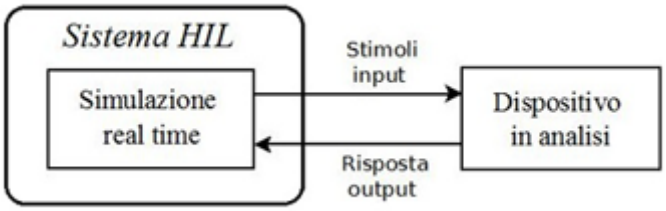

Figura 2.9: *Sistema HIL*

2.3.1 Storia

L'utilizzo della tecnica di simulazione HIL è stato introdotto negli ultimi due decenni e trova le sue radici nell'industria della locomozione e areonautica. Per arrivare alle motivazioni che hanno portato alla sua nascita partiamo da un esempio, comparando un generico motore automobilistico odierno e uno di 40 anni fa.
In particolare concentriamo l'attenzione sul controllo di tale motore. Nell'esemplare più datato erano presenti sostanzialmente due controlli di basso livello:

uno per la gestione della miscelazione benzina aria, mentre il secondo per regolare il calore di emissione del carburante. Le aspettative per tale motore erano sincronizzate con l'epoca e non sono lontanamente paragonabili a quelle per una vettura attuale: il mercato di oggi presenta una richiesta di motori piccoli ma con prestazioni elevate, a basso impatto ambientale ma con alta reattività, con caratteristiche innovative ma a basso costo. La parte relativa al controllo è cresciuta enormemente, introducendo dozzine di controllori complessi per gestire i nuovi aspetti introdotti: l'iniezione controllata del carburante, la risposta dell'acceleratore, il rumore del motore, il controllo del consumo e delle emissioni. Allargando il sistema alla struttura completa dove il motore è utilizzato (macchine, treni, camion, aerei, navi) il numero di dispositivi embedded di controllo aumenta ulteriormente, in maniera esponenziale.
Si evince un aumento della complessità di realizzazione di tali strutture, correlata ad un aumento del tempo per i test antecedenti al lancio sul mercato, in condizioni di garanzia e sicurezza.
Per la realizzazione di un nuovo progetto, 20 anni fa un team di pochi ingegneri avrebbe inizialmente dedicato gran parte del tempo per la costruzione del prototipo fisico e, a lavoro concluso, avrebbe scritto poche semplici routine per il sistema di controllo. Il prototipo sarebbe stato utilizzato per completare con successo tutti i test di funzionalità: una possibile rottura dei prototipi era plausibile e già contata nel budget di realizzazione.
Questo passaggio ora non è più possibile data la complessità raggiunta per la manifattura dell'hardware costitutivo: il costo e il tempo di realizzazione compresi nel budget non permettono la costruzione, e in maggior ragione della distruzione, del prototipo fisico prima del controllo. Difatti, se tutte le parti costitutive di un aereo dovessero essere fisicamente costruite e testate per poi essere assemblate e testate nuovamente nella struttura completa, il time to market e il costo sarebbero proibitivi. Le nuove schedule di sviluppo sono diventate più costrittive, e non includono la costruzione di un prototipo prima di iniziare il design di sistemi embedded per il testing. system.
Nei processi di sviluppo odierni è stato quindi introdotto il concetto di simulazione HIL in concomitanza con lo sviluppo dell'impianto: durante il tempo necessario per la prototipazione dell'impianto, il 90% dei test saranno già stati completati con simulazioni HIL.

2.3.2 Confronto fra simulazioni HIL e prototipazione

Riassumendo, perchè risulta migliore verificare sistemi hardware tramite simulazioni HIL e non tramite la connnesione diretta nell'impianto reale di funzionamento.
Secondo logica il metodo più efficace per sviluppare un sistema embedded è quello di collegarlo direttamente all'impianto di utilizzo, se questo esiste.

Per capire la necessità e comparare l'efficacia di un sistema HIL occorre valutare le seguenti metriche:

- Costo
- Tempo
- Sicurezza

Il costo di un approccio rappresenta la somma del costo degli strumenti e del lavoro uomo necessario: l'impianto prototipato, di norma, rappresenta un costo maggiore sia in termini di strumenti (e materiali) che di ore uomo rispetto a un simulatore ad alta fedeltà: risulta più economico sviluppare e testare tramite connessione con un framework HIL rispetto all'impianto effettivo. Ad avvalorare questa affermazione si porta l'esempio di come per la realizzazione di controlli per motori di aerei, il costo effettivo sfruttando tecniche HIL è più piccolo di un ordine di grandezza rispetto alla prototipazione.

La tecnica HIL permette il design e il testing del sistema di controllo in parallelo alla realizzazione dell'impianto, riducendo il tempo totale di manifattura e anticipandone il lancio sul mercato. Al contrario, seguendo l'ottica alternativa occorrerebbe prima realizzare il prototipo e poi testare il sistema di controllo: compiendo queste azioni sequenzialmente e non parallelamente si capisce come il tempo richiesto con la prototipazione risulti maggiore. Il secondo vantaggio temporale dell'approccio HIL è la possibilità di automatizzare l'esecuzione di test, permettendo di arrivare celermente ad un corretto profilo del dispositivo scrivendo delle routine nei più comuni linguaggi di programmazione.

Il fattore sicurezza è tipicamente associato sia alla misura di costo che alla misura di tempo: realizzare un prototipo che prevede input sensoriale umano necessità di adeguate misure di sicurezza prima di essere testato, aumentando entrambe le misure di costo e tempo.
In questo contesto, iniziando le procedure di test con delle simulazioni HIL, si garantisce affidabilità e maggiore sicurezza per gli operatori che utilizzeranno il dispositivo nelle fasi di validazione: la risposta del sistema sarà già stata ampiamente validata tramite simulazione.

Capitolo 3

Analisi e implementazione

Questo capitolo tratta in maniera approfondita i dettagli implementativi per la configurazione della scheda FPGA e per il completamento del framework di simulazione hardware in the loop, descritti nei primi due capitoli.
Partendo da una visione generale dell'architettura dell'intero progetto verranno poi elaborate le parti funzionali all'interno del capitolo, analizzando con precisione le problematiche incontrate.

3.1 Analisi del sistema

Durante l'analisi di alto livello della struttura del progetto sono emersi due macropassaggi, da realizzare con uno specifico ordine cronologico: il primo passo consiste nella codifica dell'algoritmo di ANC sull'hardware, ovvero la sua implementazione in un linguaggio di descrizione hardware (VHDL) in blocchi funzionali; completato il blocco algoritmico la parte seguente elabora il test del sistema, realizzando un framework che permetta la simulazione hardware in the loop di una (generica) scheda FPGA. Partendo da questa considerazione, le due fasi presentate sono state ulteriormente sezionate in sotto-passaggi chiave, qui presentati secondo l'ordine logico di realizzazione.
L'implementazione in linguaggio hardware necessita in primo luogo di ristrutturare l'algoritmo per la cancellazione del rumore, in modo tale da essere eseguito su un dispositivo embedded a precisione finita. In questo passaggio si presta particolare attenzione ai problemi legati alla rappresentazione dei dati, che può portare alla non convergenza delle iterazioni.
Adattato l'algoritmo è possibile codificare, nel senso stretto del termine, il componente hardware che realizza l'algoritmo di controllo attivo del rumore. In questa fase risulta fondamentale designare una struttura a blocchi generici, in modo tale che, una volta operativa, possa lavorare in modalità online con dati prelevati dai sensori nell'ambiente o in modalità debug con dati virtuali da HIL.

Completato questo step si entra nel cuore del progetto per la connessione della board FPGA in anello chiuso con il computer.
Come primo passo è necessario stabilire il collegamento fisico ottimale per fornire gli stimoli alla scheda. La scelta deve ricadere su un link che garantisca una banda tale da eseguire una simulazione con le specifiche di alta fedeltà audio (HiFi), in un intervallo di tempo ragionevole.
Stabilito il protocollo fisico occorre scegliere uno standard di comunicazione per la trasmissione peer to peer, restando nell'ottica di assenza di errori nei dati. Per questo motivo sarà scelto un protocollo che permetta la gestione della trasmissione attraverso un bus di controllo specifico, affiancato al bus dei dati.
L'ultimo passaggio per la realizzazione della pila protocollare è rappresentato da una scelta adeguata del protocollo applicativo, ovvero delle modalità di organizzazione dei dati e dei comandi di simulazione.
A questo punto il design dell'architettura di comunicazione è completo e si può procedere alla scrittura del blocco hardware per la trasmissione dati; il protocollo codificato in VHDL seguirà specifiche del livello fisico e applicativo concordato nel passaggio precedente, realizzando un collegamento punto punto tra FPGA e computer. Un aspetto importante in questa fase è quello di gestire opportunamente i dati della trasmissione via hardware, per comunicarli al reparto dell'algoritmo evitando problemi di timing e metastabilità.
La simulazione HIL viene infine effettuata programmando adeguatamente il master della comunicazione, rappresentato da un computer notebook. A questo proposito viene creato un software per la gestione che possa offrire funzionalità di configurazione, esecuzione, consegna dei risultati. In particolare tramite interfacciamento del software con un ambiente matematico si può facilmente creare dei segnali di stimolo per la board e concludere con la presentazione/-verifica dei risultati in formato visivo.
A questa analisi iniziale viene fatta seguire l'elaborazione di tutti i punti elencati, presentati secondo l'ordine temporale in cui sono stati descritti.

3.2 Implementazione dell'algoritmo su hardware

In questa sezione vengono presentati il design e la codifica in linguaggio hardware dell'algoritmo FxLMS descritto nel secondo capitolo.
La prima parte analizza le problematiche relative alla convergenza dell'algoritmo con un dispositivo a logica finita.
La seconda parte introduce e descrive i blocchi hardware costitutivi della versione FxLMS.

3.2.1 Ristrutturazione FxLMS

Durante la trattazione teorica dell'algoritmo FxLMS i segnali in gioco sono numeri reali, i quali per definizione sono costituiti da un numero indefinito di cifre decimali e permettono di avere una precisione infinita.
Per un utilizzo digitale di tali segnali occorre quantizzare i dati ed utilizzare una risoluzione finita, data dalla precisione macchina sulla quale vengono eseguiti i calcoli.
Implementando l'algoritmo in matlab si sfrutta la rappresentazione dei dati in floating point (ovvero a virgola mobile), gestita internamente dall'ambiente di calcolo. Rispetto alla versione teorica si ottengono delle performance dipendenti dal grado di fedeltà dei coefficienti, e quindi dalla precisione con la quale vengono memorizzati dal programma. A questo proposito in matlab, digitando eps, si ottiene la precisione usata del sistema per i calcoli in floating point, pari a $(2.22 * 10^{-16})$: le simulazioni del FxLMS a PC hanno mostrato come questo numero permette di memorizzare i dati con sufficiente risoluzione, consentendo la convergenza dei pesi e quindi dell'algoritmo.
Passando ad un'implementazione su sistema DSP o FPGA, la realizzazione di un algoritmo in floating point risulta onerosa sia in termini di complessità delle risorse, che in termini di velocità e potenza consumata [9]. E' stata quindi preferita una rappresentazione in fixed point (virgola fissa) dei dati, memorizzati in registri a dimensione fissa di 16 bit.

Rappresentazione fixed point e algebra

Molti processori di calcolo non supportano la gestione floating point dei dati per motivi di risorse e complessità: serve quindi rivedere l'algoritmo FxLMS originale e ricorrere all'algebra per l'elaborazione in fixed point.
Come suggerisce il nome, un numero in fixed point pone la virgola che separa la parte intera dai decimali in una locazione fissa degli n bit totali. Ad esempio, con la configurazione 8.8 si hanno 8 bit di parte intera e 8 di parte decimale: con parte decimale fissa in 8 bit, è facilmente calcolabile quale sarà il range e la precisione dei dati utilizzati (256 livelli).
Nella nostra applicazione la rappresentazione scelta per i dati è su una struttura

a 16 bit secondo lo schema 0.16 dove tutti i bit descrivono la parte frazionaria. Per la natura dei calcoli eseguiti occorre descrivere i dati con segno, imponendo una memorizzazione in complemento a due del dato. Il numero di interi con segno rappresentabili con 16 bit va da -32768 (-2^{15}) a 32767 ($2^{15}-1$): passando alla rappresentazione frazionaria da noi utilizzata si ottiene un range da -1 a 0.9999 con passo $2^{-15} = 0.000030$.

Implementazione da floating point a fixed point

Nell'algoritmo FxLMS identifichiamo due parti di calcolo strutturale: filtraggio ed adattamento. La parte di adattamento dei pesi è la più sensibile delle due in quanto, in presenza di errori di approssimazione specialmente con numeri vicini allo 0, si può ottenere uno stallo dei pesi: questo significa che il procedimento di update si ferma prima della convergenza ottima per quella precisione. Per diminuire il rischio di stallo [9] propone l'utilizzo di un formato a 32 bit dei pesi, organizzati secondo il formato:

$$w_n = 2^{16} w_n^{hi} + w_n^{lo} \tag{3.1}$$

con w_n^{hi}, w_n^{lo} definiti a 16 bit.
Con questa rappresentazione il filtraggio lavora ancora a 16 bit, sfruttando i 16 intermedi dei 32 totali ottenuti con

$$y(n) = \left\lfloor 2^{-8} w_n^{hi^T} x_n \right\rfloor \tag{3.2}$$

la parte di aggiornamento pesi sfrutta invece tutti e 32 bit, permettendo una maggiore precisione e una minore probabilità di stallo dei pesi. L'equazione per l'aggiornamento dei pesi utilizza l'intero w_n

$$w_{n+1} = w_n + \mu_q e(n) x_n \tag{3.3}$$

Per aggiustare il guadagno nella parte esatta del dato il guadagno viene shiftato di 8 bit:

$$\mu_q = \lfloor 2^8 \mu \rfloor \tag{3.4}$$

3.2.2 Progetto VHDL: architettura

In figura 3.1 viene mostrato lo schema a blocchi per implementare l'algoritmo FxLMS, facente riferimento al caso 2x2x1: un unico microfono di riferimento, due altoparlanti e due microfoni d'errore.

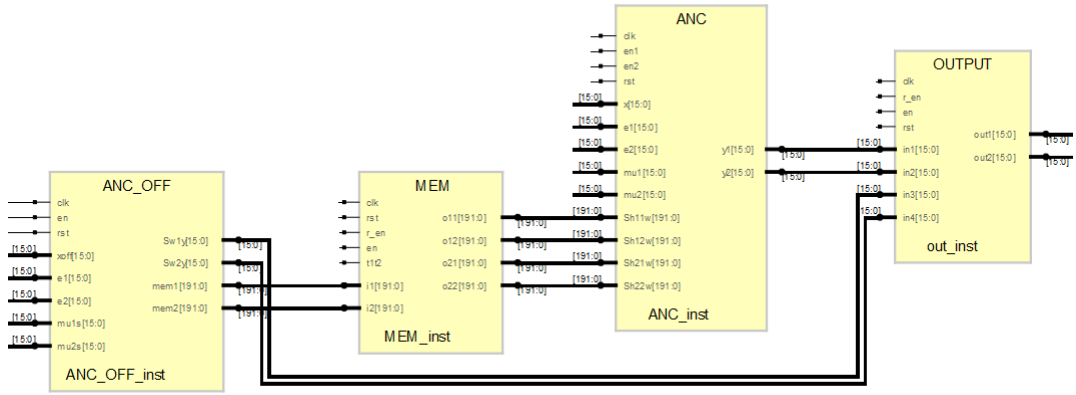

Figura 3.1: *Schematico del progetto VHDL - visione TOP*

Il progetto è diviso in 2 macroblocchi che lavorano in alternanza tra di loro e da alcuni blocchi di contorno necessari per la gestione dei dati e dei risultati.
I segnali audio di input per l'entità di alto livello sono dati in modalità online dal segnale misurato dal microfoni di riferimento/errore, mentre in modalità HIL dagli stimoli inviati dal computer. Sempre in ingresso sono presenti quattro segnali per la gestione della simulazione e della sincronizzazione: i guadagni del metodo sia per la parte di algoritmo LMS offline sia per quello online; un segnale di clock; un segnale di enable; un segnale di reset; un segnale di switch tra algoritmo offline e algoritmo online.
Gli output del blocco sono i segnali di correzione destinati in modalità online ai due altoparlanti, mentre in modalità offline al blocco per il calcolo dell'errore residuo.
Come descritto nel capitolo 2, il metodo LMS necessita di conoscere alcuni parametri (quelli del filtro $\hat{S}$ che in linea teorica dovrebbe cercare di essere una copia perfetta del filtro S) e per far ciò viene applicato lo stesso algoritmo LMS ad una sorgente di rumore bianco: questa prima fase, detta offline, consente di stimare i parametri necessari al metodo per funzionare ed è identificata dal blocco ANC_OFF; una volta terminata questa fase il sistema può entrare in modalità online e cercare di ridurre il rumore indesiderato; questa fase è identificata dal blocco ANC.
I due blocchi funzionali rimanenti sono utilizzati per la gestione dell'interna struttura: in modo particolare MEM consente di memorizzare i parametri sti-

mati in modalità offline e poterli riutilizzare online, mentre OUTPUT gestisce i segnali di uscita diretti agli altoparlanti.

3.2.3 Blocchi di lavoro: algoritmo LMS

I blocchi di lavoro sono i blocchi principali del sistema; essi contengono al loro interno lo sviluppo dell'algoritmo LMS e vengono utilizzati per minimizzare i parametri di errore per poi stimare i nuovi coefficienti del filtro. I due macroblocchi principali sono ANC_OFF che si occupa della stima preliminare dei parametri di filtraggio necessari in input al successivo macroblocco di ANC: il suo compito è la stima del canale fisico relativo al percorso P(z) compiuto dal segnale di rumore. Entrambi questi macroblocchi utilizzano dei blocchi fondamentali che sono in comune tra i due o comunque presentano una struttura simile. In questa sezione viene presentata una visione di insieme dei due macroblocchi principali e una descrizione dei singoli blocchi fondamentali, almeno per quanto riguarda la loro funzionalità.

