POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Implementazione e test su FPGA di un sistema di comunicazione a eventi per applicazioni robotiche

Relatori:
prof. Maurizio MARTINA
dott. Paolo MOTTO ROS

Candidato:
Antonio PARADISO

Luglio 2021

Sommario

Il paradigma di tipo event-driven è di fondamentale importanza in ambito robotico al giorno d'oggi per ridurre al minimo latenza, banda, potenza e risorse computazionali.
Un sensore che lavora secondo questo principio segnala, in maniera asincrona, una variazione della grandezza fisica misurata senza dover attendere precisi istanti di campionamento come avviene per sistemi di tipo clock-driven.
Il senso del tatto nei robot può beneficiare ampiamente di questo modello, anche di più rispetto alla percezione visiva come dimostrato da diversi studi.
Questo lavoro di tesi si focalizza sulla parte di comunicazione tra due strutture hardware event-driven necessarie per il processamento di dati sensoriali tattili per il robot umanoide iCub [15]: il Tactile Hub, una unità che produce gli eventi e un'altra che riprende alcune delle funzionalità della HPU, ovvero l'unità di processamento dei dati di iCub, connesse tra di loro tramite il protocollo seriale asincrono AS-AER.
Il Tactile Hub acquisisce dati campionati clock-based, genera e spedisce gli eventi attraverso il bus AS-AER oltre ad unire il flusso di eventi locali con il flusso di eventi di altri dispositivi. L'altro modulo acquisisce, deserializza, applica un timestamp e carica i dati nella memoria della CPU.
È stata allestita una configurazione che comprende una FPGA, un modulo FTDI per la configurazione dei registri del Tactile Hub tramite protocollo I2C e un convertitore UART-USB per consentire la comunicazione con il PC.
Il primo obiettivo di questa tesi è la verifica del corretto funzionamento del setup allestito, indipendentemente da iCub.
Per simulare gli stimoli tattili vengono utilizzati dei generatori interni al Tactile Hub mentre gli eventi, comprensivi di timestamp, sono spediti tramite UART al PC e interpretati da un programma C.
Come secondo obiettivo di questa tesi, un trasmettitore e ricevitore seriali basati su codifica data-strobe, che utilizzano solo due fili per la comunicazione (data e strobe), sono stati progettati, per essere poi utilizzati per scopi di debug.
Questi due sistemi sono stati verificati e testati attraverso un apposito testbench su FPGA; in particolare, per caratterizzare i due moduli hardware progettati è stata creata una struttura contenente trasmettitore e ricevitore, un LFSR modificato per generare i dati da trasmettere, un circuito di comparazione per verificare l'uguaglianza tra il dato trasmesso

e ricevuto e una serie di contatori per tenere traccia di eventuali errori. Sono state effettuate 2^{32} trasmissioni di dati su 32 bit per testare che ogni pattern trasmesso fosse ricevuto correttamente, effettuando prove utilizzando un clock al trasmettitore nel campo 70-180 MHz, tenendo fissa la frequenza di lavoro del ricevitore a 200 MHz; non si sono ottenuti errori per il range di frequenze al trasmettitore [76-147] MHz.
I risultati ottenuti dal setup allestito, invece, hanno fornito buoni risultati, in linea con quanto previsto.

Indice

Sommario I

I Introduzione 1

1 Introduzione 2
1.1 Sensori a eventi . . . 3
1.2 L'utilità della pelle artificiale nei robot . . . 4
1.3 Sensori tattili artificiali . . . 5
1.4 Design del polpastrello di iCub . . . 7
1.5 Protocollo AER . . . 9
1.6 Protocollo seriale asincrono AS-AER per sistemi a eventi . . . 10
1.7 Sistema tattile a eventi di iCub . . . 15
1.7.1 AD7147 . . . 19
1.7.2 Codifiche a eventi per sensori tattili clock-based . . . 24

II Implementazione e test del setup allestito 27

2 Implementazione e descrizione dei moduli utilizzati 28
2.1 Tactile Hub . . . 28
2.1.1 Protocollo I2C . . . 30
2.2 AS-AER2UART . . . 31
2.3 Protocollo di handshake . . . 32
2.4 Hardware utilizzato . . . 33
2.5 Implementazione del setup sperimentale . . . 34
2.5.1 Registri del Tactile Hub . . . 36
2.5.2 Scrittura registri con PyFtdi . . . 38
2.5.3 Pacchetti generati e trasmessi dal Tactile Hub . . . 41
2.5.4 Verifica dei dati ricevuti al PC . . . 45
2.6 Risultati . . . 48

3 Test **50**
3.1 Test del modulo AS-AER2UART 50
3.2 Test del Tactile Hub 53
3.3 Test del ricevitore AS-AER 54
3.4 Test del ricevitore data-strobe 55
3.5 Risultati 55

III Trasmettitore e ricevitore data-strobe **56**

4 Trasmettitore e ricevitore data-strobe **57**
4.1 Trasmettitore 58
4.2 Ricevitore 61
4.3 Testbench Modelsim 65
4.4 Testbench su FPGA 67
4.4.1 Configurazione allestita per il testbench su FPGA 72
4.4.2 Risultati sperimentali 75

5 Conclusioni e prospettive future **78**

Bibliografia **80**

Elenco delle figure

1.1 Polpastrello di iCub: a) disegno CAD del polpastrello, b) – e) mostrano diversi stadi della manifattura di un prototipo del polpastrello [16]. 8
1.2 Protocollo AER [8]. 9
1.3 Diagramma a blocchi del transceiver AS-AER [19]. 11
1.4 Esempi di codifica per mostrare le differenze tra il protocollo AS-AER e codifiche simili [19]. 12
1.5 Diagramma a blocchi del trasmettitore AS-AER [19]. 13
1.6 Diagramma a blocchi del ricevitore AS-AER [19]. 13
1.7 Curva d'errore in funzione della differenza relativa di clock al TX e all'RX, sia con accoppiamento AC che DC [19]. 14
1.8 Infrastruttura sensoriale tattile a eventi sul robot iCub [5]. 16
1.9 Integrazione della tecnologia tattile a eventi su iCub [5]. 17
1.10 Diagramma a blocchi del sistema tattile a eventi descritto [5]. 18
1.11 Spazio di indirizzamento per il protocollo AS-AER [5]. I sottospazi di indirizzamento sono allocati gerarchicamente e dipendono dal massimi numero di sottoelementi. 18
1.12 Diagramma a blocchi funzionale dell' AD7147 [2]. 19
1.13 Principio di funzionamento di rilevazione tattile dell'AD7147 [2]. 20
1.14 Variazione del valore contenuto nel registro CDC_RESULT_Sx e soglie di attivazione del sensore [2]. 21
1.15 Diagramma temporale raffigurante l'interrupt di completa conversione [13]. 23
1.16 Calcolo della deriva della temperatura[13]. 23
1.17 [13]. 23
1.18 Corrispondenza valore - dato trasmesso [13]. 24
2.1 Schema ad alto livello del Tactile Hub [18]. In grassetto è scritto il nome dei moduli, in basso a destra è presente una legenda. 28
2.2 Rappresentazione ad alto livello del modulo AS-AER2UART. 31
2.3 Protocollo di handshake tra due moduli, rispettivamente sorgente e destinazione 32
2.4 Modulo TE0712 della Trenz 33
2.5 Modulo TE0701-05 della Trenz 33