ANC_OFF

Il blocco ANC_OFF ha il ruolo di utilizzare l'algoritmo LMS per poter stimare i parametri di filtraggio di S(z) sconosciuti al sistema: la stima del cammino primario compiuto dal rumore necessita di conoscere le caratteristiche del percorso compiuto dal segnale in uscita dal sistema fino al suo rientro attraverso i microfoni d'errore. Queste caratteristiche vengono sintetizzate da un filtro $\hat{S}(z)$, compito del macroblocco è quindi quello di calcolare i parametri di questo filtro. Per far ciò l'algoritmo LMS viene applicato per la minimizzazione dell'errore nella stima di questi parametri. In ingresso a questo macroblocco

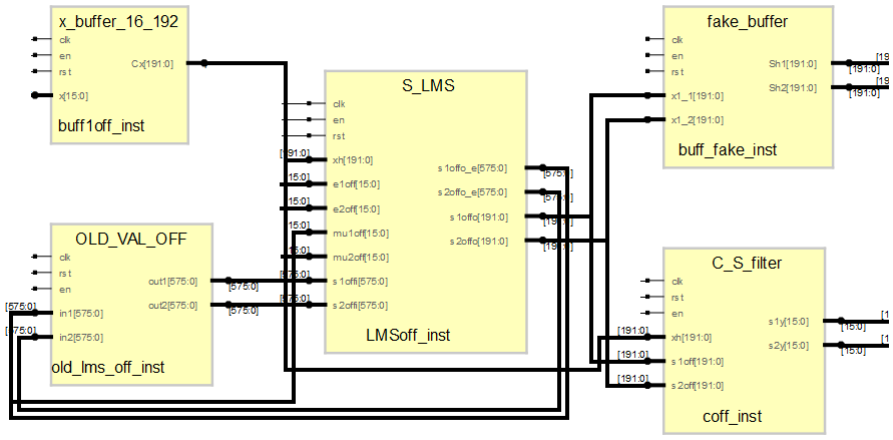

Figura 3.2: *Blocco Macroblocco ANC_ OFF*

ci sono il segnale sorgente del rumore bianco generato per la fase di training, i segnali acquisiti dai due microfoni di errore, le costanti del metodo LMS, un bit di clock, uno di enable e un reset. Le uscite del blocco saranno gli N parametri del filtro per ciascuno dei quattro filtri rappresentanti le quattro possibili combinazioni di cammino tra ciascuno dei due altoparlanti e ciascuno dei due microfoni. All'interno del blocco ANC_OFF sono presenti dei blocchi funzionali che rappresentano le varie sezioni e le varie funzionalità dell'algoritmo: nello specifico sono presenti un blocco che esegue la funzionalità di filtraggio, un blocco che rappresenta il "core" dell'algoritmo LMS, blocchi che svolgono il semplice compito di bufferizzazione e un blocco per la semplice memorizzazione di determinati valori che dovranno essere riutilizzati ricorsivamente.

ANC

Il blocco ANC entra in funzione una volta terminata la prima fase di stima dei parametri del filtro S(z): si occupa di eseguire il lavoro di stima del canale primario per poter ricreare un segnale identico al rumore generato ma in opposizione di fase; sommandolo poi al segnale originale verrà diminuita la sua intensità fino ad una ideale cancellazione dello stesso.
Il blocco utilizza come sorgenti esterne i segnali acquisiti dai vari microfoni con lo scopo di minimizzare l'errore di stima in funzione del rumore di ingresso. In ingresso a questo macroblocco ci sono il segnale sorgente del rumore da

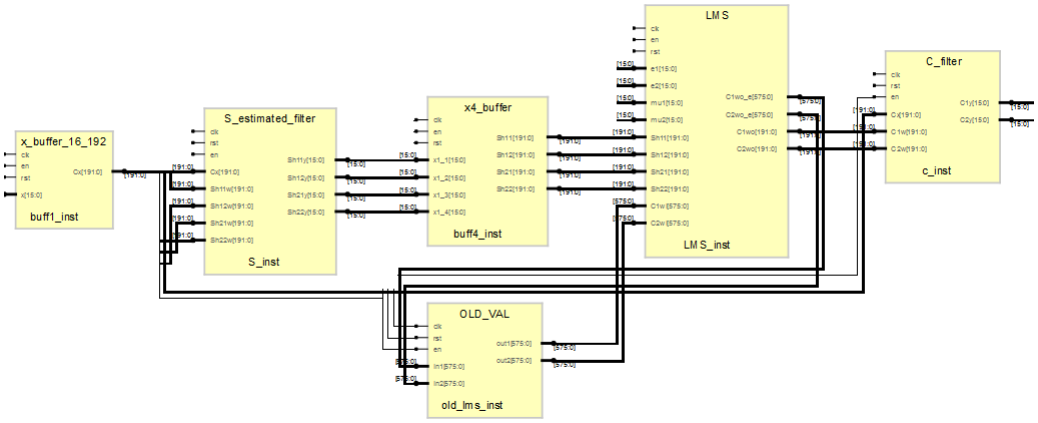

Figura 3.3: *Blocco Macroblocco ANC*

cancellare, i segnali acquisiti dai due microfoni di errore, le costanti del metodo LMS (non per forza identiche a quelle del caso offline), un bit di clock, uno di enable, un reset e i parametri calcolati durante la fase offline di stima del filtro. Le uscite del blocco saranno gli output forniti a ciascuno dei due altoparlanti rappresentanti la stima dei cammini compiuti dalla sorgente di

rumore per giungere a ciascuno dei microfoni di errore. All'interno del blocco ANC, come nel macroblocco precedente, sono presenti dei blocchi funzionali che rappresentano le varie sezioni e le varie funzionalità dell'algoritmo: nello specifico sono presenti due blocchi che eseguono la funzionalità di filtraggio, un blocco che rappresenta il "core" dell'algoritmo LMS, blocchi che svolgono il semplice compito di bufferizzazione e un blocco per la semplice memorizzazione di determinati valori che vengono riutilizzati durante questa fase.

Blocchi fondamentali

I blocchi fondamentali rappresentano le funzionalità base a cui gli altri blocchi (ANC e ANC_OFF) fanno riferimento: in modo particolare essi forniscono la funzionalità di filtraggio, memorizzazione, bufferizzazione e calcolo dei pesi secondo l'algoritmo di minimizzazione LMS.

Blocco x_buff

Il blocco x_buff rappresenta un incrocio tra una FIFO ed un multiplexer in cui il segnale d'ingresso viene letto e memorizzato all'interno di una coda. L'output sarà un segnale rappresentante il valore di input allo stato attuale e quello agli N-1 istanti passati. Oltre a questi valori in ingresso sono presenti il segnale di clock, il reset e il segnale di enable. Diversi blocchi x_buff possono essere raccolti in un unico blocco aggregativo in base ai segnali che si hanno in ingresso: per esempio il blocco X4_BUFFER raccoglie al suo interno 4 singoli blocchi x_buff ciascuno dei quali memorizza e restituisce il valore attuale e gli N-1 valori passati dei singoli segnali.

Blocco LMS e S_LMS

Il blocco LMS ha il compito di aggiornare i pesi del filtro di stima del canale primario.

Gli ingressi del blocco vengono dati dagli errori misurati dai microfoni d'errore, dalle costanti del metodo, e dalle uscite del filtro di stima dei percorsi tra gli altoparlanti e i microfoni d'errore; come ulteriori ingressi vengono considerati i valori delle uscite che vengono retroazionati e utilizzati come valore di base a cui sommare o sottrarre l'errore commesso. In aggiunta sono presenti anche tre bit di controllo: clock, enable e reset. Le uscite, e di conseguenza gli ingressi in retroazione, corrispondono ai pesi del filtro che cerca di stimare al meglio il percorso del cammino primario. La differenza tra i blocchi LMS e S_LMS risiede semplicemente nel numero di operazioni interne da eseguire, senza togliere il fatto che la tipologia è la medesima per tutti i blocchi che utilizzano il componente lms.

Blocco filter (C_FILTER)

Il blocco filter rappresenta un semplice blocco di filtraggio in cui si esegue una convoluzione tra i vettori di dati in ingresso. Oltre ai bit di gestione quali l'enable e il clock, questo componente riceve in ingresso i segnali da convolvere tra di loro (quello proveniente dal microfono di ingresso e i pesi calcolati dall'algoritmo). In uscita viene presentato il risultato di tale operazione. La differenza tra i vari blocchi che utilizzano questo componente è data semplicemente dal nome ed eventualmente dal numero di sorgenti in entrata e conseguentemente dal numero di risultati in uscita.

Blocco old_val (OLD_VAL e OLD_VAL_OFF)

Il blocco old_val ha il semplice compito di memorizzare i valori in uscita dai blocchi lms (i pesi del filtro di stima) e riproporli agli stessi blocchi in qualità di ingressi da correggere con l'algoritmo adattativo. Oltre ai segnali veri e propri che vengono memorizzati e riproposti in uscita, vengono utilizzati i soliti bit di gestione per il componente: enable e clock. La differenza tra i diversi blocchi che utilizzano questo tipo di componente sta semplicemente nel numero di input e, di conseguenza, degli output.

3.2.4 Blocchi di gestione

I blocchi di gestione sono quei particolari blocchi all'interno del progetto che non hanno un vero e proprio ruolo computazionale ma servono per gestire lo scambio di dati e il passaggio tra una prima fase offline, in cui il sistema determina i parametri per il lavoro, e una successiva fase online in cui il sistema esegue l'algoritmo per la cancellazione vera e propria del rumore.

MEM

Il blocco MEM presenta come ingressi i segnali relativi ai parametri di filtraggio calcolati dall'algoritmo offline, un bit di clock, uno di reset e uno di switch tra la fase di lettura (dei dati memorizzati) e quella di scrittura. L'output del blocco è dato dai parametri di filtraggio nel caso in cui il bit di switch sia impostato per la fase di lettura, altrimenti è posto a zero. Come già accennato questo blocco ha il compito di memorizzare i parametri calcolati durante la fase offline e ripresentarli durante la fase di cancellazione del rumore fornendoli al blocco che si occuperà della fase online. Lo switch tra queste due fasi viene gestito dal bit r_en: quando questo indica la fase di memorizzazione i parametri in ingresso vengono memorizzati in variabili temporanee mentre quando il bit indica la fase di lettura i parametri vengono associati da queste variabili alle corrispondenti uscite e quindi riutilizzate dall'algoritmo LMS.

OUTPUT

Il blocco OUTPUT presenta come ingressi i segnali in uscita dal blocco ANC, il clock di sistema, il bit di reset e il bit che consente lo switch tra la fase offline e la fase online in cui l'algoritmo LMS cerca di compiere il proprio dovere. L'uscita di questo blocco è rappresentata dai segnali che verranno condotti agli altoparlanti atti a generare il segnale in opposizione di fase per la cancellazione del rumore. Il funzionamento principale del blocco è quello di gestire le uscite del sistema in funzione della fase in cui esso si trova: nella fase offline in cui il sistema esegue il training per il calcolo dei parametri di filtraggio l'uscita verso gli altoparlanti sarà disabilitata (l'uscita sarà impostata con tutti i valori pari a zero), mentre nella fase online in cui il sistema lavora sulla cancellazione del rumore all'uscita sarà associato il valore stimato per la cancellazione e calcolato dall'algoritmo.

3.3 Simulazione HIL

3.3.1 Collegamento della scheda

Come riportato dall'analisi, in primo luogo è necessario stabilire il collegamento fisico fra la board FPGA ed il notebook che eseguirà il test. In questo contesto il problema principale è assicurare un link ad una velocità adeguata per le operazioni da eseguire, che possa quindi completare delle simulazioni in un tempo ragionevole e senza perdita di dati.
La tabella mostra le caratteristiche di un segnale audio generico per essere utilizzato come ingresso in simulazioni HIL ad alta fedeltà (HiFi).

Requisito	Valore
Campionamento audio	48Khz
Risoluzione audio	16 bit

Per chiudere l'anello di simulazione occorre inviare all'algoritmo nella scheda un canale con il segnale audio x e ricevere 2 canali audio per gli errori residui: in totale 3 canali. La banda richiesta è pari a 768 Kb/s per ogni canale, data dalla moltiplicazione della risoluzione richiesta per ciascun dato e dal campionamento. . Partendo dallo schema di canali descritto, pari a 1 input e 2 output, si ottiene una banda totale di 3 Mbit/s, su canale bidirezionale.
Per realizzare il collegamento sono state vagliate diverse alternative, tra cui:

- Testing da una seconda FPGA, con collegamento a bus
- Testing da computer tramite una scheda multifunzione DAQ
- Testing da computer con la scheda PCI-express
- Testing da computer tramite link USB 2.0 - segnali UART con adattatore

Utilizzando una board FPGA aggiuntiva è possibile realizzare il test HIL in modalità completamente indipendente da un PC: viene programmata ad-hoc la seconda board per fornire gli input necessari, salvando in memoria i risultati prodotti; in un secondo momento si procederà poi allo download degli output su pc ad una velocità indipendente da quella dal test nel quale vengono prodotti. Il problema che sorge con questa soluzione è il costo dell'aggiunta del nuovo dispositivo ed il tempo di riprogrammazione per ogni test
Con una scheda di acquisizione DAQ il problema del tempo necessario per la riprogrammazione non si pone: infatti i dati di input all'algoritmo vengono scritti dinamicamente dal computer al DAQ, e letti sui canali di output dalla scheda; rimane comunque un problema di costo non indifferente, in quanto non è usuale comprare un sistema di acquisizione per campionare segnali a 3 Mbit.
La scheda PCI-express è un'alternativa a basso costo, difatti sulla maggior

parte dei pc ini produzione dopo il 2007 è presente un adattarore di tipo PCI-express. Il vantaggio di questa scheda è l'elevata velocità di trasmissione, si parla di link full-duplex con banda di Gb/s al confronto dei Mb/s richiesti dalla nostra applicazione. Il problema riscontrato per questa soluzione è però la complessità di programmazione dei pin dell'adattatore: sarebbe infatti necessario usare delle direttive del sistema operativo di non facile utilizzo ed accessibili solo tramite kernel linux per riuscire ad leggere/scrivere livelli di tensione al singolo pin.
La scelta finale è stata utilizzare un modulo adattatore della FTDI (FT2232H) che permetta di instaurare una connessione tra il pc e la board: dal lato del computer il link esporrà un connettore USB, mentre dall'altro avrà intestato un bus di cavi per collegarsi alla board.

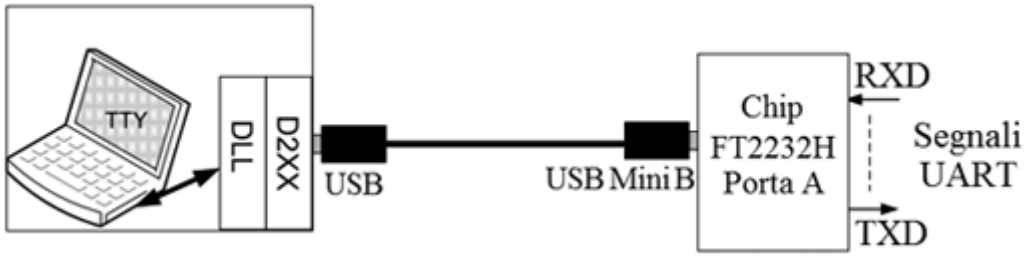

Figura 3.4: *Connessione tra computer e scheda FPGA*

La velocità supportata dal collegamento è conforme agli standard USB e con una scelta accurata del protocollo di comunicazione il limite teorico è di 400 Mbit/s: garantisce quindi un ampio margine dalla richiesta delle specifiche di progetto. Lo standard USB 2.0 fornisce un link di comunicazione half duplex, quindi può gestire sia l'invio degli input alla scheda che la ricezione degli output.
Per quanto riguarda il prezzo, un chip modello FT2232H contornato di spese di spedizione costa 25/30 euro e rientra nei budget previsto per un progetto sperimentale.

Modulo FTDI FT2232H

La FTDI *Future Technology Device International* è un'azienda specializzata nel convertire periferiche standard per comunicare con il protocollo USB, offrendo soluzioni pronte all'uso per l'interfacciamento con il pc.
In particolare il chip FT2232H rappresenta la 5a generazione dei loro dispositivi e permette la conversione da protocollo USB 2.0 high speed (480 Mb/s) a segnali UART/FIFO IC, riconfigurabile in diversi protocolli ed interfacce seriali. Una key feature di questo adattatore è la capacità di gestire l'intero protocollo USB on chip, semplificando il lato di comunicazione della FPGA, che nella maggior parte delle configurazioni non possiede un controller USB preprogrammato. Per l'invio dati si può settare uno fra i diversi protocolli di scambio dati disponibili sul chip, in particolare è possibile impostare quello ottimale per la specifica applicazione.

Il chip scelto per questo progetto ha due porte di trasmissione [2] (A e B) ed un adattatore USB, ciascuna delle porte può essere usata indipendentemente per stabilire un channel di comunicazione half duplex.

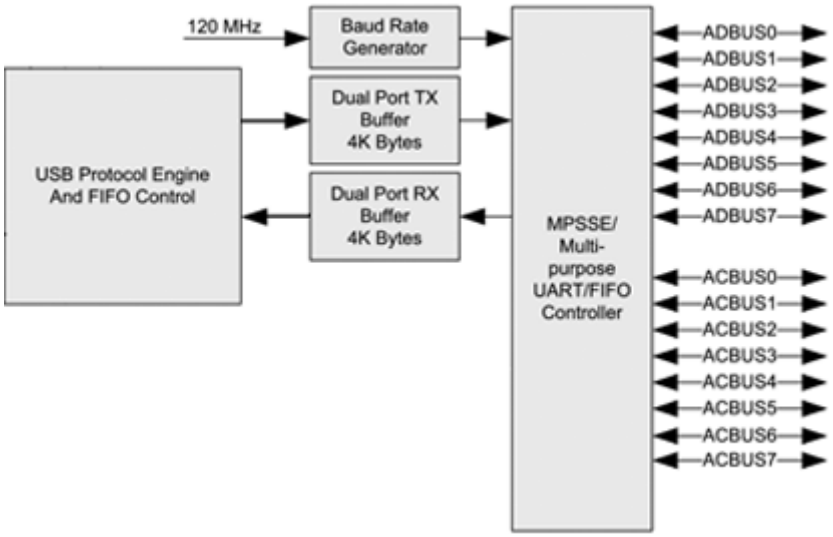

Figura 3.5: *Diagramma a blocchi porta A*

Il diagramma mostra la porta A del modulo, la quale espone all'esterno 16 bit divisi in 2 canali: uno per i dati e uno per il controllo della comunicazione.
Il canale di trasmissione dati è a 8 bit, ciò significa in termini di tempistica che ad ogni colpo di clock il componente trasmette un byte alla FPGA: da lato hardware il componente VHDL per la ricezione dei pacchetti dovrà campionare 1 byte a fronte dell'edge del clock.

Il canale per il controllo presenta anch'esso 8 bit, il cui significato dipende dal protocollo scelto per la comunicazione.