2.6 Xilinx debug card FMC XM105 34
2.7 Modulo FTDI 2232H mini module 34
2.8 MMCM globale per i moduli Tactile Hub e AS-AER2UART. 35
2.9 Schema del setup sperimentale allestito che comprende Tactile Hub, AS-AER2UART, modulo FTDI, trasmettitore UART e PC. 35
2.10 Setup allestito che comprende la carrier board con trasmettitore UART, l'FPGA, il modulo FTDI e il PC 38
2.11 Struttura del pacchetto dati AER 42
2.12 Assegnazione dei bit nel pacchetto dati AER 42
2.13 Assegnazione dei bit del payload del pacchetto di tipo indirizzo 43
2.14 Assegnazione dei bit del payload del pacchetto di tipo sample 43
2.15 Assegnazione dei bit del payload del pacchetto di tipo evento di errore . . 44
2.16 Assegnazione dei bit del pacchetto degli eventi del Tactile Hub 44
2.17 Divisione in campi dei pacchetti ricevuti dal PC. 45
2.18 Estratto dell'output del programma C che interpreta i pacchetti ricevuti. . 48
3.1 Output prodotto dal modulo AS-AER2UART, con dati generati dal dummy generator interno, acquisito da PC tramite UART. 52
3.2 Circuito utilizzato per testare il passaggio dei dati tra i moduli del Tactile Hub 53
3.3 In **viola** è mostrato il segnale trasmesso, in **blu** la and di tutti i segnali d'errore, in **giallo** la and tra il srdy e il drdy. 54
3.4 In **viola** è mostrato il segnale trasmesso, in **blu** il srdy e in **giallo** il drdy. . 55
4.1 Esempio di evoluzione dei segnali di `data`, `strobe` e clock ricostruito (xor dei segnali precedenti) nel tempo. 57
4.2 Rappresentazione ad alto livello del trasmettitore. 58
4.3 Rappresentazione RTL del trasmettitore. 59
4.4 Macchina a stati finiti trasmettitore. 60
4.5 Rappresentazione ad alto livello del ricevitore. 61
4.6 Rappresentazione RTL ricevitore 62
4.7 Macchina a stati finiti ricevitore 64
4.8 Frammento di testbench su modelsim di una trasmissione completa 66
4.9 Frammento di testbench su modelsim di una ricezione completa 67
4.10 Rappresentazione ad alto livello del modulo error detector utile ad effettuare delle statistiche sugli errori rilevati in ricezione. 68
4.11 Rappresentazione RTL della logica per effettuare il testbench su FPGA . . 69
4.12 Macchina a stati finiti testbench su FPGA 70
4.13 LFSR 8 bit 71
4.14 Rappresentazione ad alto livello del testbench su FPGA allestito: sono presenti i due moduli di trasmissione e ricezione connessi a un generatore di dati e al di circuito error detection. 73
4.15 Curva degli errori in funzione della frequenza di clock al TX 77

Parte I

Introduzione

Capitolo 1

Introduzione

I sensori dei robot umanoidi possono beneficiare enormemente dei vantaggi offerti dalla codifica a eventi secondo cui l'informazione è rappresentata dal tempo trascorso tra eventi e dall'indirizzo della sorgente.
Utilizzare uno schema di campionamento classico dei sensori impatta su banda, consumi e risorse computazionali oltre che sulla latenza in quanto si utilizza una acquisizione dei dati sensoriali sincrona.
Tra tutti i componenti di un robot uno dei più importanti è la mano, infatti, secondo recenti studi, per l'uomo l'uso della mano ha consentito lo sviluppo dell'intelligenza grazie alla possibilità che essa offre di interazione con l'ambiente.
Attualmente i sensori visivi sono ben sviluppati e sfruttano appieno i vantaggi della codifica a eventi applicandola direttamente sul segnale analogico prodotto in uscita nativamente.
I sensori tattili nativi a eventi, invece, sono ancora in sviluppo ma è stato prodotto un sistema di codifica event-driven applicato ai campioni clock-based prodotti da sensori tattili capacitivi, il Tactile Hub, utilizzato su iCub. Questo elemento genera e trasmette gli eventi ad una board che acquisisce, deserializza e applica un timestamp ai dati, attraverso il protocollo seriale asincrono AS-AER. Questo protocollo è stato progettato in maniera asincrona per evitare di avere un filo dedicato per trasmettere il segnale clock e per ridurre al minimo il consumo di energia, diversamente da quanto adottato da altre soluzioni. Inoltre minimizza la latenza, non necessita di rapporti precisi tra il clock al trasmettitore e quello al ricevitore e non utilizza dei meccanismi complessi di ECC in quanto è robusto

per costruzione.
È necessario allestire una configurazione che comprende i due moduli hardware descritti per testare il protocollo di comunicazione AS-AER e l'intero funzionamento del sistema. In aggiunta la progettazione di un protocollo asincrono seriale a bassa complessità basato sulla codifica data-strobe è utile per scopi di debug sui componenti interni al Tactile Hub.

1.1 Sensori a eventi

Il principio di funzionamento dei sensori a eventi si basa su quello degli organi di senso umani, i quali acquisiscono un segnale, lo convertono in un impulso e lo inviano al sistema nervoso centrale.
I segnali generati da un un sensore a eventi sono chiamati spike e sono segnali asincroni che contengono informazione sulla localizzazione e sul tempo. L'uscita prodotta da sensori event-driven è generalmente non ridondante e sparsa come si può evincere prendendo in considerazione, ad esempio, il Dynamic Vision Sensor (DVS) [17], una camera event-driven, la quale produce eventi quando vi è un cambiamento di intensità luminosa che supera una certa soglia.
I sensori a eventi, a differenza di quelli convenzionali basati sul campionamento, fungono da driver dei dati generati. La codifica event-driven presenta tre grandi vantaggi:

- **auto-campionamento**: il tempo di integrazione è impostato dal sensore anzichè da una frequenza di campionamento costante;
- **non ridondanza**: gli eventi generati contengono solo informazione utile;
- **bassa latenza** : gli eventi sono generati e trasmessi direttamente dal sensore senza attendere un'istante di campionamento determinato nel tempo.

Le videocamere convenzionali catturano scene con un frame-rate fisso. Calibrare quindi il frame rate sulla base dell'oggetto della scena che si muove più velocemente è qualcosa di poco efficiente, perché in scenari tipici si hanno diversi oggetti, cose, persone che hanno velocità diverse.
Nelle camere a eventi DVS viene utilizzato un campionamento dinamico che si adatta alla natura del segnale osservato in maniera più efficiente rispetto alle camere tradizionali,

oltre a beneficiare dei vantaggi di un sistema a eventi.

Questo paradigma di campionamento dinamico è il Send-on-Delta[14], secondo il quale dei campioni vengono acquisiti solo se il segnale misurato è cambiato di un certo valore rispetto all'ultimo campione misurato.
Questo genera molti campioni nelle regioni dove il segnale campionato varia rapidamente e pochi campioni nell'altro caso e non permette una ricostruzione fedele del segnale campionato, ma offre delle buone prestazioni nel complesso. Analizzando il funzionamento di una camera a eventi, che utilizza questo modello, è possibile notare come non vengano prodotti valori di ampiezza con ciascun evento, ma grandezze di tipo temporale. Tuttavia è semplice ricavare informazioni di ampiezza da questi ultimi, quindi è lecito affermare che l'elaborazione dei dati può avvenire mediante i classici sistemi di processamento del segnale.
Il campionamento dinamico per generare eventi può essere realizzato in vari modi, ad esempio sfruttando degli ADC asincroni: l'output analogico di un sensore viene convertito attraverso ADC asincroni generando eventi e trasmettendoli nel più breve tempo possibile. ADC asincroni sigma-delta possono essere utilizzati a tale scopo ma, molto spesso, per le loro dimensioni si preferisce utilizzare altri sistemi di campionamento dinamico.

1.2 L'utilità della pelle artificiale nei robot

Principalmente gli attuali robot utilizzano sistemi artificiali di percezione visiva per interagire con l'ambiente circostante, ma ci sono dei compiti che risultano di difficile svolgimento se non impossibili senza dotare i robot del senso del tatto.
Utilizzare sensori visivi per ricavare informazioni tattili richiede elaborazioni complesse; le informazioni tattili forniscono al robot, ad esempio, dati relativi sulla posizione dell'oggetto nella mano, sulla deformazione, sull'attrito e sulla forza esercitata dalle dita [6].
Si stanno studiando degli algoritmi per il mantenimento di una presa stabile per oggetti fragili o scivolosi in condizioni sfavorevoli basandosi sul rilevamento dello slittamento e sul controllo della forza; per tale sviluppo sono necessari sensori tattili che forniscano letture accurate delle forze normali e tangenziali applicate su di essi.
Il tatto umano è il sistema di riferimento per la robotica in termini di sensibilità, frequenza, risoluzione e altri parametri. La pelle umana utilizza quattro tipi di recettori meccanici,

ciascuno dei quali con un tipo di risposta diversa e diverse proprietà fisiche; dimensione, forma e struttura forniscono risposte diverse alla pressione e alla vibrazione. È difficile creare un sensore che copra tutte le possibili risoluzioni spaziali, frequenze e proprietà di interazione, infatti, i sensori utilizzati per la pelle elettronica hanno proprietà diverse, sono realizzati con materiali differenti e sfruttano vari meccanismi di trasduzione.
Prendendo in considerazione lo stato attuale dello sviluppo dei sensori tattili è fondamentale sviluppare nuovi materiali e strutture che possano rendere la risposta dei sensori agli stimoli esterni più precisa, in quanto è necessario distinguere tocchi delicati con alta precisione per manipolazioni critiche [6].
La sensibilità richiesta dei sensori va da qualche Hz a qualche KHz per discriminare la struttura di un oggetto durante la manipolazione. Per quanto riguarda l'isteresi della pelle umana, il sistema sensoriale umano è in grado di compensarla in maniera egregia, tuttavia, in ambito robotico si preferisce utilizzare sensori con isteresi ridotta per evitare un complesso processamento del segnale.
Prendendo in considerazione la pelle umana, un robot completamente equipaggiato di pelle artificiale necessita di migliaia di elementi sensibili, e questo causa costi elevati in termini di consumo di energia e fili, inoltre i dispositivi sensibili dovrebbero essere in grado di ricoprire superfici non planari e di difficile accesso con diverse conformazioni e curvature. La maggior parte dei sensori descritti in letteratura hanno alcuni dei requisiti chiave elencati, ma molti di essi ha degli svantaggi che ostacolano il loro utilizzo in robotica perché spesso sono troppo fragili o rigidi per essere applicati su superfici curve; in altri casi, invece, essi richiedono una produzione complessa, che si traduce in costi elevati [6].
È possibile affermare, quindi, che il tatto umano è sì una linea guida per lo sviluppo di robot che possano svolgere compiti da umano, tuttavia non è strettamente necessario dotare un robot di un elevato numero di sensori su tutta la sua struttura se, ad esempio, si considerano lavori di tipo industriale o che necessitino di controllo da remoto.

1.3 Sensori tattili artificiali

I sensori tattili adottati in robotica possono essere di diverso tipo [6]:

- **capacitivi**

- **resistivi**
- **piezoelettrici**
- **ottici**
- **magnetici**

I sensori **capacitivi** [6] sono costituiti da due strati di materiale conduttivo, separati da un materiale dielettrico deformabile; applicando una pressione su di uno strato esterno si ha di conseguenza la deformazione del dielettrico che fa variare la capacità della struttura la quale viene tradotta in una stima di pressione.
La compatibilità di questi sensori con superfici curve e l'adozione di componenti elettronici standard facilmente reperibili rende questa tecnologia particolarmente adatta all'ambito robotico. Tali sensori sono altamente sensibili e compatti, ma sono caratterizzati purtroppo anche dalla degradazione del dielettrico causata dall'usura, dalla deriva della sensibilità per la temperatura e da isteresi.

Nei sensori **resistivi** [6] vi sono due elettrodi per misurare la variazione di resistenza dovuta a forze applicate al sensore. La loro struttura è relativamente semplice e possono essere implementati su circuiti stampati flessibili.
Per leggere il valore della resistenza del sensore sono richiesti diversi elementi: un divisore di tensione e un ADC, componenti semplici e compatti. I principali punti di forza sono il loro basso costo, basso rumore e buona sensibilità mentre gli svantaggi sono un elevato consumo energetico, l'isteresi e una breve durata dei materiali.

I sensori **ottici** [6] sfruttano un principio di funzionamento molto semplice; emettono luce a infrarossi e rilevano quando un ostacolo in prossimità interrompe il flusso luminoso, garantendo quindi la rilevazione di oggetti in avvicinamento oltre ai contatti. I punti a sfavore sono la diminuzione delle prestazioni in condizioni di luce intensa e l'elevato consumo energetico.

I materiali **piezoelettrici** [6] generano cariche in modo proporzionale alla forza applicata al sensore: la loro risposta è rapida e assume comportamento lineare per un ampia gamma di stimoli, rendendoli adatti al rilevamento di una forza.