Protocolli di comunicazione del modulo

Il chip scelto per instaurare il collegamento offre svariate configurazioni di trasmissione, che permettono di scegliere il protocollo di comunicazione più adatto alla nostra applicazione.
Viene presentata una breve descrizione [1] delle più utilizzate:

- FIFO CPU style: è un protocollo con modalità sincrona di trasmissione, viene utilizzato per fornire il collegamento USB a CPU e microprocessori. E' semplice da implementare ed utilizza solo 3 segnali di controllo;
- ASYNC Serial (RS232) : questo protocollo segue lo standard seriale (RS232) con i pin TXD, RXD per la comunicazione dati e RTS, CTS per il controllo;
- SYNC Bit-Bang: il protocollo bit-bang istaura un canale trasmissivo di semplice concetto ma allo stesso tempo difficile da strutturare: si ha accesso diretto al valore delle linee dato e si definisce la trama dei bit senza alcun vincolo;
- FT245 SYNC FIFO: protocollo proprietario FTDI ad alta velocità con 8 segnali dato e 8 di controllo.

I protocolli sopra elencati offrono tutti caratteristiche ottimali per il trasferimento dati punto-punto. Restringendo il campo di ricerca ai soli con trasmissione sincrona (per una più semplice implementazione lato hardware) è stato scelto il protocollo con maggiore stabilità e banda garantita: FT245 sincrono. Questo livello fisico garantisce trasferimenti con frequenza di clock a 60MHz e data rate fino a 40MBytes/s (8 Mbytes/s effettivamente raggiunti) , sfruttando il link USB in modalità half duplex.
Da questa prima analisi, i requisiti della nostra applicazione di banda 3Mbit/s con invio bidirezionale sono soddisfatti.

Configurazione iniziale dispositivo

Per impostare la modalità di trasferimento FT245 è necessario configurare il chip sia con la programmazione della EEPROM interna (tramite il programma fornito dall'azienda), sia prima di aprire la porta di comunicazione utilizzando le specifiche direttive del driver.

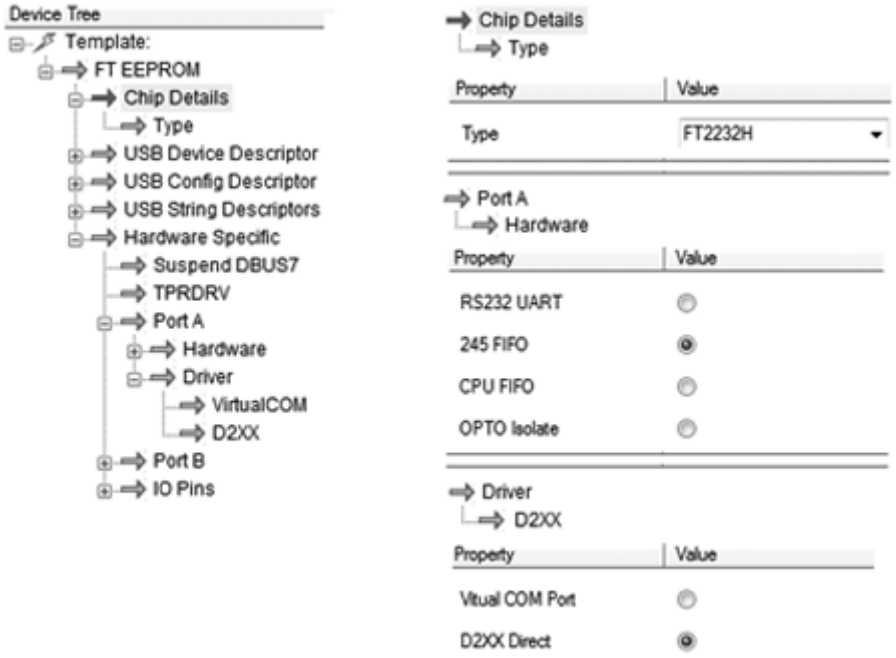

Figura 3.6: *Schema di configurazione*

La figura mostra sulla sinistra l'albero delle proprietà della scheda: per il protocollo utilizzato serve modificare le sezioni Chip Details->Type, Port A->Hardware e Port A->Driver.
Le opzioni corrette di configurazione sono riportate sulla destra, le prime due indicano rispettivamente il tipo di dispositivo e il protocollo. La terza serve a stabilire come apparirà il chip all'interno di windows: nel caso VCOM port sarà creata una porta virtuale associata al dispositivo da cui si potrà accedere in lettura o scrittura con un semplice terminale; nel caso D2XX drivers si utilizzeranno i drivers proprietari per accedere alla periferica.
Per la particolare configurazione di protocollo è stato ritenuta migliore la scelta dei driver proprietari, di più facile gestione in ambiente .NET.
Il secondo step di configurazione è rimandato alla sezione specifica del capitolo 3, contenente la descrizione delle routine dei driver.

Protocollo proprietario FT245 Sync

FT245 Sync è un protocollo di comunicazione proprietario della FTDI per instaurare una comunicazione USB punto-punto. Esso permette di gestire uno stream di dati utilizzando 16 pin di logica, suddivisi in 8 per segnali dati e 8 per segnali controllo. La particolarità del nome sta nel fatto che quando il dispositivo è un idle, i pin si trovano a 2.45V, esattamente a metà fra il livello logico 1 e 0. Per realizzare un collegamento con questo protocollo vengono utilizzati i pin della sola porta A del modulo FT2232H, organizzati secondo la tabella riportata.

Pin N°	Segnale	Tipo	Descrizione
16-24	ADBUS[7:0]	I/O	Bus a 8 bit dei dati, organizzati nel buffer interno secondo una politica FIFO. E' un canale input (da FPGA a PC) per default, output (da PC a FPGA) se OE è basso
26	RXF#	OUTPUT	Se alto non c'è alcun dato presente in FIFO, se basso sono presenti dati in coda disponibili per la lettura che posso essere prelevati abbassando RD#. In modalità sincrona i dati vengono prelevati ad ogni colpo di clock con RXF# e RD# bassi. OSS: per leggere il bus è necessario abilitare il segnale OE basso un clock prima del comando di lettura.
27	TXE#	OUTPUT	Se alto è disabilitata la scrittura dati nella FIFO, se basso è possibile trasmettere byte pilotando il comando WR# basso When high, do not write data into the FIFO. In modalità sincrona i dati vengono campionati ad ogni colpo di clock con TXE# e WR# bassi. Il bus è configurato come input con il valore di default di OE#
28	RD#	INPUT	A livello basso viene prelevato il primo byte disponibile nella FIFO e pilotato sui pin di dato ADBUS. Ad ogni colpo di clock viene esposto il dato successivo finchè il comando RD non sale a livello logico alto
29	WR#	INPUT	Attivo a livello basso, campiona i dati presenti in ADBUS e li trasmette nel buffer FIFO. Ogni colpo di clock viene trasmesso un nuovo byte, finchè WR torna alto.

Pin N°	Segnale	Tipo	Descrizione
32	CLKOUT	OUTPUT	Clock a 60Mhz prodotto dal chip FTDI. La tempistica dei segnali e dei dati di trasmissione deve essere sincronizzata con questo clock.
33	OE#	INPUT	Gestisce la configurazione dei pin dati (in o out) per la trasmissione. A livello basso combinato con il comando RD sono in lettura, a livello alto con il comando WR sono in scrittura.
30	SIWU	INPUT	Il segnale *Send Immediate / WakeUp* combina due funzioni su un singolo pin. Se il modulo è in stato di sospensione, pilotando SIWU basso è possibile mandare sul bus USB un segnale di resume (Wake up del pc). Se è in stato di normale operatività, pilotando SIWU basso tutti i dati nel buffer TX verranno immediatamente inviati sul bus USB (Send Immediate). Per quanto riguarda la funzionalità del nostro progetto terremo SIWU disabilitato a 1

Dopo avere introdotto i segnali utilizzati per gestire il protocollo si procede con l'analisi delle tempistiche di lettura e scrittura, al fine di poter scrivere correttamente il modulo VHDL per la trasmissione.
In particolare vengono presentati i cicli di trasmissione dato con i vincoli temporali di setup e hold che ne conseguono.

Lettura di un byte dal buffer USB

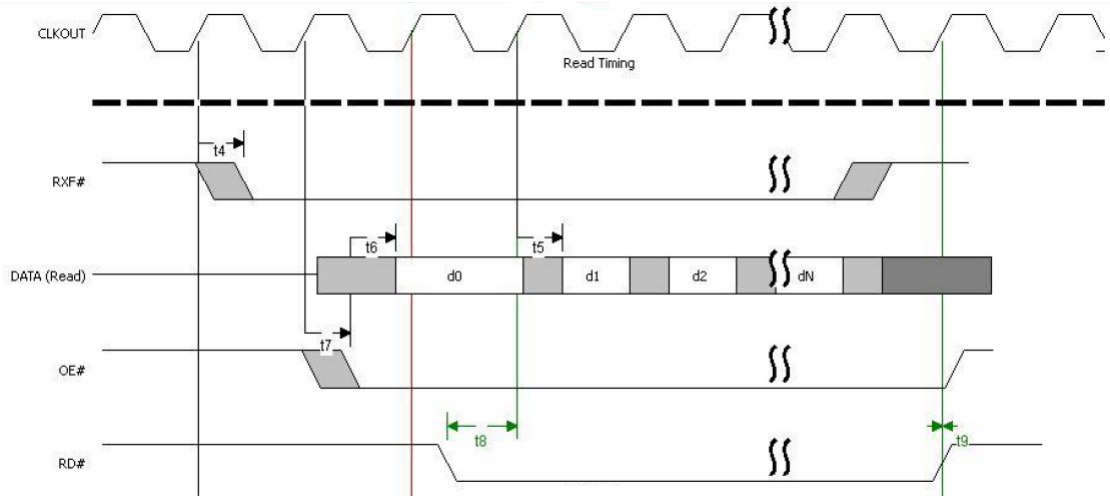

Figura 3.7: *FT245 Sync Read*

La figura 3.7 contiene il diagramma temporale dei segnali per effettuare la lettura di un byte dal chip adattatore, secondo il protocollo FT245. Analizzando lo schema si deducono i seguenti questi passaggi:

1. Il chip segnala la presenza di dati nel buffer con il segnale RXF#.
   Esso viene aggiornato al fronte di salita del clock e portato a livello logico basso per indicare un nuovo burst pronto per la lettura.
   Il tempo t4 avverte che la transizione è soggetta ad un ritardo minimo di 7.15 ns.

2. Il dispositivo per la ricezione prepara i pin del dispositivo per la lettura con il segnale OE#.
   Portato a livello logico basso configura i pin del dispositivo in alta impedenza. Il primo byte viene presentato stabile dopo t6 (7.15ns)

3. Sempre il blocco di ricezione comanda RD#, mandando a livello logico basso almeno un colpo di clock dopo OE. Il byte d0 rimane stabile per la lettura fino al successivo fronte di salita del clock. t8 rappresenta il setup time per la RD (11ns)

4. Ad ogni colpo di clock un nuovo byte (d1,d2,dn) viene letto e presentato sui pin dati. t5 (7.15ns) rappresenta il tempo minimo di attesa per il fetch da FIFO del nuovo dato

Scrittura di un byte nel buffer

La figura sottostante mostra l'andamento temporale dei segnali per la scrittura con il protocollo FT245.

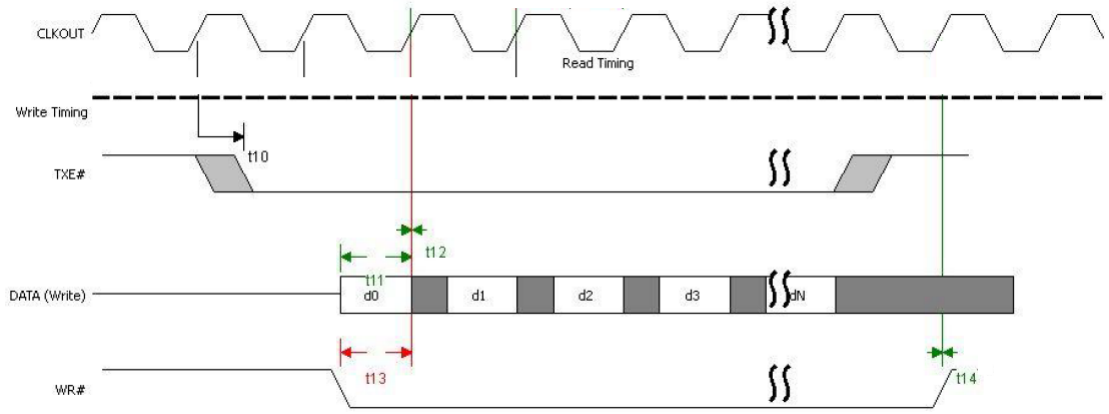

Figura 3.8: *FT245 Sync Write*

1. Il chip FTDI pilota TXE a livello logico basso, segnalando il bus libero e i pin del chip in configurazione alta impedenza (adatti a leggere un segnale in tensione). Il tempo t10 indica il ritardo massimo, pari a 7.15 ns, della transizione di segnale a fronte del clock.

2. Il master di comunicazione presenta il primo dato sul bus (d_0), con t_{11} minimo setup time (11ns)

3. Il master comanda al chip la trasmissione del dato nel bus in FIFO configurando il segnale di scrittura WR basso. t_{12} rappresenta l'hold time (0ns) sincronizzato con CLKOUT

4. Ad ogni successivo colpo di clock con WR basso verrà trasmesso il dato presente sul bus (d_1, d_2, d_n)

3.3.2 Protocollo di comunicazione applicativo

Il protocollo scelto per l'interfacciamento pc-fpga e realizzato sopra il link FT245 segue lo schema di una comunicazione a pacchetti. In questo contesto il master della comunicazione è il pc, unico interlocutore che può inviare pacchetti contenenti keyword di comando: la scheda non può trasmettere niente senza che il pc lo richieda.
La scelta dei pacchetti necessari per compiere una simulazione HIL non è banale: da un lato deve permettere al pc di inviare i dati di configurazione, di stimolo e di comando; dall'altro lato la scheda fpga deve essere in grado consegnare i dati di risultato della simulazione. Data la natura duplice del payload da inviare (comandi e dati) si è deciso di suddividere l'insieme dei pacchetti in due tipologie:

- Pacchetti di tipo Service (SP): di struttura a lunghezza fissa, contenente informazioni di servizio e keyword.
- Pacchetti di tipo Data (DP): di struttura con payload variabile, per l'invio grandi burst di dati.

Pacchetti di tipo service

I pacchetti SP vengono utilizzati per lo scambio di keyword e informazioni ausiliarie a basso payload, e hanno una dimensione fissa pari a 7 byte.

Byte N°	**Campo**	**Valore**	**Posizione**
1	Trama di start 1	7F	bit(7:0)
2	Trama di start 2	7E	bit(7:0)
3	Formato dati Keyword comando	11 -	bit(7:6) bit(3:0)
4	Aux data	-	bit(7:0)
5	Aux data	-	bit(7:0)
6	Trama di stop 1	FE	bit(7:0)
7	Trama di stop 2	FD	bit(7:0)

Dopo due byte contenenti la sequenza di start pattern, al terzo byte inizia l'header di pacchetto: i primi due bit contengono il formato 11, che identifica il tipo di pacchetto; gli ultimi 4 bit sono la parola chiave di comando di quel pacchetto. Concluso l'header il payload è composto da due byte ausiliari per informazioni di poco ingombro, ad esempio il livello delle soglie delle FIFO.
Infine il delimitatore è rappresentato da due byte di stop pattern a valore fisso.

Elenco dei pacchetti di tipo service

Tipo pacchetto	Parola chiave	Descrizione
SP Send THR	0010	Pacchetto per richiedere alla scheda il livello delle soglie della FIFO. Viene utilizzato per inferire se la memoria interna sta raggiungendo i limiti di spazio disponibile ed è quindi necessario mettere in pausa la simulazione e prelevare i dati di risultato
SP Unload	0100	Pacchetto per comandare alla scheda di inviare i dati di risultato contenuti nella FIFO di uscita (FIFOe)
SP Ack	0111	Pacchetto di ack per segnalare la fine delle procedure di svuotamento della FIFOe
SP StartCfg	0101	Pacchetto di inizio configurazione, precede i dati di configurazione
SP DbgReal	1010	Pacchetto di fine configurazione, indica l'inizio della simulazione con dati provenienti da DAC e ADC
SP DbgPath	1011	Pacchetto di fine configurazione, indica l'inizio della simulazione con dati provenienti da computer
SP FReset	1100	Questo pacchetto contiene il comando di reset delle FIFO dati. Viene utilizzato per pulire i dati residui nelle memorie a seguito di un comportamento anomalo della simulazione.
SP TestCh	1011	Contiene la parola chiave per entrare nella modalità di test del canale: viene sganciato il reparto di algoritmo per testare in anello chiuso il percorso dei dati.

Pacchetti di tipo Data

I data packet vengono utilizzati per inviare quantità di dati ad elevato payload, e presentano una dimensione variabile in base al loro contenuto.
La loro struttura è riportata in figura.

Byte N°	**Campo**	**Valore**	**Posizione**
1	Start Pattern 1	7F	bit(7:0)
2	Start Pattern 2	7E	bit(7:0)
3	Formato dati Keyword registro	10-01-00 -	bit(7:6) bit(3:0)
4	Payload Lenght 1	-	bit(7:0)
5	Payload Lenght 2	-	bit(7:0)
6..N	Payload data	-	bit(7:0)
N+1	Stop Pattern 1	FE	bit(7:0)
N+2	Stop Pattern 2	FD	bit(7:0)

Il pacchetto è strutturato in pattern di start, header, payload e pattern di stop. Il primo byte dell'header è suddiviso in formato dati e keyword: il formato descrive la risoluzione dei dati, ovvero se sono organizzati in gruppi da 3 (codice 10 - 24 bit), da 2 (codice 01 - 16 bit) o da 1 byte (codice 00 - 8 bit); la keyword indica l'indicativo del registro dove i dati vengono memorizzati per la simulazione.
I successivi due byte contengono la lunghezza del payload espressa in byte: questa informazione verrà utilizzata dal parser per delimitare la fine del pacchetto.
Il payload e il pattern di stop contengono rispettivamente i dati core del pacchetto e la trama conclusiva di controllo. I tipi di pacchetto si differenziano in base al significato dei dati che contengono, e quindi alla keyword del registro dove verranno scritti: per questo motivo la keyword è anche usata come identificativo.