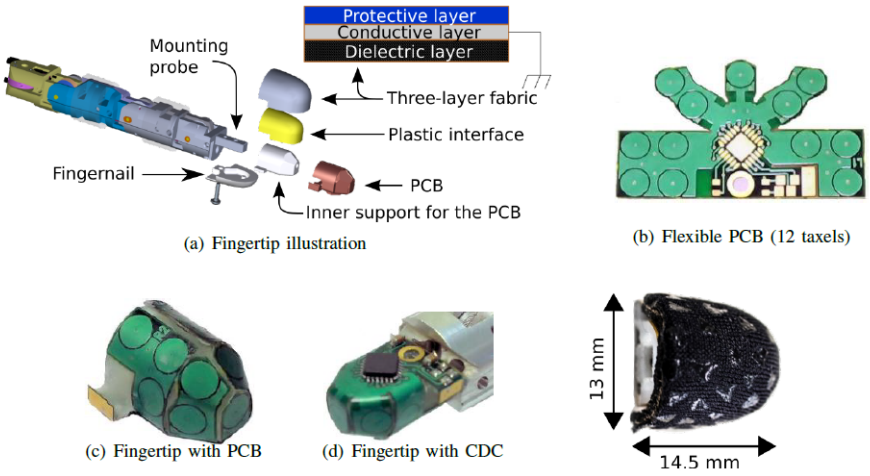
Questo tipo di materiali sono utilizzati per l'implementazione di sensori tattili **POSFET** (Piezoelectric Oxide Semiconductor Field Effect Transistor) [9], ovvero dei dispositivi integrati caratterizzati da materiale piezoelettrico deposto sul gate di un MOS complementare, per riuscire a rilevare cariche generate dalla forza applicata al sensore.
Il POSFET permette l'integrazione del circuito di lettura con quello di rilevamento ottenendo una riduzione al minimo del rumore e dei cablaggi, massimizzando la risoluzione. La produzione di questo sensore richiede però lo sviluppo di circuiti flessibili e di processi per la lavorazione di un polimero da utilizzare sopra l'elemento sensibile.

I sensori **magnetici** [6] infine incorporano dei magneti in uno strato di materiale deformabile; vengono misurate variazioni del campo magnetico causate dal movimento dei magneti per ottenere delle stime della forza applicata. Punto a sfavore di questa tecnologia è la rilevazione alterata del segnale in prossimità di oggetti metallici, per questo tale sensore ha usi limitati in robotica.

1.4 Design del polpastrello di iCub

I sensori tattili del dito di un robot umanoide devono garantire affidabilità e ripetibilità, avere bassa isteresi oltre ad essere robusti e con bassi costi di produzione. I polpastrelli artificiali implementati su iCub [16] sfruttano il principio della trasduzione capacitiva per misurare le forze applicate su di essi.
Tipicamente, in ambito robotico, il materiale dielettrico dei sensori capacitivi è composto da elastomero ricoperto da uno strato conduttivo; la composizione di questo, però, tende a limitare la durata del sensore a causa dell'invecchiamento, complica notevolmente il processo di produzione ed è affetto da isteresi.
Il polpastrello progettato per iCub [16], invece, utilizza sensori con una struttura a tre strati che comprendono uno strato di dielettrico deformabile, uno conduttivo e uno protettivo, realizzati con tecniche di produzione standard. Il risultato è affidabilità, robustezza e facilità di fabbricazione.

Come mostrato in Figura 1.1, la forma del dito riprende i tratti di uno umano. Il supporto interno è realizzato in plastica, il PCB flessibile è avvolto intorno al supporto interno e i 12 sensori sono distribuiti su piani intagliati. Il PCB ospita il chip che esegue la conversione da capacità a segnale digitale (CDC) mentre una superficie in plastica di 1mm



(a) Fingertip illustration (b) Flexible PCB (12 taxels)

(c) Fingertip with PCB (d) Fingertip with CDC

Figura 1.1: Polpastrello di iCub: a) disegno CAD del polpastrello, b) – e) mostrano diversi stadi della manifattura di un prototipo del polpastrello [16].

serve da interfaccia meccanica. Il guscio esterno del sensore è composto da tre strati: uno strato di neoprene deformabile, un materiale tessile conduttivo (lycra) connesso a massa e uno protettivo.

Nel polpastello di iCub[16] il PCB agisce come una delle due piastre conduttive di un condensatore, e lo strato conduttivo del tessuto a tre strati funge da seconda piastra conduttiva. Tra le due armature c'è poi uno strato di neoprene deformabile e uno in plastica, che funge da materiale dielettrico del condensatore.

La forza applicata deforma lo strato di neoprene cambiando la distanza tra le due armature del condensatore, con conseguente variazione del valore di capacità misurato. Il PCB incorpora 12 aree conduttive circolari che agiscono come piastre per 12 diversi condensatori, garantendo 12 punti di pressione distinti.

Esperimenti [16] hanno mostrato come ciascun sensore sul polpastrello possa differenziare forze a partire da 0.05N, mentre l'isteresi del sensore dipende dalla deformazione dello strato dielettrico del tessuto a tre strati, cioè il neoprene.

L'isteresi è presente solamente quando forze di almeno 1 N sono applicate, inoltre dipende anche dalla durata oltre che dall'entità della forza applicata: sono stati eseguiti altri esperimenti [16] per studiare l'effetto di una forza applicata per molto tempo sull'isteresi del polpastrello arrivando alla conclusione che l'isteresi è un fattore da prendere in considerazione e compensare.

È stato studiato anche il crosstalk: da esperimenti effettuati si evince che anche tra taxel vicini questo problema non è mai presente. Test effettuati per studiare la risoluzione spaziale hanno evidenziato come tutti i sensori di un taxel forniscano la medesima risposta.

1.5 Protocollo AER

Mahowald e Sivilotti proposero il protocollo Address Event Representation [8] per trasmettere impulsi (spike) da un array all'altro di neuroni su chip diversi.

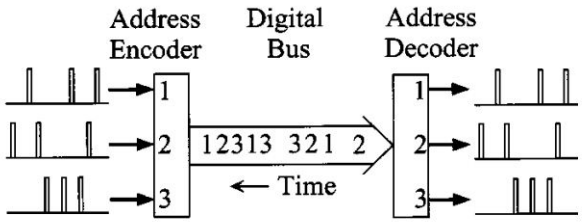


Figura 1.2: Protocollo AER [8].

Nel loro schema, rappresentato in figura 1.2, un address-encoder genera un unico indirizzo binario per ogni neurone ogni volta che tale neurone genera un impulso. Un bus trasmette questi indirizzi al chip ricevitore, dove un decoder di indirizzi seleziona la locazione corrispondente.
Gli impulsi AER sono trasmessi dai neuroni in modo seriale inviando indirizzi su un bus digitale. Il multiplexing è trasparente se il processo ciclico di codifica, trasmissione e decodifica è più breve di Δ/n secondi, dove Δ è la precisione della temporizzazione degli spike desiderata ed n è il massimo numero di neuroni che sono attivi durante questo tempo.
Otto anni dopo che i due scienziati proposero il protocollo AER [8], questo divenne il candidato numero uno per comunicazioni tra chip neuromorfici e le sue prestazioni originali furono molto migliorate.