Elenco dei pacchetti di tipo data

Tipo Pacchetto	Keyword	Payload	Descrizione
DP Signal x	0001	1024 byte	Pacchetto con il segnale di stimolo, da porre x in ingresso al percorso primario.
DP DAC	0011	2 byte	Pacchetto con la configurazione del componente DAC
DP Mu	0101	8 byte	Pacchetto con i valori dei 4 guadagni dell'algoritmo, due utilizzati per la stima offline e due per la stima online
DP ADC	1000	2 byte	Pacchetto con la configurazione del componente ADC
DP BUS	0110	3 byte	Pacchetto con la descrizione della risoluzione e dei taps dei filtri
DP TAPS P1..S22	1010 to 1111	7 taps	Pacchetto con i valori dei pesi dei filtri. P1,P2 contengono la descrizione della funzione di trasferimento del percorso primario da stimare, mentre S11..S22 la funzione di trasferimento del percorso secondario.

3.3.3 Ristrutturazione VHDL per HIL

La top-entity del progetto originale contenente l'algoritmo FxLMS viene riconfigurata per la simulazione HIL aggiungendo tre reparti:

- trasmissione
- memorizzazione dati
- test

La parte di trasmissione gestisce la comunicazione con il pc per la condivisione di input e risultati. La sezione di memorizzazione dei dati consente di scambiare informazioni fra il reparto trasmissivo e algoritmico. La sezione di test è utilizzata per sganciare la parte di logica e testare il canale trasmissivo in anello chiuso.

Reparto di trasmissione

Il reparto di trasmissione è contenuto nella top-entity *TOP CONN* ed è formato da due componenti:

- TX CTR: blocco di trasmissione per inviare pacchetti dalla FPGA al computer;
- RX USB: blocco di ricezione per pacchetti dati e configurazioni da computer sulla FPGA.

I due componenti lavorano al medesimo clock (60MHz), chiamato clock di trasmissione e prodotto dall'oscillatore interno al chip FTDI.
Avendo un unico bus per i dati , esso è condiviso in lettura/scrittura da questi due componenti che lo occupano secondo una politica di prenotazione con i segnali usb_tx_busy e usb_oen.
A livello di codice VHDL il bus viene dichiarato come segnale INOUT, ed assegnato ad alta impedenza(IN) o bassa impedenza(OUT) in base al segnale di chi sta occupando attualmente il bus.

```
... Dichiarazione
usb_dbus  : inout std_logic_vector(7 downto 0);
... Architecture
usb_dbus <=  USB_DBUS_OUT when (USB_TX_BUSY ='1')
                                          else (others => 'Z') ;
USB_DBUS_IN <= usb_dbus;
```

Componente di trasmissione: IO

Il componente per la trasmissione, contenuto nel blocco *TX CTR* , costruisce e regola il flusso dei pacchetti che vengono inviati da FPGA verso il pc tramite il chip della FTDI. Per interfacciarsi con i pin del chip è stata scritta una macchina a stati che implementa il protocollo FT245, che instaura un canale di comunicazione a 8 bit di dato e 8 di controllo.
Prima di descrivere i processi costitutivi della struttura viene presentata una tabella riassuntiva di IO del blocco.

Segnale	**Tipo**	**Descrizione**
res	IN 1 bit	Reset asincrono del componente di trasmissione. Reinizializza la macchina a stati che costituisce la trasmissione nonchè tutti i segnali interni di gestione
clk	IN 1 bit	Clock per il reparto trasmissione a 60Mhz: viene generato dal chip FTDI e ricevuto qui come input sincronizzatore
init	IN 1 bit	Segnale di inizializzazione della trasmissione, posto a 1 di default
txe	IN 1 bit	E' il segnale connesso al pin TXE del chip FTDI, indica la possibilità di trasmettere in FIFO se basso
usb oen	IN 1 bit	E' il segnale di OE che gestisce la configurazione dei pin del chip FTDI. Viene usato come bit di prenotazione, se è a livello basso non è possibile trasmettere
afull x, full x, aempty x, empty x	IN 1 bit	Sono i segnali provenienti dalla FIFO con i dati x. Il loro significato è quello di segnalare le soglie della memoria varcate se si trovano a livello logico alto.
afull e, full e, aempty e, empty e	IN 1 bit	Sono i segnali provenienti dalla FIFO con i dati e. Il loro significato è identico ai pin precedenti.
word bus	IN 32 bit	E' il bus con gli errori calcolati dall'algoritmo proveniente dall'uscita della memoria FIFOe
cmd bus	IN 4 bit	E' il bus con i comandi ricevuti dal PC. Si ricorda che il computer è master nella comunicazione, la FPGA elabora i risultati e li trasmette quando il comando lo permette

Segnale	Tipo	Descrizione
usb wre	OUT 1 bit	Segnale che controlla la scrittura sulla FIFO del chip di trasmissione, attivo basso. Viene collegato al pin WRE del chip
end txc	OUT 1 bit	Segnala la fine della trasmissione di un pacchetto, attivo alto
usb tx busy	OUT 1 bit	Segnale portato alto quando il bus viene prenotato per la scrittura, bus occupato
tx err	OUT 1 bit	Segnale portato alto se il componente produce un errore nella trasmissione di un pacchetti
fifo err en	OUT 1 bit	Segnale di enable per la lettura dei dati dalla fifo errori. Durante la procedura di unload degli errori viene attivato per inviare i pacchetti prelevando i risultati.
usb_dbus	OUT 8 bit	Bus di trasmissione dati a 8 bit

Componente di trasmissione: struttura VHDL

Il blocco per la gestione del protocollo di invio pacchetti si basa sulla definizione di 4 processi, sincronizzati con il clock di trasmissione.

- Processo di timeout
- Contatore dati
- Decoding comandi
- Compositore/sender dei pacchetti

Il processo di timeout gestisce il segnale tx_err attraverso un contatore countdown: se pari a zero è stato commesso un errore di trasmissione e viene alzato il bit corrispondente.
Questo tipo di controllo viene utilizzato nel momento in cui il reparto di trasmissione aspetta un segnale comando ack di conferma, nel caso il segnale non giunga entro un tempo stabilito si intuisce un errore di trasmissione.

Il processo contatore dati stabilisce se è stata completata la trasmissione di un pacchetto, segnalando la fine del blocco dati alla macchina a stati responsabile. Prima della trasmissione del burst viene configurata la lunghezza del payload in bytes, selezionato in base al tipo di pacchetto da inviare. Il conteggio viene poi aggiornato ad ogni colpo di clock se il controllo di scrittura byte è attivo (WTDT ='0'), .

Il processo decoding comandi interpreta il bus dei comandi a 4 bit, aggiornato dal componente di ricezione quando il computer invia un pacchetto di servizio.
Una volta decodificato, nel caso debba inviare un pacchetto di risposta, calcola i segnali HD_B3, PAYLOAD_LENGTH che contengono rispettivamente l'header del pacchetto da inviare e il suo PAYLOAD.
La tabella seguente mostra i valori delle risposte in base al comando ricevuto.

Comando	**Condizioni**	**Pacchetto**	**Hd**	**Payload**
Livello soglie	-	SP livello soglie	x85	1 byte
Unload FIFO	!almost empty	DP data FIFO_e	x47	1024 byte
Unload FIFO	almost empty	SP fine trasmissione	x48	1 byte

L'ultimo processo è costituito da una macchina a stati che implementa la creazione di un pacchetto e la sua conseguente trasmissione.
Lo stato iniziale legge il comando presente su cmd_bus e stabilisce se è necessario inviare un pacchetto, controllando che il bus non sia già occupato:

```
if ((cmd_bus="0100" or cmd_bus="0010") and usb_oen='1')
  usb_dbus    <= START_PATT1;     -- Trama di start
  IUSBWR      <= '0';             -- Comando di scrittura
  usb_tx_busy <= '1';             -- Bus prenotato
  CURR_STATE  <= HD1;             -- Stato successivo
end if;
```

Negli stati successivi viene inizialmente completato lo start pattern, in seguito scritto l'header e il payload.

```
when HD1 =>
  usb_dbus <= START_PATT2;
  CURR_STATE <= HD2;
when HD2 =>
  usb_dbus <= HD_B3;
  CURR_STATE <= HD3;
when HD3 =>
  usb_dbus <= PAYLOAD_LENGTH(7 downto 0);
  CURR_STATE <= HD4;
```

```
when HD4 =>
  usb_dbus <= PAYLOAD_LENGTH(15 downto 8);
  IUSBWR <= '0';
  if HD_B3=X"47" then
          CURR_STATE <= MDT1;
          fifoErr_en<='1';       -- fonte dati FIFOe
  elsif HD_B3=X"48" then
          CURR_STATE <= TDT1;    -- fonte soglie
  end if;
```

Con un pacchetto di livello soglia il processo preleva i valori e li scrive in un byte.

```
when TDT1 =>
  usb_dbus <= aempty_x & empty_x & afull_x &full_x &
              aempty_e & empty_e & afull_e &full_e ;
```

Nel caso di un pacchetto dati viene abilitato il segnale di lettura dalla FIFOe, e viene scompattato il dato memorizzato a 32 bit nei due errori a 16 bit. In ciascuno degli stati (MDTD) ho IUSBWR e WDTD a 0 in quanto sto trasmettendo e il dato partecipa a fare aumentare il conteggio dei byte.

```
when MDT2 =>
  fifoErr_en <='0';
  usb_dbus <= word_bus(31 downto 24);
  CURR_STATE <= MDT3;
when MDT3 =>
  fifoErr_en<='0';
  usb_dbus <= word_bus(23 downto 16);
  CURR_STATE <= MDT5;
when MDT5 =>
  fifoErr_en<='1';
  usb_dbus <= word_bus(15 downto 8);
  CURR_STATE <= MDT4;
when MDT4 =>
  usb_dbus <= word_bus(7 downto 0);
  fifoErr_en<='0';
  if DTCOUNT_TC ='0' then
    CURR_STATE <= MDT2;
  else
    CURR_STATE <= STP1;
  end if;
```

Componente di ricezione: IO

Il componente per la ricezione dati, contenuto nel blocco RX_USB, riceve il flusso di byte proveniente da PC, interfacciandosi con lo standard proprietario FT245.
Questo blocco svolge le seguenti funzionalità:

- Parser di pacchetto: interpreta il flusso di byte nel buffer in pacchetti, secondo la struttura definita dal protocollo di comunicazione. Viene utilizzato lo schema DP se contengono dati, SP se contengono informazioni di servizio o comandi.
- Interpretazione e consegna comandi di simulazione: con pacchetti SP il comando in formato 8 bit viene decodificato e vengono attivati i segnali di output necessari al completamento dell'istruzione.
- Scrittura dei dati e delle configurazioni nel reparto FIFO: con pacchetti DP i dati vengono salvati in memorie ausiliarie FIFO nel reparto addetto alla risoluzione del problema della metastabilità.

Come per il blocco precedente viene presentata una tabella rissuntiva con i segnali di IO necessari per la ricezione, suddivisi in figura come input a sinistra e output a destra:

Segnale	**Tipo**	**Descrizione**
res	IN 1 bit	Reset asincrono del componente RX. Reset della macchina a stati per la ricezione dati e dei segnali di gestione ad essa associati
clk	IN 1 bit	Clock per il reparto ricezione a 60Mhz: viene generato dal chip FTDI e ricevuto qui come input sincronizzatore
rxf	IN 1 bit	E' il segnale connesso al pin RXF del chip FTDI, se basso indica la presenza di almeno un byte da leggere nel buffer del chip
end_txc	IN 1 bit	E' il segnale connesso all'omonimo pin del blocco di trasmissione. Indica quando è stata inviato un pacchetto completo
usb_tx_busy	IN 1 bit	Attivo alto, indica quando il bus dati condiviso è occupato dal componente di trasmissione. Viene connesso all'omonimo del blocco TX_CTR
ch2dot1	IN 1 bit	Attivo alto, indica quando il chip è attivo. Usato come segnale di presenza.
usb_dbus	IN 8 bit	Bus dati condiviso, connesso al chip FTDI.

Segnale	Tipo	Descrizione
usb_rd	OUT 1 bit	Segnale di lettura connesso al pin RD del chip FTDI, attivo basso.
usb_oen	OUT 1 bit	Segnale di configurazione bus (I/O) del modulo FTDI. Connesso al pin OEN
rx_err	OUT 1 bit	Se alto segnala un errore generico nel pacchetto ricevuto: l'errore standard è in presenza di una perdita nel bytestream dati, che innesca l'anomalia nel conteggio finale.
cmd_ack	OUT 1 bit	Se alto segnala l'acknowledge del comando ricevuto.
deb_nreal	OUT 1 bit	Indica che la scheda sta funzionando in modalità debug, attivo alto. Se basso la FPGA preleva i dati ADC/DAC
conf_cmd	OUT 1 bit	Quando è a livello logico 1 la FPGA è in stadio di configurazione, i dati inviati dal pc contengono informazioni di setup per la configurazione.
P1taps_en, P2taps_en, S11taps_en, S12taps_en, S21taps_en, S22taps_en	OUT 1 bit	Segnali di enable delle FIFO per memorizzare i dati dei taps , trasmessi in fase di configurazione dal PC.
fifo_en	IN 1 bit	Segnale di abilitazione per la scrittura del segnale di stimolo x nella FIFO corrispondente. Viene attivato durante la ricezione di un DP x
adc_we, dac_we, mu_we, buscfg_we	IN 1 bit	Segnali per la scrittura delle configurazioni di adc, dac, guadagni di sistema e bus.
data_bus	OUT 16 bit	Bus dati condiviso da tutta l'architettura. Può contenere segnali, configurazioni, taps
cmd_bus	OUT 4 bit	Contiene il nibble con l'istruzione di comando ricevuta dal computer

Componente di ricezione: struttura VHDL

Il blocco per la ricezione pacchetti si basa sulla definizione di 5 processi principali, sincronizzati con il clock di trasmissione.

- Processo di read timeout
- Contatore dati ricevuti
- Ricezione, decodifica, propagazione comandi
- Parser dei pacchetti
- Indirizzamento e memorizzazione dei dati

Il processo di read timeout è utilizzato per indicare un errore nella ricezione alzando il segnale rx_err. Durante la ricezione dei byte che compongono un pacchetto, se il buffer è vuoto il conteggio viene abilitato: nel caso il master non scriva i byte rimanenti nel buffer prima del trigger di timeout si considera un errore di trasmissione e si alza il segnale rx_err. Il processo contatore ha il compito di determinare la fine del payload, e quindi del burst, quando viene ricevuto un pacchetto dati.
Alla ricezione dell'header di un data packet il processo campiona la lunghezza del payload in byte, contenuto in PAYLOAD_LENGTH. Quando il parser preleva dati di payload il contatore viene abilitato: l'aggiornamento procede per ogni colpo di clock con segnale di lettura abilitato; completato il conteggio il processo comunica la fine del pacchetto al parser, il quale si mette in attesa del pattern di delimitazione.

```
...
if (DTCOUNT_TC ='0' and DTCOUNT_EN ='1') then
  DTCOUNT <= DTCOUNT  + not(rxf);          -- update counter
  if DTCOUNT = (PAYLOAD_LENGTH - 2)then    -- end condition
    DTCOUNT_TC  <= '1';                    -- trigger signal
  end if;
end if;
```

Il processo di gestione comandi è attivato al momento della ricezione di un pacchetto di servizio (CMD_REG_CE='1'). Il suo compito è di estrarre la keyword della dimensione di un byte dall'header e di propagarla all'interno dell'architettura sui segnali cmd_bus.

```
case CMD_STATE is
when S0 =>
  cmd_bus <= (others=>'1');  -- no keyword
  if CMD_REG_CE ='1' then
   ACK_STATE <= S1;
  end if;
when S1 =>
  cmd_bus <= DP_ADD;                -- propagate
  if CMD_REG_CE ='0' then         -- end cmd word
   CMD_STATE <= S0;
  end if;
when others =>
CMD_STATE <= S0;
end case;
```

In particolare se il pacchetto fornisce il setup per la modalità di funzionamento (DEBUG/ONLINE), vengono attivati due segnali aggiuntivi: deb_nreal e conf_cmd.

```
case DP_ADD is
 when "0101" =>        -- Enter configuration mode
  conf_cmd <= '1';
 when "1010" =>        -- Exit conf, enter ONLINE mode
  deb_nreal <='0';
  conf_cmd <= '0';
 when "1011" =>        -- Exit conf, enter DEBUG mode
  deb_nreal <='1';
  conf_cmd <= '0';
end case;
```

Il processo parser interpreta il bytestream da pc secondo la struttura a pacchetti prevista.
In modalità DEBUG, se il bus non è occupato per la trasmissione il componente parser occupa il link, predispondendo i pin per la lettura:

```
if rxf ='0' then                -- Data should be read
  case CURR_STATE is
  when START =>                  -- Idle state
    DTCOUNT_EN  <= '0';
    DTCOUNT_RES <= '1';          -- Reset count of payload
    usb_oen     <= '1';          -- Still don't busy the bus
    IUSBRD      <= '1';          -- Don't read
    if (usb_tx_busy ='0') then   -- Is bus free?
      CURR_STATE <= TCHECK;
    end if;
  when TCHECK =>
    if (usb_tx_busy ='0') then   -- If bus is free take it
      usb_oen    <= '0';         -- Config pins
      DTRX_ERR   <= '0';
      CURR_STATE <= WBUS1;
    else                         -- Else let the TX write
      CURR_STATE <= START;       -- Return to idle
      usb_oen    <= '1';
    end if;
```

Lo stream in lettura viene strutturato in pacchetti ben delimitati, riconscendo il pattern di start ed interpretando l'header per la lunghezza del payload e quindi il pattern di stop.

```
when WBUS1 =>                              -- Bus disponibile
  IUSBRD       <= '0';                     -- Comando di lettura
  CURR_STATE   <= HD1;
when HD1 =>
  if usb_dbus=START_PATT1 then             -- Controllo trama
    IUSBRD     <= '0';                     -- Continuo a leggere
    CURR_STATE <= HD2;
  else                                     -- Reset
   CURR_STATE  <= START;
   IUSBRD      <= '1';
   usb_oen     <= '1';
  end if;
when HD2 =>
  if usb_dbus=START_PATT2 then             -- Controllo trama
    CURR_STATE <= HD3;
  else
   DTRX_ERR    <= '1';
   CURR_STATE  <= EX1;
  end if;
```

Seconda informazione nell'header è la tipologia di pacchetto, in base alla quale si diversificherà la sequenza di lettura in archiviazione dati o propagazione comandi. In base al tipo di pacchetto DP_ADD conterrà una locazione di registro su cui scrivere (DP), o una parola di comando(SP).

```
when HD3 =>                                -- Check packet type
  DP_ADD        <= usb_dbus(3 downto 0);
  DT_FRMT       <= usb_dbus(7 downto 6);
  if usb_dbus(7 downto 6)="11" then
    CURR_STATE <= SP1;                     -- Type is Service
  else
    CURR_STATE <= HD4;                     -- Type is data
  end if;
```

Il parsing per la lettura di un pacchetto di tipo service è lineare, data la lunghezza fissa dei campi che lo compongono.
Una volta compionata stabilmente la parola di comando, viene propagata all'interno della struttura abilitando il segnale CMD_REG_CE. Come ultimo passaggio viene delimitato il pacchetto con il pattern di stop.

```
when SP3 =>
  CMD_REG_CE  <='1';              -- Propagate cmd
  if usb_dbus =STOP_PATT1 then    -- Check 1st Stop OK
    CURR_STATE <= SP4;
  else                            -- RX Error
    DTRX_ERR <= '1';
    CURR_STATE <= EX1;
  end if;
when SP4 =>
  IUSBRD <= '1';
  if usb_dbus =STOP_PATT2 then  -- Check 2nd Stop OK
          CURR_STATE <= SP5;
  else                          -- RX Error
    DTRX_ERR <= '1';
    CURR_STATE <= EX1;
  end if;
```

Il parsing per la lettura di un data packet dipende dal formato dei byte da leggere, organizzato in 8-16-24 bit. La lettura si conclude a segnale di fine pacchetto dato dal processo di data count sul payload.

```
when DP1 =>
  data_bus(23 downto 16)<= usb_dbus;        -- 24 bit
  CURR_STATE <= DP2;
when DP2 =>
  if (DT_FRMT ="10" or  DT_FRMT ="01") then
    DTEN <='1';                             -- Enable register
  end if;
  data_bus(15 downto 8) <= usb_dbus;        -- 16-24 bit
  CURR_STATE <= DP3;
when DP3 =>
  data_bus(7 downto 0) <= usb_dbus;         -- 8-16-24 bit
  if (DTCOUNT_TC ='1') then                 -- End data trigger
    CURR_STATE <= STP1;
  else                                      -- Read next
    ...                                     -- With same format
  end if;
```

L'ultimo processo calcola l'indirizzamento dei dati ricevuti, in base alla keyword contenuta nell'header. Per indirizzamento si intende attivare il segnale di read enable del registro che dovrà leggere il bus condiviso quando i dati vengono resi disponibili.

```
if res='1' then
  ...                          -- Reset all read enable signals
  RX_DATA <= '0';
elsif rising_edge(clk) then
  if DTEN='1' then             -- If payload
   RX_DATA <= '1';
  case DP_ADD is               -- Switch keyword for enable
  when x"1" =>
   fifo_en     <=  '1';        -- Enable FIFOx
  when x"3" =>
   dac_we      <=  '1';        -- Enable DAC
  when x"5" =>
   mu_we       <=  '1';        -- Enable Mu
   ...
  when x"F" =>
   S22taps_en  <=  '1';        -- Enable S22 taps
```

3.3.4 Reparto dati

I dati necessari per la simulazione, comprendenti segnali di stimolo e dati di configurazione, vengono memorizzati dopo la ricezione in un'area dedicata. Questa sezione oltre a fornire supporto per la memorizzazione risolve il problema della metastabilità dei dati, dato dal passaggio dal dominio di clock di trasmissione al dominio di clock di logica.

Domini di clock

Il reparto di trasmissione e il reparto di logica lavorano su due clock diversi: il primo è dettato dallo standard USB della comunicazione pari a 60 MHz; il secondo viene scelto in base al blocco di logica più lento, stabilito a 27Mhz.
Quando due componenti lavorano su due clock diversi il sistema complessivo è multiclock e i due reparti lavorano in due diversi domini di clock. Nel momento in cui componenti in due domini di clock differenti necessitano di trasmettersi dei dati occorre affrontare il problema della metastabilità dei dati.

Metastabilità

Si parla di metastabilità di un segnale elettrico quando il circuito che lo produce non termina in uno stato stabile a livello logico '0' o '1' nel tempo richiesto per l'operazione.
Come risultato il circuito a valle reagisce in maniera impredicibile, conducendo inevitabilmente ad un errore di sistema. Gli stati metastabili sono propri dei sistemi digitali asincroni e dei sistemi sequenziali sincroni con più di un dominio di clock, come nel caso in analisi. In uno scenario tipico di comunicazione tra due domini di clock i dati provenienti dal primo dominio vengono campionati in flip flop con clock del secondo dominio: inevitabilmente il secondo clock arriverà a violare i tempi di hold e di setup ed il segnale prodotto sarà metastabile.

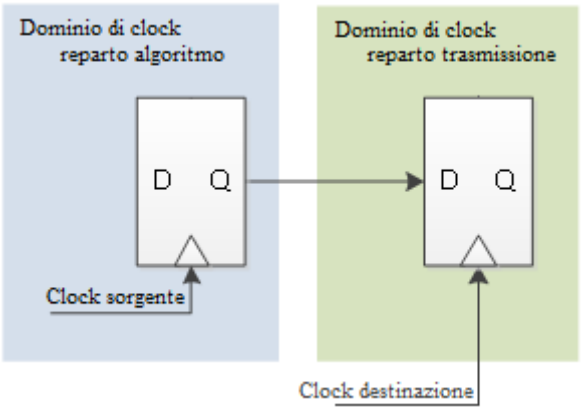

Figura 3.9: *Coppia di Flip Flop metastabili*

Risolvere il problema della metastabilità

Sono state implementate due tecniche per affrontare la metastabilità: una per la trasmissione dei dati e una per le configurazioni.
Poichè i dati di configurazione rimangono stabili per tutta la durata della simulazione non occorre trasmetterli al reparto di logica, ma semplicemente memorizzarli stabilmente in un registro prima del segnale che dà il via alla simulazione. In questo modo la metastabilità viene completamente evitata in quanto il dato di configurazione sarà già stabile nel registro nel momento in cui verrà campionato dal componente di logica.
Per quanto riguarda la trasmissione dei dati la soluzione proposta per risolvere il problema della metastabilità è interfacciare i due domini di clock istanziare una memoria FIFO asincrona per ogni bus di trasmissione. Una fifo dual port è una memoria che gestisce il flusso di dati secondo la politica First in First out, e presenta due clock distinti per la scrittura e per la lettura dei dati al suo interno.
Per comunicare la quantità di memoria occupata espone all'esterno quattro segnali di soglia: empty, almost empty, full, almost full. Empty e full sono standard per ogni FIFO e rappresentano rispettivamente le condizioni di memoria vuota e piena. Almost empty e almost full vengono configurate durante la creazione del componente in base alle esigenze: tipicamente si calcola il tempo necessario al componente più lento per capire che ci si sta avvicinando ad una soglia e si converte in bytes.

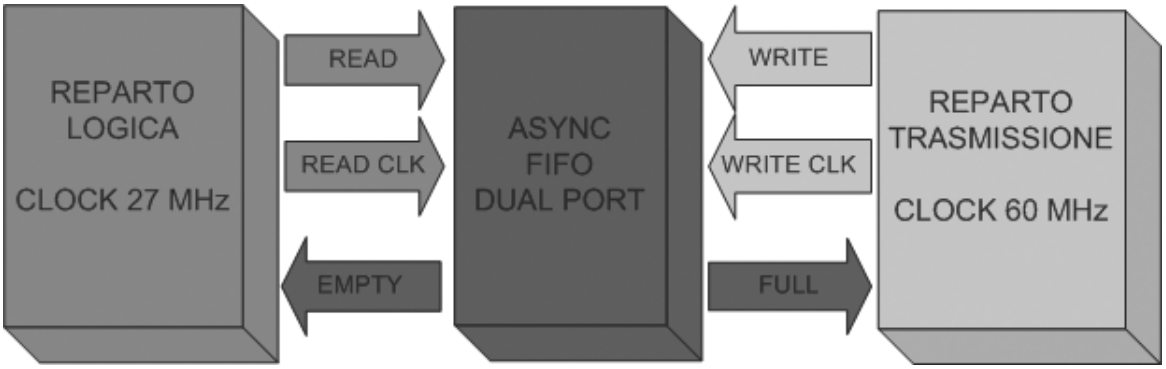

Figura 3.10: *ASYNC FIFO*

In questo progetto l'interfacciamento tramite FIFO avviene come segue:

- Il componente di trasmissione riceve dei dati di stimolo x e li scrive nella memoria condivisa FIFOx attivando il segnale di write enable sincronizzato con il proprio clock a 60 MHz (Write clk)
- Fifox indica la presenza di dati in memoria con il segnale di empty.
- Il blocco che implementa l'algoritmo FxLMS attiva la propria elaborazione in seguito all'interpretazione di empty: la lettura dalla memoria avviene attivando il segnale di read enable sincronizzato con il suo clock di logica.

Blocchi utilizzati

Sono state istanziate due tipologie di componenti, uno per ogni soluzione proposta: un registro a scorrimento per per la gestione delle configurazioni; una memoria FIFO asincrona dual port per la trasmissione dei dati verso il reparto logica e per i risultati provenienti da esso.

Generazione registri

La particolarità del registro a scorrimento è di avere una struttura riutilizzabile per memorizzare ciascun dato di configurazione, ed è stato quindi implementato come componente generico sul numero di byte di risoluzione e il numero di dati totali.

L'unità base che lo costituisce è un registro con reset asincrono e segnale di enable per la memorizzzione di un singolo dato, generico sul numero di byte.

```
entity REG_ONE is
generic (bit_res  : integer);
port    (res      : in  std_logic ;
         clk      : in  std_logic ;
         en       : in  std_logic ;
         word_in  : in  std_logic_vector (bit_res-1 downto 0);
         word_out : out std_logic_vector (bit_res-1 downto 0));
end REG_ONE;
```

L'unità top del registro a scorrimento genererà tanti REG_ONE quanti ne saranno necessari per contenere tutti i bit del parametro di configurazione. Per settare tale numero è stato inserito il generico intero n_bits.

```
entity REG_SC is
generic (bit_res:integer;n_bits:integer);
...
end REG_SC;
architecture RTL of REG_SC is
...
last:= (n_bits/bit_res)-1;
for index in 0 to last  generate
  first: if (index=0) generate
           First_REG : component REG_ONE
                       ...
  n_th: if ((index/=0) and (index/=last) generate
           Nth_REG : component REG_ONE
                       ...
  last: if ((index=last) generate
           last_REG : component REG_ONE
                       ...
```

E' stato istanziato un registro per ogni paramentro di configurazione necessario, ovvero: taps dei filtri, guadagni dell'algoritmo, DAC, ADC.

Generazione FIFO

Per la creazione del codice VHDL delle FIFO è stato utilizzato il tool automatico proprietario della Lattice IPexpress.
Si seguito viene riportata l'interfaccia per implementare una FIFO dual port

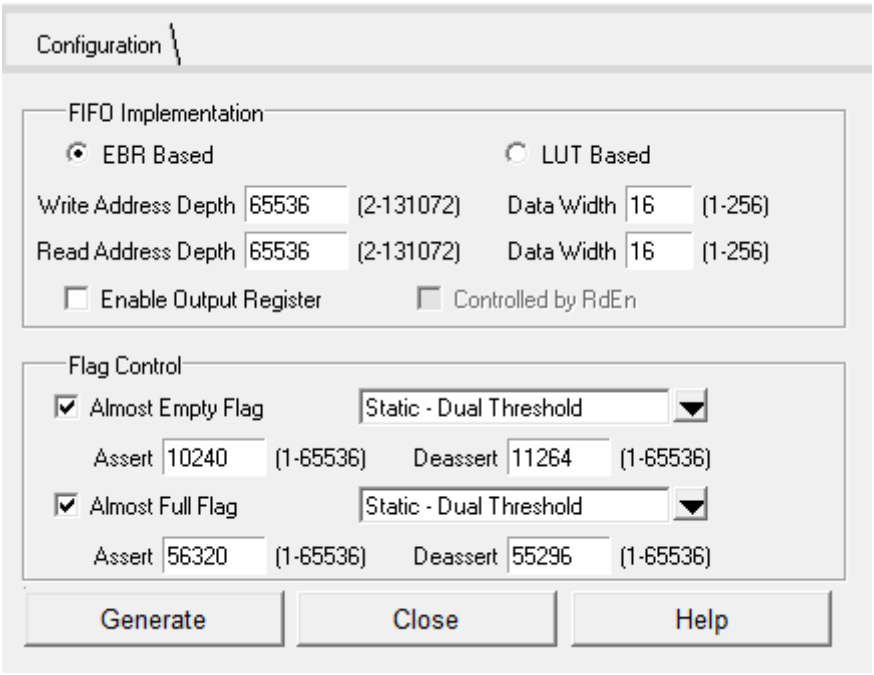

Figura 3.11: *Ipexpress FIFO DC*

Il primo passo è configurare la profontità di scrittura e lettura (indicata come depth): questi due parametri determineranno la grandezza della memoria e di conseguenza il numero di dati nel buffer in attesa di essere processati.
Il parametro data_width è la larghezza di parola della FIFO, nel nostro progetto corrisponde alla risoluzione dei segnali utilizzati (16bit).
Infine i flag permettono di settare il livello statico delle soglie almost empty e almost full, necessarie per il controllo del flusso dei dati ed evitare perdite.
Nel progetto sono state istanziate due FIFO: FIFOx per la bufferizzazione del segnale x in attesa del prelievo da parte del reparto di logica; FIFOe per la memorizzazione dei risultati dell'algoritmo in attesa dell'invio al PC. Data la presenza di due path per un singolo segnale x, verranno prodotti due risultati di errori per ogni dato e di conseguenza la memoria FIFOe è stata implementata con larghezza di parola doppia rispetto a FIFOx, pari a 32 bit.

3.3.5 Reparto test di canale

Il reparto test di canale è stato introdotto per verificare la correttezza della sezione di trasmissione, isolando l'intero blocco di algoritmi e filtraggio. Questo componente viene abilitato nel momento in cui viene ricevuto il pacchetto di servizio con la keyword di attivazione, propagata a tutta l'architettura tramite il bus interno dei comandi (1001).
Durante l'esecuzione del test i dati ricevuti vengono prelevati dalla memoria FIFO di entrata (FIFO_x) e scritti direttamente in quella di uscita (in FIFO_e) da un processo determinato dallo stesso clock del reparto logica: l'utilizzo dei due clock distinti è utile per riprodurre l'asincronicità delle letture-scritture data dalla comunicazione dei due reparti in fase di simulazione HIL.
Accumulati i dati nella FIFO di uscita se i pacchetti di risposta conterranno gli stessi byte inviati per il test sarà possibile inferire il corretto timing delle operazioni.

Input Output pin

Segnale	Tipo	Descrizione
res	IN 1 bit	Reset asincrono del componente. Reinizializza la macchina a stati per la lettura-scrittura delle FIFO
clk	IN 1 bit	Clock del reparto, sincronizzato con il clock di logica
empty_x_i	IN 1 bit	Segnale di empty della FIFO_x, se basso significa che sono presenti dati da leggere
x_i	IN 16 bit	Bus di lettura dati da FIFO_x
e1_out	IN 16 bit	Bus del segnale e1_out del reparto di logica, instradato se la modalità test è disabilitata
e2_out	IN 16 bit	Bus del segnale e2_out del reparto di logica, instradato se la modalità test è disabilitata
wey_i	IN 1 bit	Segnale di scrittura dato su FIFO_e dal reparto di logica, instradato se la modalità test è disabilitata
rex_i	IN 1 bit	Segnale di prelievo stimolo da FIFO_x dal reparto di logica, instradato se la modalità test è disabilitata

Segnale	**Tipo**	**Descrizione**
STestChActivated	IN 1 bit	Attivo se la modalità di test di canale è abilitata
rex_o	OUT 1 bit	Segnale di comando lettura da FIFO_x
wey_o	OUT 1 bit	Segnale di comando scrittura su FIFO_e
e_bus	OUT 32 bit	Bus di scrittura dato su FIFO_e
empty_x_o	OUT 1 bit	Segnale di presenza dato per la simulazione, prelevato dal reparto di logica

3.3.6 Interfaccia PC

Completato il blocco hardware per la ricezione e invio dei dati è necessario programmare adeguatamente l'altro capo del canale trasmissivo. Il computer viene quindi posto all'interno del sistema HIL interpretando il ruolo di master nella simulazione e nella comunicazione con la FPGA, svolgendo i seguenti compiti:

- Instaurazione del canale comunicazione come master
- Invio della configurazione iniziale alla scheda FPGA
- Invio dei dati di input per la simulazione
- Collezione dei risultati
- Presentazione dei risultati

L'insieme dei requisiti elencati necessita di un ambiente completo che possa offrire sia delle funzioni per la gestione di un canale di comunicazione ad alta velocità (USB High Speed), sia degli accorgimenti grafici per presentare i risultati in maniera altamente leggibile.
Di conseguenza è stato implementato un software di controllo, interfacciato con i driver FTDI per la comunicazione e con MATLAB per mostrare i risultati.

Per il controllo e la realizzazione della simulazione da PC è stato realizzato un programma in C# con tecnologia .NET. La simulazione viene gestita instaurando un canale di comunicazione stabile con la scheda FPGA tramite il chip FTDI: le direttive per la trasmissione sono chiamate dai driver proprietari del dispositivo. Il software oltre a questa funzionalità svolge anche da interfaccia con l'ambiente matlab, utilizzato per la creazione dei segnali di test e la presentazione dei risultati.
In ultimo offre un utility di supporto per la creazione del file degli stimoli per la piattaforma HIL, contenente tutti i pacchetti necessari per il protocollo stabilito.

GUI: Finestra principale

La finestra iniziale del programma presenta una sezione di accesso alle utility, una sezione di simulazione ed uno screen per il debug.
Per la sezione di simulazione:

- I controlli input e output permettono di inserire il riferimento a due file di testo, organizzati in byte. Specificando questi due file il programma avvierà la simulazione prelevando i pacchetti della comunicazione contenuti nel file di input e scriverà i risultati nel file di output.

Figura 3.12: *Form iniziale*

- Il pulsante Config realizza la configurazione del chip, e deve essere utilizzato prima dell'esecuzione di una simulazione.
- Il pulsante Start avvia una nuova simulazione con i file I/O selezionati.