1.6 Protocollo seriale asincrono AS-AER per sistemi a eventi

Il protocollo AER [8] è un protocollo di comunicazione a eventi utilizzato in sistemi neuromorfici: l'informazione non è contenuta nell'ampiezza ma in un identificatore, ovvero l'indirizzo della sorgente che ha generato l'evento, e nel tempo trascorso tra due eventi.
Le implementazioni più comuni di protocolli di comunicazione AER sono di tipo parallelo e utilizzano meccanismi di handshake, tuttavia applicazioni embedded beneficiano enormemente delle implementazioni seriali del protocollo poiché la crescente capacità dei chip neuromorfici richiede un numero crescente di bit per codificare l'indirizzo [20] [10] [7] [11].
Il protocollo AS-AER [19], rispetto al classico AER, garantisce una comunicazione ad alta velocità di tipo seriale utilizzando un numero ridotto di interconnessioni a discapito della necessità di utilizzare risorse hardware dedicate.

I requisiti fondamentali di un protocollo seriale AER sono:

- comunicazione full duplex;
- controllo esplicito del flusso per una riduzione dei collegamenti;
- clock esplicito in modo da semplificare notevolmente la progettazione del ricevitore;

Applicazioni robotiche richiedono inoltre:

- accoppiamento in AC;
- LVDS raccomandato.

Il protocollo AS-AER rispetta tutti i punti sopra elencati, inoltre è asincrono per utilizzare un numero ridotto di interconnessioni e avere un consumo esiguo in termini energetici. Utilizzare una comunicazione asincrona significa che il ricevitore non ha bisogno di alcun circuito per la ricostruzione del clock del trasmettitore.

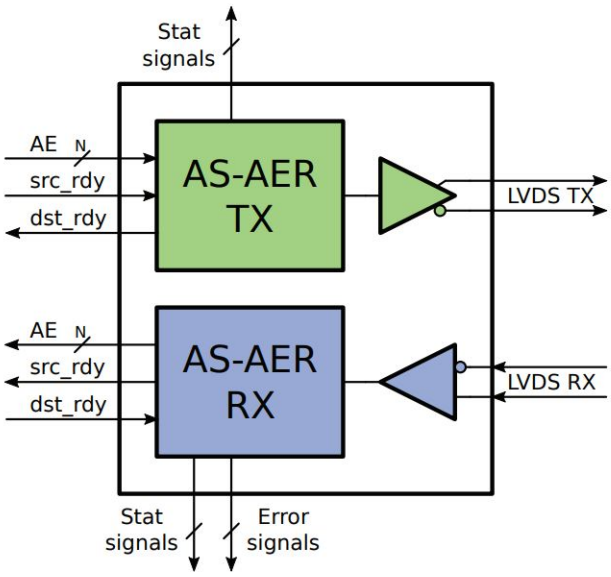


Figura 1.3: Diagramma a blocchi del transceiver AS-AER [19].

In Figura 1.3 è presente una rappresentazione ad alto livello del trasmettitore e ricevitore connessi a dei driver LVDS. Entrambi i moduli sono connessi ad altri sottosistemi attraverso interfaccia sincrona parallela AER (SPAER) e utilizzano segnali per fornire informazioni riguardo lo stato ed eventuali errori.
L'approccio alla base di questo protocollo [19] consiste nel distinguere e misurare, con una certa precisione, gli intervalli temporali tra eventi, inoltre, utilizzando intervalli discreti, la precisione richiesta non è alta. Gli eventi, lavorando con segnali digitali, possono essere i fronti di un segnale che varia.

La Figura 1.4 mostra la codifica di una word a 16 bit più un bit di parità ottenuta con diversi protocolli. Le transizioni possono esserci solamente da un bit all'altro o nel mezzo di un bit, avendo quindi un intervallo temporale tra eventi (IEI) come multiplo di $1/2T_b$, con T_b che rappresenta la durata di un bit.

La codifica Manchester è basata sulla transizione all'inizio di ciascun bit e una al centro di un bit quando un '1' viene trasmesso.
La codifica NRZM, invece, segnala un '1' con una transizione al centro del bit e ignora uno '0'. Inconveniente di questa codifica è che non è DC-free e richiede grande precisione

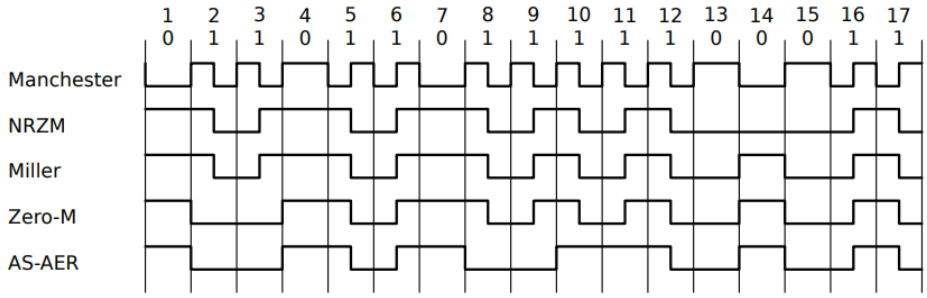


Figura 1.4: Esempi di codifica per mostrare le differenze tra il protocollo AS-AER e codifiche simili [19].

nelle misure temporali poiché teoricamente non c'è limite alla distanza massima tra due fronti.

La codifica di Miller [12], rispetto a quella NRZM, introduce una transizione tra due zeri consecutivi, ma non risolve il problema del bilanciamento della corrente continua.

Un miglioramento della codifica di Miller è la Zero-Modulation che si basa sull'osservazione che qualsiasi sequenza arbitraria di bit può essere divisa in sequenze alternative del tipo /01*0/ (una sequenza di '1', eventualmente nulla, racchiusa tra due '0') e del tipo /1+/, non comprendendo alcuno '0' e includendo almeno un '1'. Si hanno problemi con il bilanciamento della continua solo se ci sono sequenze del primo tipo che hanno un numero pari non nullo di '1', ovvero nella forma /0(11)+0/.
Questo è dove la codifica differisce da quella di Miller in quanto cambia la codifica di questi '1' piazzando delle transizioni ai confini di ciascuna coppia di '1'. Si risolvono così due problemi, il bilanciamento della continua e il limite della massima distanza tra due transizioni consecutive ($2T_b$). Problema di questa codifica è che essa richiede di conoscere in anticipo se una sequenza di bit del primo tipo ha un numero pari non nullo di '1', ma questo richiede una memoria infinita.