Per la sezione di utility:

- Il controllo File Reader permette l'accesso al form per l'interpretazione ed il plot dei risultati della simulazione, richiamando l'interfaccia grafica di matlab.
- Il pulsante Test effettua un controllo del sistema per l'interpretazione di due pacchetti prova
- Il pulsante Packet Creator apre la finestra per la creazione del file degli stimoli della simulazione.

GUI: Finestra per la creazione del pacchetto

La funzionalità di questo tool è quella di assistere nella creazione di un file di testo contenente tutti i pacchetti necessari alla corretta configurazione ed esecuzione della simulazione.
Il file prodotto sarà quindi utilizzato in fase di test HIL come sorgente dei byte da inviare.
E' altresì importante notare come in fase di validazione esso abbia assunto un ruolo fondamentale nella preparazione di pacchetti di prova per testare il reparto di trasmissione del progetto, testato all'interno dell'ambiente ACTIVEHDL.

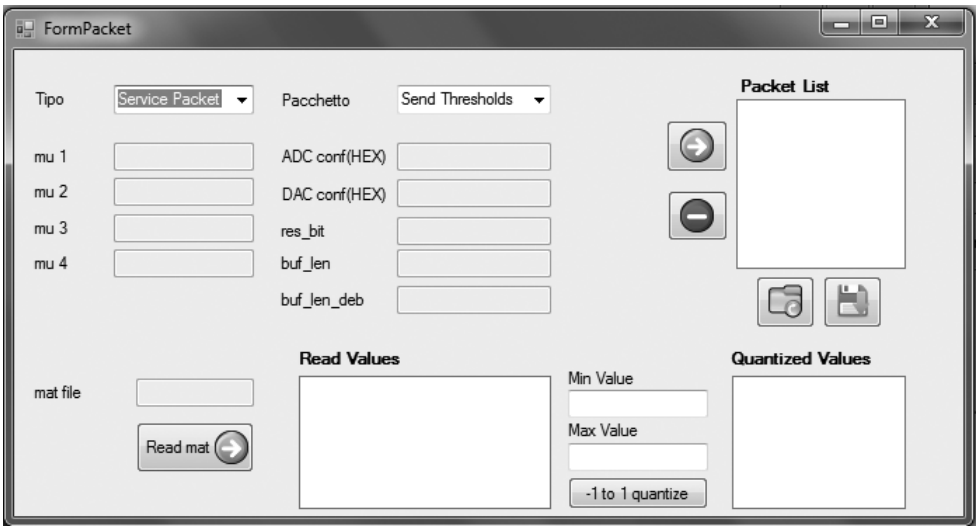

Figura 3.13: *Finestra di creazione pacchetto*

L'immagine mostra la finestra per la creazione dei pacchetti contenenti i dati e le configurazioni necessari alla simulazione, che saranno poi salvati in un byte file di stimoli.
La prima box permette di scegliere il tipo di pacchetto, distinguendo fra service packet e data packet; la seconda box sceglie un pacchetto specifico della macro-categoria impostata.
Nel caso di un pacchetto service non occorre impostare alcuna fonte di dati in quanto tali pacchetti non contengono payload ma solo informazioni di comando e di servizio per la comunicazione.
Nel caso di un pacchetto data verranno attivati i controlli sottostanti necessari a completare i dati richiesti.
In particolare per i pacchetti di segnale e di taps è necessario specificare un file matlab da dove prelevare i valori ed eseguire la quantizzazione seguendo lo schema di controlli Read mat, Quantize.
Una volta completati dati necessari per un pacchetto lo si aggiunge alla lista dei pacchetti premendo il controllo con la freccia di colore verde.
La lista conterrà ordinatamente tutti i pacchetti creati per la simulazione, nel caso di errore è possibile eliminare il pacchetto selezionato con il comando rosso.
Completata la lista dei pacchetti da inviare, premendo l'apposito tasto di salvataggio verranno creati 3 file: uno per la simulazione vera e propria con la board contenente i dati in formato byte (ChosenName.txt); uno per la simulazione in ambiente ACTIVEHDL (ChosenName_bin.txt), in formato stringa codice binario; l'ultimo solo per fini di debug in formato esadecimale (ChosenName_hex.txt).

Di seguito una panoramica dei pacchetti che si possono creare.

Pacchetto	Payload bytes	Controllo input
Segnale x	1024	Input da file matlab
Taps per P1,P2,S11,S12,S21,S22	8-16	Input da file matlab
Mu packet	8	Controlli mu
Bus configuration	6	Controlli cfg
ADC configuration	2	Controllo adc cfg
DAC configuration	2	Controllo dac cfg

GUI: Finestra di lettura risultati

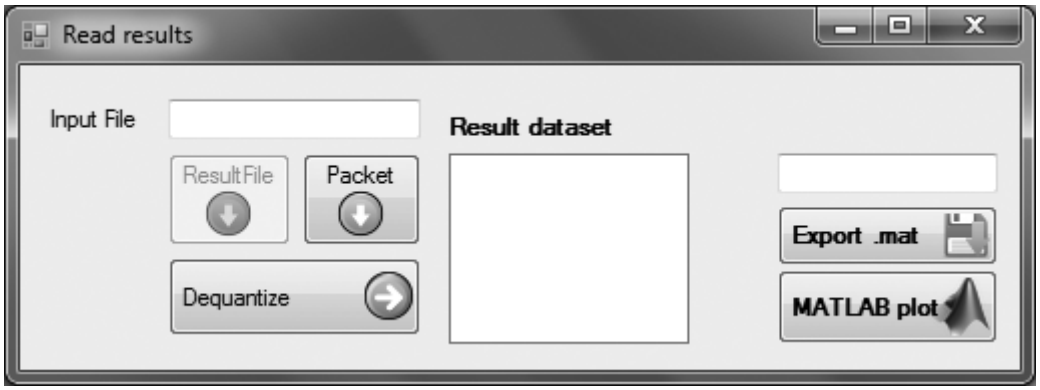

Figura 3.14: *Finestra risultati*

Il form Read File consente l'interpretazione del file di output prodotto dalla simulazione e la sua visualizzazione mediante grafici matlab. Il controllo input file seleziona inizialmente la sorgente dei risultati, premendo poi il pulsante ResultFile o Packet il sistema leggerà ed interpreterà il file differentemente.
Con il controllo resultFile si interpreta una struttura contenente già i dati puri di errore, mentre con Packet una struttura a pacchetto contenente i risultati da parsare.
Successivamente premendo il pulsante dequantize si avvia la procedure di dequantizzazione dei dati da 16 bit a floating point, ottenendo l'output in result dataset. A questo punto è possibile salvare i risultati come mat file, oppure visualizzare il plot di matlab mostrante l'andamento dell'errore.

GUI: Finestra di test

Figura 3.15: *Finestra di test*

Il form di test costituisce un interfaccia per fare un rapido controllo delle funzionalità della scheda.
Il pannello di gestione del chip FTDI presenta i controlli Open/Close del canale con il protocollo FT245: premendo Open viene instaurato il canale con il chip, configurando i parametri di trasmissione secondo gli standard USB; premendo Close si chiude l'handle della trasmissione.
Queste procedure vengono utilizzate per verificare che le chiamate ai driver del chip funzionino correttamente ed il link per la simulazione venga aperto senza problemi.
Il pannello Send packet permette di inviare alla scheda pacchetti singoli: questa funzionalità è stata molto utile in fase di verifica per garantire la corretta interpretazione di tutti i comandi.
Aprendo il menu a tendina si seleziona il tipo di pacchetto da inviare, mentre premendo il tasto send si inviano i bytes al dispositivo; nel caso si aspetti una risposta si seleziona la tick corrispondente, e i bytes ricevuti verranno visualizzati nell'area di testo sottostante.
L'ultimo controllo è rappresentato dal bottone Check bytes: esso controlla se il dispositivo sta trasmettendo dati verso il pc e viene utilizzato per capire se c'è stato un errore nella trasmissione e sono stati lasciati dei dati invalidi nel buffer.

Codice e librerie

Il software per il testing è completamente interfacciato con matlab, per fornire alla simulazione uno strumento avanzato con utility di supporto alla creazione di input, configurazioni e visualizzazione dei risultati.
L'interoperabilità tra .NET e il server matlab è resa possibile dalle librerie COM di matlab e CSMATIO.

MLApp COM

Per eseguire routine e comandi MATLAB è stata utilizzata l'interfaccia COM fornita dalla MATHWORKS, *MLAPP* . Per utilizzare un ambiente COM MATLAB all'interno di C# occorre prestare attenzione in quanto la libreria non verrà inclusa nel pacchetto finale, ma sarà invocata dal path specifico all'interno della macchina in cui è stata installata. E' necessario quindi sfruttare il meccanismo del late binding, senza utilizzare la libreria a compile time ma richiamarla quando utile durante runtime. Implementare il late binding di una funzione COM in C# è una procedura semplice, tuttavia occorre seguire una serie di passaggi chiave spesso tralasciati quando si studia un nuovo ambiente di sviluppo.

- Il primo passo è dichiarare un oggetto generico (object type) per contenere il componenete COM e un type (dalla libreria System.Reflection) per la descrizione della sua struttura. La struttura viene ottenuta con la richiesta al registro dei servizi COM.

- A questo punto è possibile inizializzare la porta COM, attivando il componente con il tipo restituito.

- Per accedere ad una funzione dell'interfaccia si utilizza il comando InvokeMember, specificando la signature della funzione (Execute) ed i parametri necessari (args):

```
private Object matlab ;
private Type   typ    ;
typ = Type.GetTypeFromProgID('Matlab.Application');
matlab = Activator.CreateInstance(typ);
ris = typ.InvokeMember('Execute', BindingFlags.InvokeMethod,
                            Type.DefaultBinder, matlab, args);
```

In questo esempio viene chiamata la funzione Execute del server matlab, passando come argomento una stringa con il comando matlab verrà eseguita come se scritta su console.
Il risultato, se presente, verrà restituito in formato stringa nella variabile ris.

CSMatIO

La libreria CSMatIO fornisce le routine per la lettura e scrittura di matrici matlab in software .NET.
Nell'applicazione sviluppata questa libreria viene utilizzata per scambiare dati fra i due ambienti, richiamando le sottolibrerie types e io. La sezione *types* contiene le definizioni dei tipi base matlab, organizzati secondo la medesima struttura utilizzata dal programma: basandosi sulla natura dei dati della simulazione sono stati utilizzati 4 tipi fondamentali.

Definizione	**Descrizione tipo**
MLArray	Classe base con un array generico di matlab
MLCell	Elemento singolo di una matrice, interpretato come char
MLInt16	Array 2D di interi a 16 bit, per l'algoritmo in fixed point
MLDouble	Array 2D di double, riquantizzati da 16 bit per la visualizzazione dei risultati

La sezione *io* fornisce le funzioni per importare ed esportare gli ambienti matlab da file .mat.
La funzione MatFileReader legge ed interpreta i dati salvati nei file .mat: secondo la procedura consueta i dati vengono letti in un tipo generico di array e poi interpretati secondo la specifica natura.
Sono stati utilizzati segnali double organizzati secondo vettori colonna:

```
MatFileReader mfr = new MatFileReader('mat_file.mat')
foreach (MLArray mla in mfr.Data)
{
  matrix = (mla as MLDouble).GetArray();
  MatDATA[j] = matrix[0];
  j++;
}
```

La funzione MatFileWriter crea un file .mat con i dati esportati dal programma .Net: utile per leggere i risultati su un plot matlab.
L'ambiente viene creato con una lista di matrici, ognuna identificata con un nome ed un tipo.

```
  double[] src = new double[] { 1.0, 2.0, 3.0 };
  MLDouble mlDouble = new MLDouble( "d_arr", src, 3 );
  MLChar mlChar = new MLChar("char_arr","dummy char");
  List<MLArray> list = new List<MLArray>();
  list.Add(mlDouble);
```

```
  list.Add(mlChar);
  new MatFileWriter("mat_file.mat",list,true);
```

Driver del chip FT2232H

Per gestire la comunicazione con il chip FTDI sono stati utilizzati i driver proprietari [3] compilati per ambiente .NET, FTD2XX_NET.dll .
La classe principale - FTDI() - rappresenta la versione virtuale del modulo e contiene il set completo delle funzioni accessibili del dispositivo.
Premendo il pulsante Config del form iniziale si innescano le chiamate alle procedure di rilevamento dispositivo e configurazione.

```
Pseudocodice onConfigClick()
 ... Istanza classe
 FTDI myFtdiDevice = new FTDI();
 ... Numero di chip connessi
 myFtdiDevice.GetNumberOfDevices(ref ftdiDeviceCount);
 ...
 myFtdiDevice.GetDeviceList(ftdiDeviceList);
 ... Apertura modulo
 myFtdiDevice.OpenBySerialNumber(ftdiDeviceList[0].SN);
 ... Baud Rate della comunicazione
 myFtdiDevice.SetBaudRate(9600);
 ... FT245 Sync Style
 myFtdiDevice.SetBitMode(0,FT_BIT_MODE_SYNC_FIFO);
```

A termine della procedura di configurazione viene instaurato il canale di trasmissione: il pc comunica con il chip con funzioni di tipo read e write.

```
SimulationStart()
... Scrittura del buffer toWrite
device.Write(toWrite[],toWrite.Length,ref BytesWritten);
...
... Bytes disponibili
device.GetRxBytesAvailable(ref BytesAvailable);
... Lettura
device.Read(toRead[],BytesAvailable,ref BytesRead);
```

Capitolo 4

Risultati e sviluppi futuri

In questo capitolo vegono esposti i risultati del progetto: la piattaforma é stata validata tramite simulazione funzionale in ambiente software e tramite simulazione operativa hardware in the loop.

4.1 Simulazione funzionale VHDL

Prima di configurare la scheda FPGA e procedere con il download del codice VHDL é stata condotta una simulazione interamente lato software, per validare il codice scritto a livello funzionale.
Scopo di questa simulazione é fornire uno strumento per la prima verifica dl reparto di trasmissione, che possa far emergere gli eventuali errori nella struttura logica del sistema.

4.1.1 Introduzione ActiveHDL e configurazione

Per la simulazione dei blocchi VHDL é stato scelto l'ambiente ACTIVE HDL della Aldec.
Questo tool rappresenta una valida opzione per la realizzazione di progetti hardware in quanto fornisce tutti gli strumenti necessari per:

- Scrivere e comporre blocchi funzionali VHDL
- Generare automaticamente testbench e test vector
- Simulare le top-entity con i testbench specifici
- Analizzare i risultati mediante grafici e onde

Inizializzazione ambiente

Per testare efficacemente il codice VHDL del sistema occorre fornire al simulatore un file di stimoli contenente una comunicazione tipo, ovvero l'insieme di pacchetti da trasmettere.
A questo proposito il software scritto in C# contiene un utility di supporto a questo problema: la finestra Packet Creator permette la creazione di un file di testo contenente i pacchetti di stimolo necessari per eseguire la simulazione funzionale.
Il passaggio successivo consiste nell'importare il file nell'ambiente ACTIVEHDL. Per realizzare la lettura di un file in un testbench serve innanzitutto includere le librerie std.textio.all e txt_util.all. In secondo luogo viene definito un processo per la lettura del file: il processo leggerá un byte per volta, presentandolo su usb_dbus nel momento in cui il comando di RD é abilitato.

```
while not endfile(stimulus) loop
 readline(stimulus, l);              -- Read stimulus
 read(l, s);                         -- Parse line
 usb_dbus <= to_std_logic_vector(s); -- Assign stimulus
 wait until cl60M = '1'              -- Wait until next read
       and usb_rd = '0';
```

Allo stesso modo durante la simulazione vengono collezionati i risultati dell'algoritmo su file per poi analizzarli tramite matlab.

```
file resultse1: TEXT open write_mode is "resp_bine1.txt";
   uno :=to_bitvector(e1_out);
   write(line_one,uno);
   writeline(resultse1,line_one);
   ...          -- same for e2_out
```

4.1.2 Test di canale

Il test di canale conduce la verifica del reparto trasmissivo, realizzato distaccando il blocco che implementa l'algoritmo.
In questa procedura vengono validati i seguenti aspetti:

- Invio e ricezione di due pacchetti di tipo dato della lunghezza di 1024 byte, con l'interpretazione/parsing dei dati
- Scrittura nella prima FIFO per l'accumulo dei dati
- Passaggio dalla FIFO di ingresso e accumulo nella FIFO di uscita
- Prelievo per trasmissione su pacchetto e costruzione del pacchetto di risposta.

Ricezione e parsing dei dati

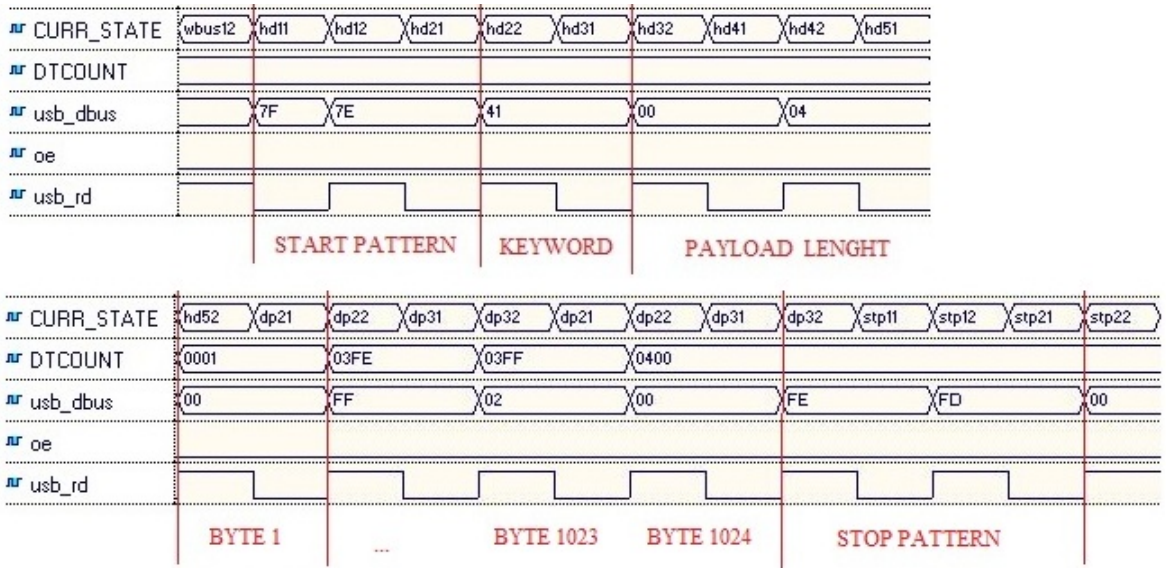

Figura 4.1: Simulazione ActiveHDL

Il diagramma temporale mostra l'evoluzione dei segnali per il test. Il processo di lettura campiona il dato su usb_dbus portando il segnale usb_rd a livello logico basso, e procede nello stato descritto in CURR_STATE.
La simulazione mostra la successione di tutte le sezioni del pacchetto, ovvero: lo start pattern, la keyword di pacchetto, la lunghezza del payload, il payload e lo stop pattern.
In particolare viene calcolata correttamente la lunghezza del payload, che determina la fine del pacchetto tramite il contatore dati DTCOUNT.