Il protocollo AS-AER [19] mostrato in Figura 1.4 migliora la codifica di Miller. Ogni sequenza del tipo /1+/, /00/ e /010/ è codificata come per la codifica di Miller, le sequenze del tipo /011+0/ sono codificate piazzando una transizione tra confini di ciascuna coppia di '1' , eccetto quando il numero di '1' è dispari: in questo caso la transizione appena

prima l'ultimo '1' è ritardata al centro di esso. Questa codifica risolve tutti i problemi di quella di Miller, ma allo stesso tempo richiede una memoria esigua di soli 2 bit.

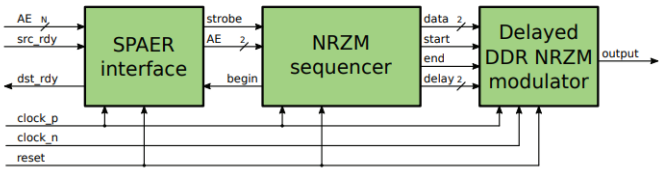


Figura 1.5: Diagramma a blocchi del trasmettitore AS-AER [19].

In Figura 1.5 è mostrata l'architettura del trasmettitore AS-AER [19]. Essa è composta da un'interfaccia SPAER e da una FSM che scansiona gli eventi AER in ingresso bit a bit e produce una sequenza di due bit codificati NRZM. Il modulatore NRZM pilota il segnale finale d'uscita utilizzando due clock complementari e registri DDR garantendo un bit rate uguale alla frequenza di clock richiesta per il funzionamento (100 MHz per 100 Mbps per l'implementazione discussa).

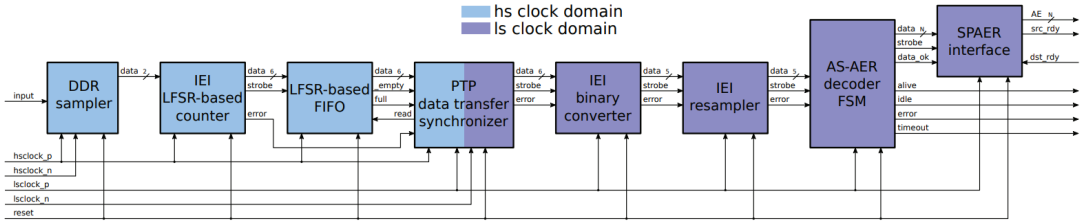


Figura 1.6: Diagramma a blocchi del ricevitore AS-AER [19].

La Figura 1.6 mostra l'architettura di riferimento del ricevitore [19]. Questo modulo può essere diviso in due parti, la prima che lavora ad alte frequenze per misurare con alta precisione gli intervalli temporali tra eventi del segnale in ingresso, e una seconda parte che lavora a più basse frequenze (la stessa del bit rate massimo, ovvero 100 MHz per 100 Mbps), per decodificare i dati ricevuti.

L'idea di base è misurare gli intervalli temporali tra eventi attraverso un contatore che lavora ad alta frequenza, resettato ad ogni fronte del segnale di input. Per avere due campioni per colpo di clock, il clock minimo richiesto del campionatore DDR deve essere tre

volte il valore del bit rate di trasferimento dei dati.
Un contatore basato su LFSR collegato ad una piccola FIFO è stato utilizato per far funzionare il protocollo su FPGA di fascia bassa. Il modulo di sincronizzazione e trasferimento dati controlla lo stato della FIFO, legge e passa i dati al sottosistema di decodifica che lavora con clock a bassa frequenza.
Nella parte di circuito che lavora a bassa frequenza, i dati generati dal contatore LFSR sono prima convertiti in binario e poi ricampionati, in modo tale da corrispondere a uno degli intervalli temporali tra eventi permessi. Questi dati sono utilizzati da una FSM che implementa un decoder AS-AER. La FSM attende per un nuovo IEI, aggiorna un registro temporaneo e un contatore di bit ricevuti e, quando un evento seriale AER è stato ricevuto interamente, viene controllata la parità e i dati vengono trasferiti all'interfaccia SPAER.

Sono stati effettuati dei test connettendo i due moduli descritti per determinare le prestazioni in funzione del rapporto tra il clock al TX e quello all'RX, ottenendo i risultati in Figura 1.7.

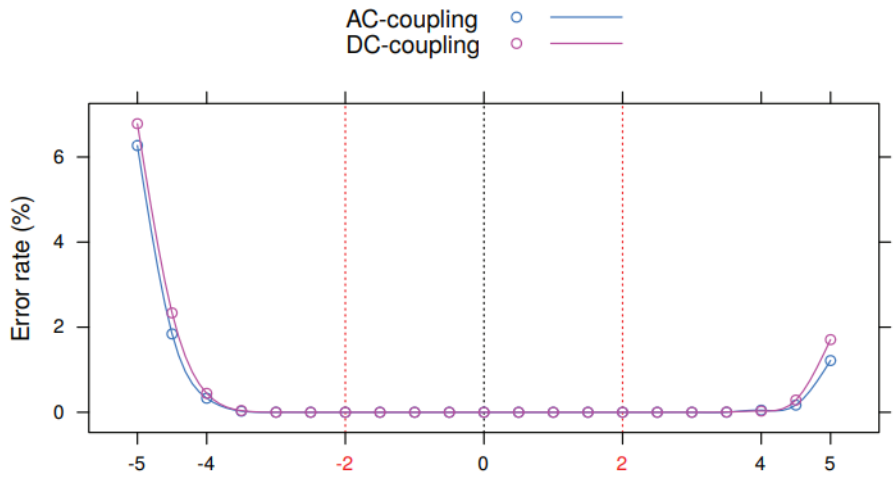


Figura 1.7: Curva d'errore in funzione della differenza relativa di clock al TX e all'RX, sia con accoppiamento AC che DC [19].

Considerando alternative di protocolli di comunicazione seriale AER, queste sono caratterizzate da event-rate più alti ma, considerando un robot umanoide completamente coperto di sensori tattili, un bit rate di 100 Mbps dovrebbe bastare [4], inoltre, tutte queste

alternative richiedono specifici componenti per l'implementazione [10] [7] [11] o sono completamente custom [20] rendendo difficoltosa e non pratica una coesistenza tra IC full custom e sistemi implementati su FPGA.

Rispetto ad altri protocolli di comunicazione, nei quali è molto efficiente includere più dati in un singolo pacchetto (come discusso in [3]), con il protocollo AS-AER [19] questo non è necessario e gli eventi sono spediti non appena sono generati.

Facendo una comparazione tra protocollo AS-AER [19] e Ethernet (non considerando i protocolli di alto livello), si nota come, per il protocollo Ethernet, l'efficienza della trasmissione dipenda dal numero di bit trasmessi per un certo payload; per ottenere il massimo dell'efficienza bisogna trasmettere un frame di 1500 byte con 1538 byte da trasmettere; considerando anche la codifica 8b/10b, l'efficienza decresce.
È possibile affermare che, con il protocollo Ethernet, l'efficienza aumenta con l'aumento della latenza e/o del jitter degli eventi spediti, perché il sistema colleziona più eventi prima di trasmetterli. Cercando di minimizzare la latenza e il jitter, è possibile ridurre la dimensione dei frame, ma l'efficienza scende drasticamente.