Scrittura/Lettura FIFO

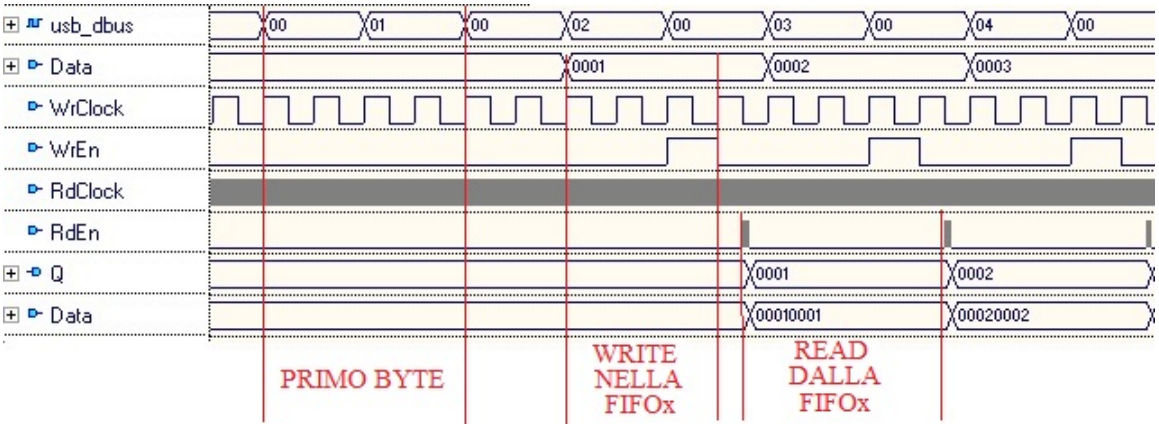

Figura 4.2: FIFOx

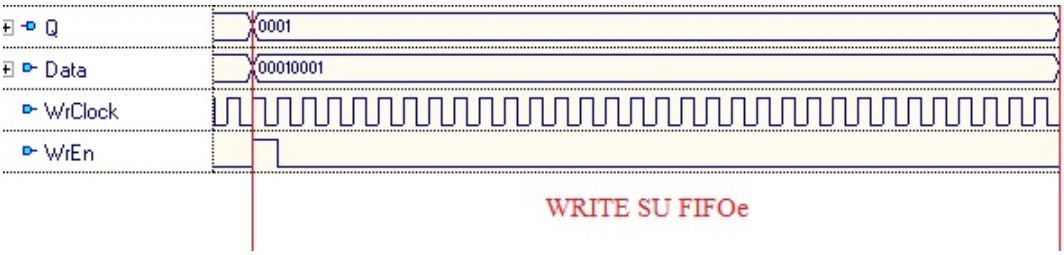

Figura 4.3: FIFOe

Il test del canale ha il compito di verificare il percorso seguito dai dati dall'ingresso con la ricezione via USB, alla trasmissione dallo specifico componente.
In questa simulazione viene validato il percorso intermedio fra la FIFO posta per gestire i dati in ricezione (FIFOx) e la FIFO per memorizzare i risultati prima della trasmissione (FIFOe).
Nella prima figura viene mostrato la ricezione dei dati su usb_dbus e la memorizzazione nella FIFOx. Il dato viene poi prelevato con il comando read e riscritto nella FIFOe di uscita, in attesa della costruzione del pacchetto di risposta.

Pacchetto di risposta

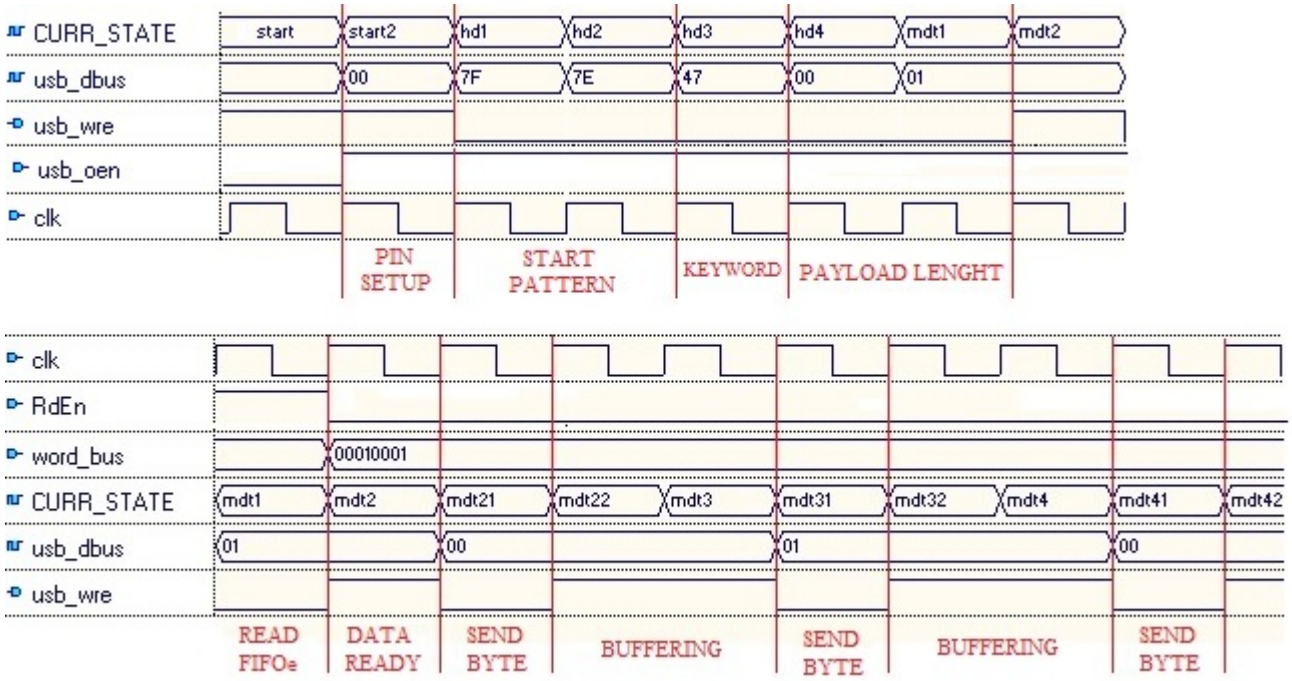

Figura 4.4: Composizione pacchetto

L'ultimo passaggio della simulazione consiste nella verifica dell'invio di un pacchetto verso il computer, tramite il componente TX_CTR.
Il primo diagramma mostra i passaggi compiuti dalla macchina a stati per comporre l'header del pacchetto: nello stato START2 viene preparato il chip per la trasmissione, configurando i pin in alta impedenza con il segnale usb_oen a 1; gli stati HD1-HD2 scrivono sul bus dati usb_dbus i due byte che componegono la trama di start (7f7e), comandando la scrittura pilotando il segnale usb_wre basso; nello stato HD3 viene scritta la keyword identificativa del pacchetto, in particolare x47 indica dati della FIFO contenente l'errore residuo; gli ultimi due byte dell'header rappresentano la lunghezza del payload del pacchetto (0001) in formato little endian, pari a 256.
Il secondo diagramma descrive la composizione del payload dati: nello stato mdt1 viene prelevato un dato a 32 bit dalla FIFO dell'errore residuo, il valore viene campionato in word_bus a segnale RdEn alto (mdt2); a questo punto in mdt21 viene scritto il primo byte sul bus, mentre in mdt22 e mdt3 viene preparato il prossimo byte nella pipeline per la scrittura; la scrittura del dato avviene poi in mdt31, e si procede nella pipeline fino a quando tutto il dato a 32 bit é stato scritto.

4.2 Risultati sperimentali

La simulazione sul campo con la scheda FPGA ha ottenuto risultati significativi per quanto riguarda il funzionamento della connessione HIL; le performance dell'algoritmo FxLMS, inizialmente deludenti, sono state migliorate a seguito di correzioni implementative, introdotte in un secondo momento a seguito di problemi sulla convergenza del sistema.
Collegata la scheda con il computer, i passaggi chiave della simulazione HIL sono stati eseguiti correttamente: i comandi di simulazione del computer vengono ricevuti dal blocco apposito, interpretati, ed eseguiti per gestire il sistema; i segnali dato inviati vengono memorizzati nella memoria FIFOx, buffer di ingresso, ed asincronicamente letti dal componente di ANC, il quale li elabora per il calcolo della risposta del sistema e memorizza l'errore residuo stimato nel buffer di uscita; il comando di download dei risultati trasferisce i dati dal buffer di uscita al computer tramite i pacchetti creati dal componente di trasmissione.
Per quanto riguarda la risposta dell'algoritmo di cancellazione sono stati riscontrati dei problemi di convergenza, dovuti alla lunghezza del filtro adattivo. A questo proposito viene presentata una panoramica della simulazione effettuata che ha evidenziato tale problema ed il metodo per risolverlo.

4.2.1 Primo test del sistema

Per il primo test eseguito sulla board é stato scelto di stimolare l'algoritmo con un rumore bianco, un segnale a valor medio e autocorrelazione nulla. I percorsi primari, secondari e i guadagni del sistema sono stati configurati tramite i pacchetti specifici, utilizzando il setup dei pesi da ambiente Simulink:

Pacchetto	**Dati**
Segnale di stimolo x	Rumore bianco
Guadagni sistema	Offline: 0,0001 - Online: 0,01
Percorso primario P1	0,01 - 0,25 - 0,50 - 0,99 - 0,5 - 0,25 - 0,01
Percorso primario P2	0,01 - 0,29 - 0,58 - 0,99 - 0,58 - 0,29 - 0,01
Percorso secondario S11	0,0025 - 0,0625 - 0,125 - 0,250 - 0,125 - 0,0625 - 0,0025
Percorso secondario S12	0,0035 - 0,0875 - 0,175 - 0,350 - 0,175 - 0,0875 - 0,0035
Percorso secondario S21	0,0030 - 0,0750 - 0,150 - 0,300 - 0,150 - 0,0750 - 0,0030
Percorso secondario S22	0,0040 - 0,1000 - 0,2000 - 0,4000 - 0,2000 - 0,1000 - 0,0040

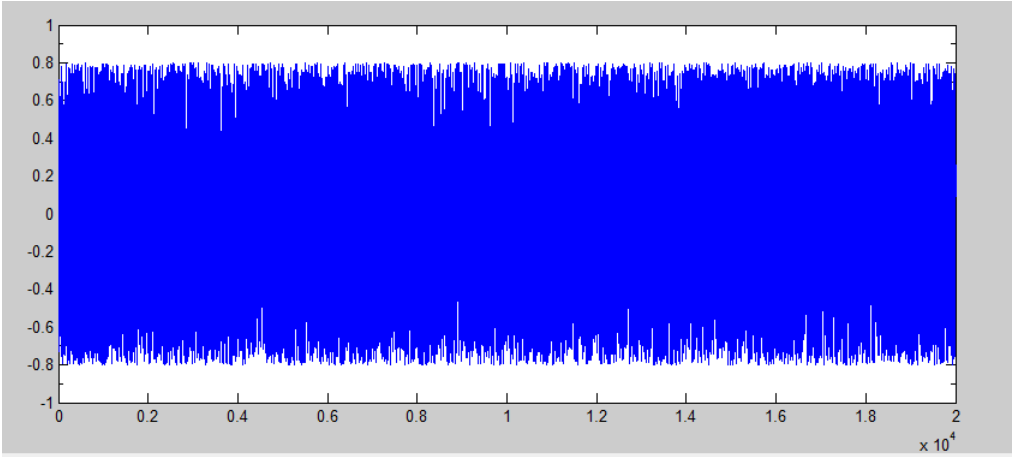

Figura 4.5: Segnale di disturbo

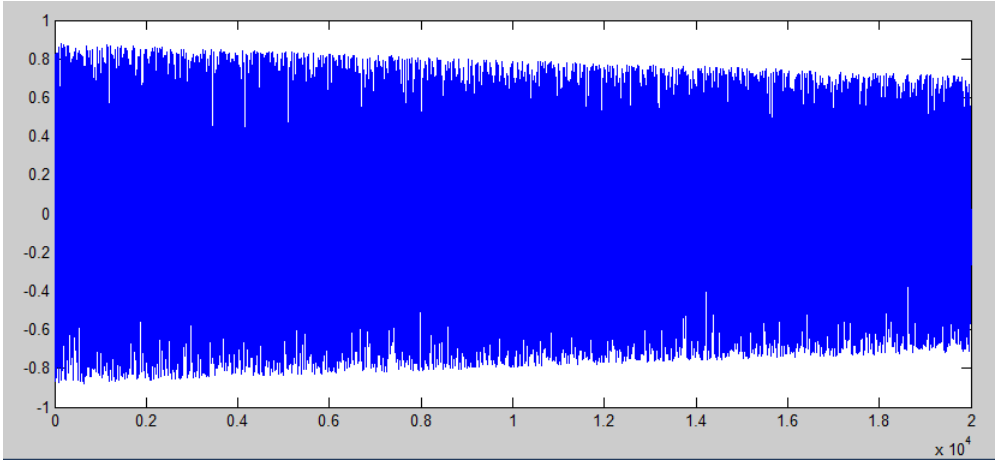

Figura 4.6: Errore residuo

Il grafico mostra i risultati dati dai pacchetti di risposta della scheda, contentente l'errore residuo del sistema a fronte del segnale di disturbo originale riportato nel grafico soprastante. Si puó notare come l'algoritmo non converga e l'errore residuo sia di pari intensitá rispetto al segnale originale. Per raggiungere un miglioramento nelle prestazioni é stato rianalizzato l'insieme dei parametri di configurazione dell'algoritmo FxLMS.

Convergenza dell'algoritmo in virgola mobile

Per sfruttare le potenzialitá dell'algoritmo FxLMS in maniera ottimale occorre prestare attenzione a due parametri di configurazione ovvero:

- La lunghezza del filtro L, ovvero il numero di tap della convoluzione;
- Il guadagno dell'equazione di aggiornamento dei pesi.

La lunghezza L fornisce innanzitutto un'idea della complessitá dell'algoritmo: la convoluzione che calcola il segnale di risposta y necessita L moltiplicazioni e L-1 somme;

$$y(n) = \mathbf{w}(n)^T * \mathbf{x}(n) = \sum_{l=0}^{L-1} w_l(n)x(n-l) \tag{4.1}$$

Inoltre L definisce in modo sostanziale le performance di convergenza dell'algoritmo di LMS. A questo proposito sono state compiute delle simulazioni in matlab al variare del parametro L mantendendo costante rumore, filtri e guadagni:

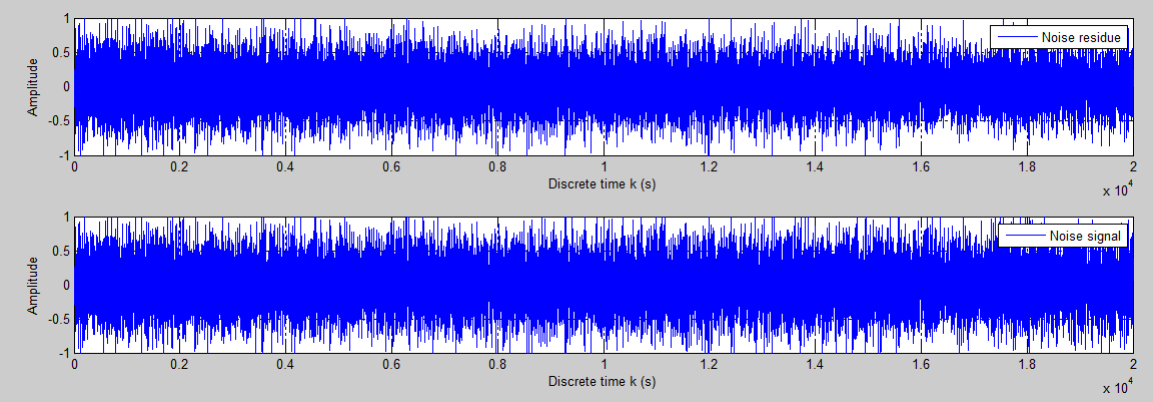

Figura 4.7: *L=8 sottodimensionato, sistema l=25*

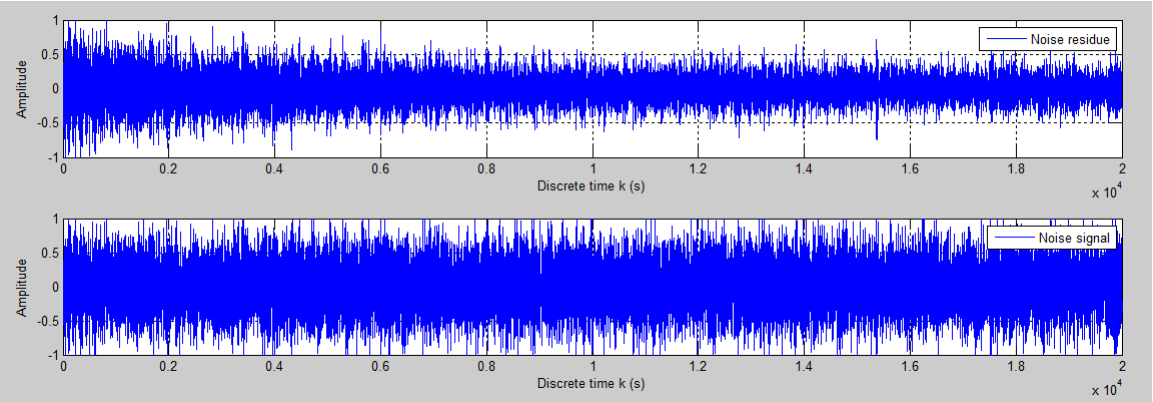

Figura 4.8: *L=32 dimensioni comparabili, sistema l=25*

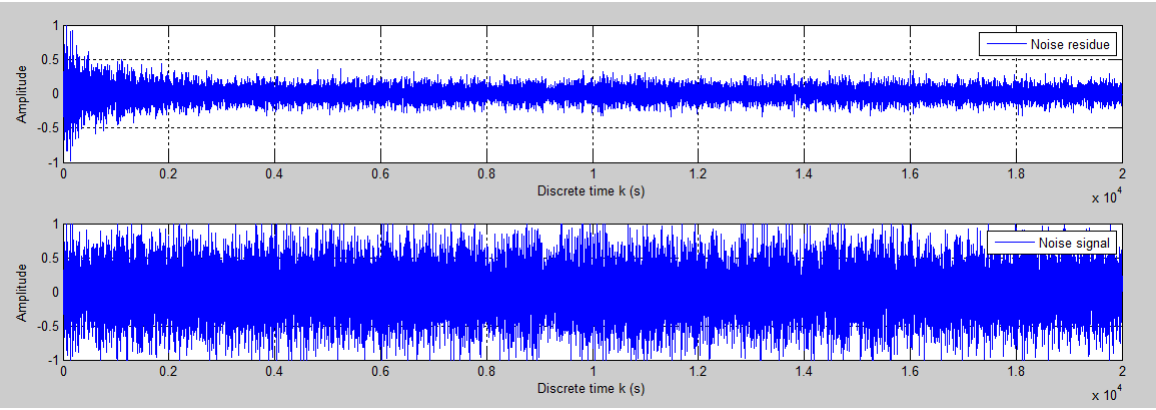

Figura 4.9: *L=50, sistema l=25*

Da una prima analisi dei grafici riportati risulta evidente come una scelta errata di L possa portare a performance di convergenza dell'algoritmo praticamente nulle, anche utilizzando la rappresentazione in virgola mobile. In particolare le simulazioni hanno riscontrato che aumentando il numero di tap del filtro adattivo, l'errore residuo decresce in termini di ampiezza assoluta e varianza. Esiste un limite superiore alla scelta del numero di tap, legato al valore del guadagno dell'algoritmo.
Per calcolare questo limite occorre analizzare la formula di convergenza dell'algoritmo:

$$|1 - \mu\lambda_l| < 1, \qquad l = 0, 1, ..L - 1 \tag{4.2}$$

In questa disequazione λ_l rappresenta l'autovalore alla posizione l della matrice di autocorrelazione del segnale x di ingresso al sistema.
La disequazione si semplifica sull'autovalore di modulo maggiore, ottenendo un vincolo per la scelta del suo valore:

$$\mu < \frac{2}{\lambda_{max}} \tag{4.3}$$

Non conoscendo il a priori la matrice R di autocorrelazione del segnale x, una stima di λ_{max} puó essere calcolata con la potenza media di x:

$$\lambda_{max} < \sum_{l=0}^{L-1} \lambda_l = tr(\mathbf{R}) = LP_x \tag{4.4}$$

Con l'ultima disequazione abbiamo ottenuto un vincolo di progetto che lega il guadagno dell'algoritmo con il numero L dei pesi del filtro:

$$\lambda_{max} < \frac{2}{LP_x} \tag{4.5}$$

Sfruttando questo vincolo é stato deciso di incrementare la lunghezza L del filtro adattivo da 7 tap a 32 tap; la stabilitá di questa scelta é garantita attraverso una scelta accurata di μ.
Va tenuto debitamente conto che le performance di convergenza saranno tuttavia peggiori rispetto al caso floating point, dato che i calcoli dall'architettura sono in rappresentazione a virgola fissa.
Prima di potere testare il sistema é stato necessario un ulteriore step di revisione, dato dalla mancanza di spazio sulla scheda per eseguire calcoli con filtri di lunghezza superiore.

Problematiche di spazio

Aumentando il numero di tap per la convoluzione nei filtri da 7 a 32 il mapper (software dedito alla trasformazione dei blocchi della sintesi VHDL in blocchi fisici nell'hardware) ha negato la compilazione per mancanza di risorse disponibili nella FPGA. Questo problema viene fatto ricondurre alla pesantezza della procedura di convoluzione tra segnali: per calcolare il valore di convoluzione in un determinato istante temporale occorre memorizzare due finestre di 32 valori (pari alla lunghezza dei filtri) a 16 bit; in un'implementazione di tipo parallelo, in cui piú convoluzioni vengono calcolate nello stesso ciclo di clock, il numero di risorse richieste aumenta vertiginosamente. Per liberare spazio nell'architettura é stato necessaria una revisione della modalitá di calcolo delle convoluzioni, per essere eseguite in ordine sequenziale e non parallelo. A questo proposito é stato introdotto il componente moltiplicatore-accumulatore MultAcc, proprietario della Lattice:

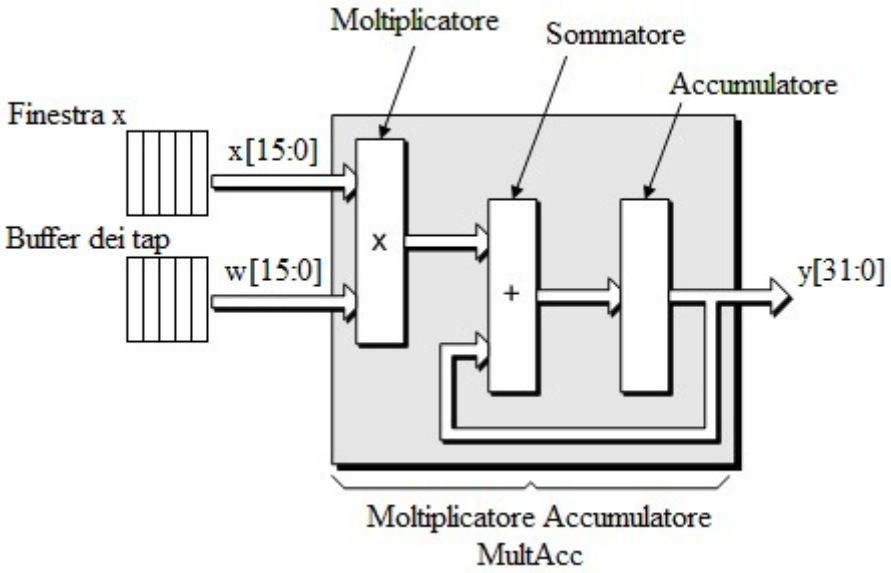

Figura 4.10: Moltiplicatore-Accumulatore

La figura mostra lo schema sequenziale dell'operazione di convoluzione: ad ogni colpo di clock viene prelevata una coppia di operandi [x(i),w(i)] dai buffer di ingresso e posta in ingresso al modulo MultAcc; il componente moltiplica i due operandi e li somma al risultato dell'operazione precedente registrato nell'accumulatore. Dopo L colpi di clock ($tc_1..tc_L$), dove L rappresenta la lunghezza 32 del filtro, il valore nel registro di accumulo rappresenta il risultato della convoluzione tra la finestra temporale di x e il filtro w.

$$[tc_L]y(n) = \sum_{l=0}^{L-1} w_l(n)x(n-l) = [tc_1]w(0) * x(n) + [tc_2]w(1) * x(n-1) + .. \\ +[tc_L]w(L-1) * x(n-L+1) \tag{4.6}$$

Tramite questa revisione le risorse occupate diminuiscono drasticamente, permettendo di implementare filtri fino a 32 tap di risoluzione. Il prezzo da pagare é un rallentamento globale del sistema in quanto, compiendo la convoluzione sequenzialmente e non parallelamente, il tempo necessario per il calcolo passa da 1 colpo di clock a 32 (seguendo il paradigma tempo richiesto/area risorse = costante).

4.2.2 Secondo test del sistema

Il progetto VHDL é stato quindi riconfigurato per implementare convoluzioni su filtri di lunghezza 32, ed é stata ripetuta la simulazione con gli stessi parametri usati in precedenza. Vengono presentati i grafici del segnale di disturbo e il segnale dell'errore residuo.

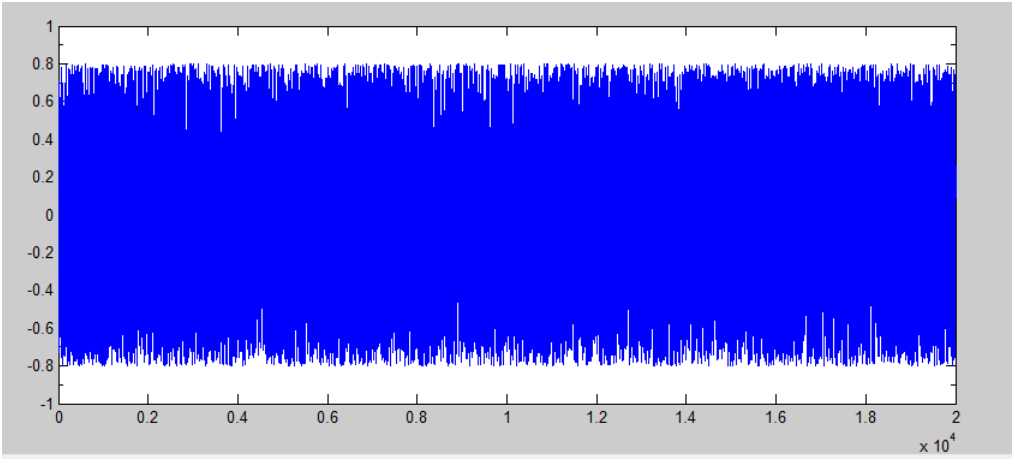

Figura 4.11: Segnale originale di disturbo

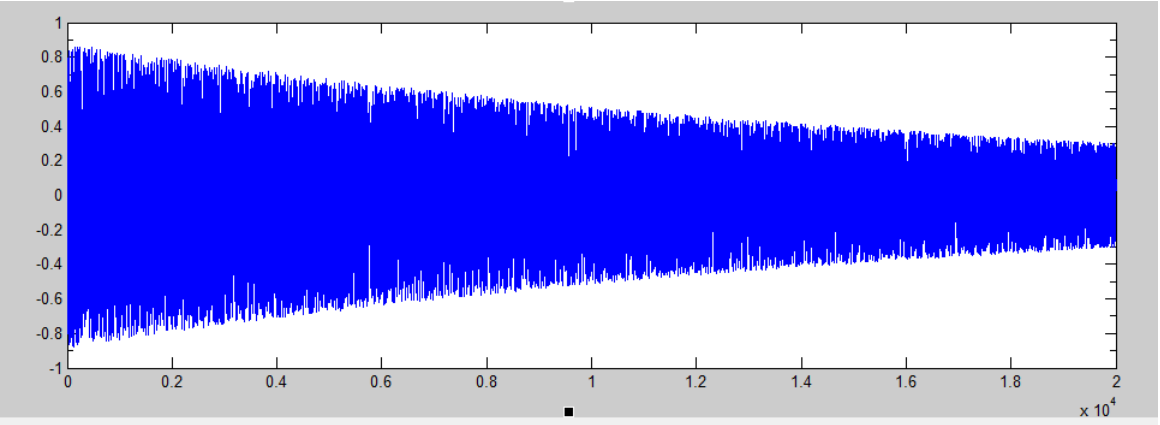

Figura 4.12: Errore residuo

I risultati mostrano come in questo caso ci sia un miglioramento delle prestazioni di convergenza, rispetto all'implementazione precedente.
Secondo le simulazioni fatte in ambiente matlab é possibile migliorare ulteriormente la convergenza dell'algoritmo realizzando convoluzioni con un numero maggiore di tap. Il problema con l'architettura utilizzata é la limitatezza delle risorse, giá occupate al massimo livello con 32 tap: é ipotizzabile che reimplementando lo stesso progetto VHDL con un numero maggiore di tap sará possibile raggiungere livelli di convergenza ottimale dell'algoritmo.

4.3 Conclusioni e sviluppi futuri

Il lavoro presentato e discusso in questa tesi sviluppa gli aspetti principali riguardanti il design, l'implementazione e la verifica di un dispositivo hardware con funzionalitá di controllo attivo del rumore.
L'algoritmo FxLMS elaborato dalla teoria di ANC é stato ridesignato per essere eseguito su una piattaforma FPGA, utilizzando la rappresentazione dei dati in virgola fissa. Per verificare il progetto é stata realizzato un ambiente di simulazione HIL, collegando in anello chiuso la scheda e il computer: in questo modo é stato possibile stimolare il sistema con input controllabili, e verificare gli output di risposta comparandoli con quelli previsti per un corretto comportamento.

4.3.1 Esperienza di progetto

Questo breve riassunto vuole fornire al lettore una panoramica delle considerazioni emerse durante la realizzazione del lavoro di tesi.
Il primo aneddoto riguarda la modalitá di scrittura del codice VHDL per la

progettazione hardware. Nonostante sia un linguaggio molto simile ad un linguaggio per compilare software, il VHDL non ha lo scopo di creare un eseguibile a fine scrittura: il suo obiettivo é essere tradotto in uno schema circuitale con blocchi combinatori e sequenziali fisici, nello specifico sommatori, porte, moltiplicatori, memorie, ecc. Durante la stesura del codice é facile perdere traccia di cosa si sta scrivendo in termini di blocchi logici che saranno istanziati: il lavoro del sintetizzatore e mapper é quello di tradurre il piú fedelmente possibile, senza tener conto della complicatezza dei blocchi che si vengono a creare. Per questo motivo é utile a suddividere la progettazione in tanti blocchi funzionali anche semplici, compilarli tramite il sintetizzatore ed analizzare come e cosa istanziano a livello fisico. In questo modo, giunti alla creazione dell'architettura complessa sará possibile visualizzarla come un insieme di blocchi di struttura nota, facilmente interpretabile e, dove necessario, correggibile.
Parlando della progettazione del framework HIL, una parte molto significativa é stata la scelta del collegamento tra la scheda e il computer. Sono state infatti vagliate diverse alternative in termini di complessitá, tempo di realizzazione, performance, costo. Una ricerca accurata su internet ha dimostrato come un modulo ready-to-use per l'interfacciamento possa essere allo stesso tempo la soluzione piú rapida, ma anche con adeguate performance e a basso costo, rispetto ad un'alternativa complessa (PCI-express) con un livello di performance di poco superiore.
Concluso il progetto, durante la fase di test del reparto di trasmissione della scheda é stato impiegato molto tempo per la correzione di problemi legati alla sincronizzazione con il modulo adattatore della FTDI. I pacchetti ricevuti sul computer e trasmessi dalla FPGA risultavano in parte corrotti e in parte diseallineati senza alcun motivo apparente. Per risolvere il problema é stato necessario applicare una politica di bufferizzazione delle scritture dei dati, sfruttando i periodi validi del dispositivo per trasmettere: questo perché i segnali di controllo del protocollo giungevano al centro della FPGA con ritardo consistente, invalidando alcune scritture. Bufferizzando i dati é stato possibile interrompere l'invio dei dati durante i cicli invalidi e riprendere dalla corretta posizione del buffer una volta sincronizzati.

4.3.2 Sviluppi futuri

Il prototipo hardware realizzato, contornato dal framework HIL software, fornisce l'elemento chiave per la progettazione e la messa in opera di un dispositivo elettronico digitale per la riduzione del rumore acustico.
In particolare, partendo dalla struttura VHDL del prototipo, sará possibile reimplementare il sistema per eseguire convoluzioni con filtri di lunghezza superiore a 32: aumentando la lunghezza del filtro si porterá il sistema a raggiungere la convergenza ottimale dell'errore residuo, necessaria per utilizzare

l'algoritmo nell'applicazione di ANC proposta; da lato hardware questo procedimento non é realizzabile nell'immediato in quanto necessita dell'acquisto di una nuova scheda FPGA, con maggiori risorse di elaborazione rispetto a quella precedentemente utilizzata.
Una volta realizzato il secondo prototipo si potrá procedere con la messa a punto dell'hardware nell'ambiente proprio di lavoro, connettendo il sistema all'impianto di microfoni e altoparlanti: in questa modalitá di funzionamento il segnale di input sará rappresentato dal rumore della ventola, prelevato dal microfono apposito; il segnale di risposta calcolato dell'algoritmo verrá convertito dal reparto DAC e diretto agli altoparlanti che produrranno il segnale di cancellazione.
Altro sviluppo di questa applicazione é la riconfigurazione della scheda con un diverso algoritmo di ANC: implementando un altro tipo di algoritmo é infatti possibile valutare le differenze con il FxLMS qui descritto in termini di performance a fronte degli stessi input; l'obiettivo sará quello di progettare una piattaforma di cancellazione altamente flessibile, in cui si potrá scegliere la configurazione piú adatta alle condizioni specifiche in cui deve operare.
Analizzando la generalitá dell'architettura di simulazione HIL si deduce la proprietá di riutilizzo dell'ambiente di test: é possibile impiegare il framework per la validazione di altri progetti, connettendo dispositivi hardware in fase di test al computer tramite l'aggiunta dei blocchi di trasmissione utilizzati in questo progetto. La loro funzionalitá sará infine verificata conducendo una simulazione HIL con il software giá predisposto, interpretando i pacchetti di risposta dell'hardware a stimoli programmati da computer.

Bibliografia

[1] FTDI Chip. *FT2232H Dual High Speed USB to Multipurpose UART FIFO IC*. 2010.

[2] FTDI Chip. *FT2232H Mini Module USB Hi-Speed FT2232H Evaluation Module*. 2010.

[3] FTDI Chip. *Software Application Development D2XX Programmers Guide*. 2010.

[4] Breton et al. A simple and complete usb interface package for test bench development. *2007 IEEE Nuclear Science Symposium Conference*, 2007.

[5] Di Stefano et al. Efficient fpga implementation of an adaptive noise canceller. *Proceedings of the Seventh International Workshop on Computer Architecture for Machine Perception*, 2005.

[6] Elliott et al. Active noise control. *IEEE signal processing magazine*, 1993.

[7] Fohl et al. A fpga-based adaptive noise canceling system. *University of Applied Science, Faculty TI*, 2009.

[8] Paiz et al. Hardware-in-the-loop simulations for fpga based digital control design. *Heinz Nixdorf Institute, University of Paderborn*, 2008.

[9] Sallberg et al. Active noise control for hearing protection using a low power fixed point digital signal processor. *Proceedings of the International Workshop on Acoustic Echo and Noise Control*, 2005.

[10] Wiśniewski et al. Synthesis of compositional microprogram control units for programmable devices. *Zielona Góra: University of Zielona Góra*, 2009.

[11] Dennis R. Morgan. An analysis of multiple correlation cancellation loops with a filter in the auxiliary path. *General Electric Company, Electronics Laboratory*, 1980.

[12] University of Toronto. *FPGA Architecture for the Challenge*. 2012.