Per il protocollo AS-AER [19] c'è un overhead fisso di 2 bit (bit di start e di parità) per evento, senza overhead introdotti dalla codifica, ottenendo una efficienza costante del 94.1%. La latenza è costante e predicibile e il jitter è lo stesso del periodo di clock.
Mentre il protocollo Ethernet (e anche CAN) include meccanismi di ECC, il protocollo AS-AER non li include in quanto è basato su un certo numero di strategie che lo rendono robusto da progetto. Inoltre, con Ethernet molti eventi sono inseriti in un unico frame, quindi perdere un frame corrisponde a molti errori, mentre per l'AS-AER, inviando evento per evento, solo l'evento corrotto sarà scartato. Per questo motivo, l'AS-AER [19] è supportato solo da un controllo di parità implicito e da uno esplicito.

1.7 Sistema tattile a eventi di iCub

Gli elementi sensoriali tattili, comunemente chiamati taxel, possono essere largamente distribuiti su tutta la struttura di un robot [5]. Il loro numero può arrivare alle migliaia introducendo problemi di interconnessioni, throughput, latenza e risorse se un approccio di tipo clock-driven è impiegato. Tuttavia è stato dimostrato come l'attivazione dei taxel

sia scarsa sia nello spazio che nel tempo.
L'utilizzo della codifica a eventi applicata ai sistemi tattili dei robot è quindi una tecnologia cruciale per ottimizzare le risorse evitando la trasmissione di dati a partire da taxel che non rilevano alcun contatto e per ottenere una bassa latenza.
I sensori a eventi nativi effettuano la codifica event-driven direttamente sul segnale analogico producendo un output nella forma di evento AER. Sensori tattili nativi a eventi sono i dispositivi POSFET [9], che però necessitano di ulteriore sviluppo per essere integrati su piattaforme robotiche.
Dato che la tecnologia capacitiva è matura e viene integrata attualmente sui robot è stato realizzato un sistema di codifica a eventi applicato a sensori capacitivi clock-based, implementato su FPGA [16]: questo sistema converte i campioni acquisiti dai sensori in treni di impulsi (eventi) che vengono poi trasmessi a un'unità di acquisizione principale attraverso il protocollo AS-AER [19].

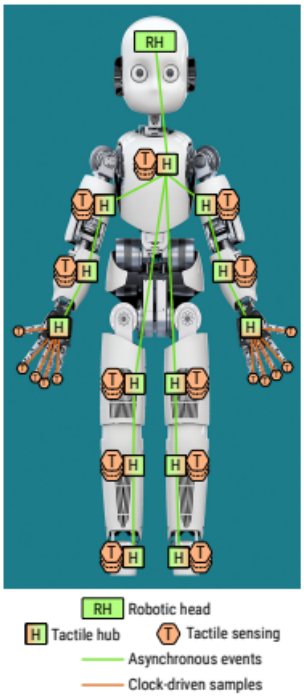


Figura 1.8: Infrastruttura sensoriale tattile a eventi sul robot iCub [5].

La Figura 1.8 [5] mostra l'infrastruttura del sistema tattile a eventi con Tactile Hub

multipli che acquisiscono i dati da uno specifico gruppo di taxel e, allo stesso tempo, propagano a monte i dati provenienti da altre regioni.

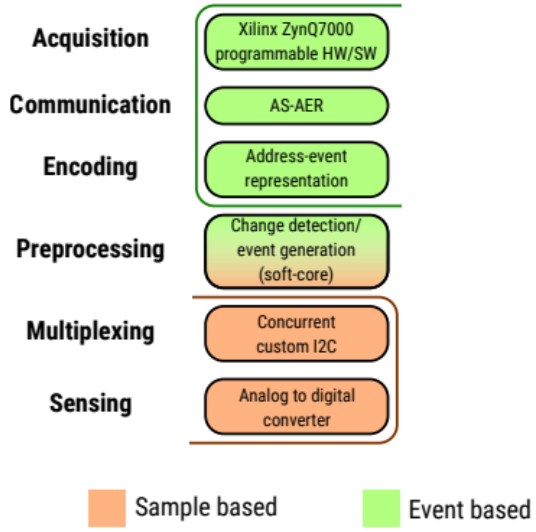


Figura 1.9: Integrazione della tecnologia tattile a eventi su iCub [5].

La Figura 1.9 [5] mostra il ruolo di ciascun componente e il suo dominio di lavoro (clock-driven o event-driven).

La Figura 1.10 [5] mostra un diagramma a blocchi dell'implementazione descritta: ogni polpastrello (o patch di pelle) possiede un certo numero di trasduttori capacitivi; il valore della capacità è acquisito e digitalizzato tramite un ADC, l'AD7147 e i campioni generati sono spediti via bus I2C al Tactile Hub su FPGA.
Il controller di acquisizione configura e acquisisce i dati sulla forza applicata ai taxel dei polpastrelli e delle patch di pelle.
Un modulo generatore di eventi converte i dati sincroni in un flusso di eventi AE (address event) utilizzando una pipeline a due stage in modo tale da gestire gli eventi alla stessa velocità del clock, con una latenza di 2 cicli di clock. Un modulo per l'arbitraggio degli eventi unisce il flusso di eventi di altre regioni con quello del generatore di eventi locale, arbitrando le collisioni. Gli eventi sono serializzati e trasmessi a una seconda board che

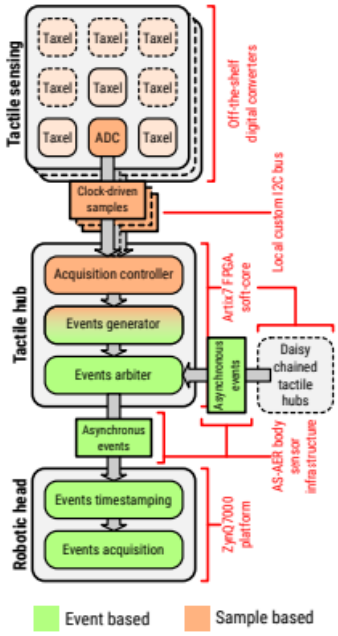


Figura 1.10: Diagramma a blocchi del sistema tattile a eventi descritto [5].

acquisisce deserializza e associa un timestamp agli eventi salvandoli in memoria.

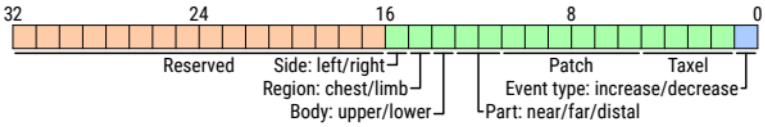


Figura 1.11: Spazio di indirizzamento per il protocollo AS-AER [5]. I sottospazi di indirizzamento sono allocati gerarchicamente e dipendono dal massimi numero di sottoelementi.

L'implementazione attuale del Tactile Hub lavora a 100 MHz.
Gli eventi sono codificati su 32 bit e identificano univocamente ciascun taxel sul corpo del robot, un bit serve per rappresentare il tipo di evento, ovvero l'incremento/decremento della forza applicata come mostrato in Figura 1.11 [5].
Questo tipo di codifica corrisponde al protocollo AER utilizzato in comunicazioni a eventi standard.

1.7.1 AD7147

L' AD7147 [2] è un circuito integrato che effettua la CDC (capacitance-to-digital conversion), dotato di sistemi per la compensazione ambientale, mostrato in Figura 1.12.
La circuiteria interna consiste in un convertitore Σ-Δ a 16 bit che è essenziale per la conversione di un input capacitivo in un valore digitale.
L'interfaccia dell' AD7147 dispone di 13 pin di input connessi a una matrice di switch per trasmettere i segnali in ingresso al convertitore.
Il risultato di ciascuna conversione CDC viene memorizzata in registri situati nel chip, che possono essere letti attraverso interfaccia seriale dall'utente. L'integrato dispone inoltre di memoria RAM utilizzata per la compensazione di agenti esterni quali umidità, temperatura e altri fattori ambientali che possono influenzare le operazioni dei sensori capacitivi: in maniera del tutto trasparente all'utente, l'AD7147 effettua continue calibrazioni per compensare questi effetti, permettendo di generare costantemente risultanti non affetti da errori.

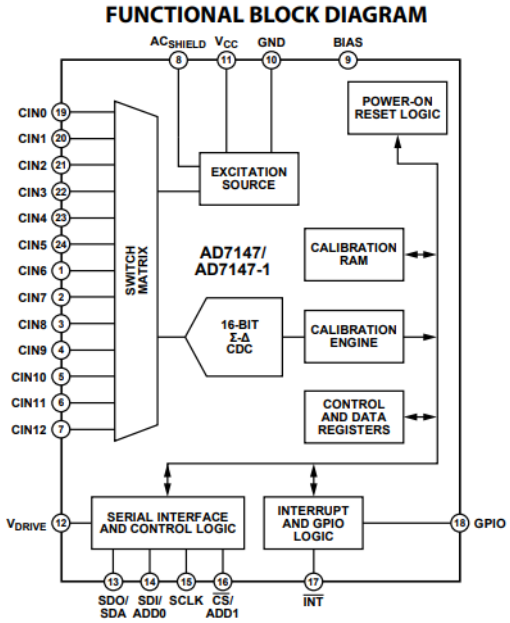


Figura 1.12: Diagramma a blocchi funzionale dell' AD7147 [2].

L'AD7147 misura variazioni di capacità da sensori a singolo elettrodo; l'elettrodo del sensore è situato sul PCB e rappresenta una delle due piastre un condensatore "virtuale",

l'altra piastra è il dito dell'utente, che è a massa rispetto all'ingresso del sensore, come è possibile osservare in Figura 1.13. L' AD7147 emette un segnale per caricare la piastra sul PCB e, quando l'utente avvicina il dito al sensore, viene creato un condensatore virtuale con l'utente che agisce appunto da seconda piastra.

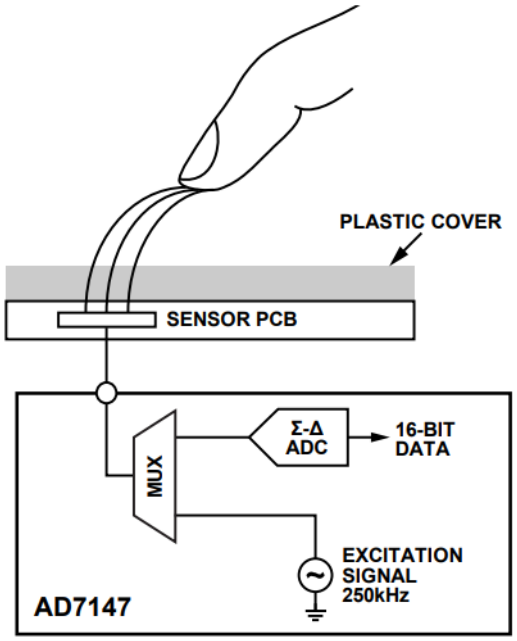


Figura 1.13: Principio di funzionamento di rilevazione tattile dell'AD7147 [2].

Andando più nel dettaglio, l'acquisizione di segnali capacitivi avviene nel seguente modo: un segnale ad onda quadra viene applicato su un ingresso CINx, il modulatore campiona continuamente la carica che passa attraverso CINx, l'uscita del modulatore viene processata attraverso un filtro digitale e i dati risultanti sono salvati nel registro CDC_RESULT_Sx per ogni stage di conversione.
Quando l'utente avvicina il dito al sensore la capacità totale associata cambia e viene misurata e, se supera un certo livello, l'AD7147 interpreta ciò come l'attivazione del sensore. La Figura 1.14 mostra la variazione del valore contenuto in CDC_RESULT_Sx quando un sensore viene stimolato: questo viene ritenuto attivo solo quando il valore del registro supera il valore della soglia alta oppure va al di sotto della soglia bassa.

In Figura 1.14 viene evidenziata l'attivazione di due sensori: il sensore A è connesso all'input positivo del convertitore, mentre quello B all'input negativo.

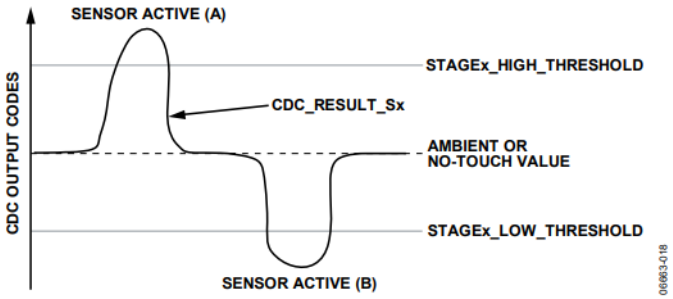


Figure 20. Sensor Activation Thresholds

Figura 1.14: Variazione del valore contenuto nel registro CDC_RESULT_Sx e soglie di attivazione del sensore [2].

L'AD7147 [2] effettua un processo di decimazione in cui vengono mediati un certo numero di campioni per produrre l'uscita. Questa tecnica riduce la quantità di rumore presente nel valore finale della conversione CDC, tuttavia, più è alto il tasso di decimazione più bassa è la frequenza di generazione delle uscite. Bisogna quindi cercare un compromesso tra la quantità di rumore nel segnale e la velocità di campionamento.

Con riferimento a [13], per effettuare la configurazione all'avvio dell'AD7147 bisogna programmare diversi registri suddivisi in tre banchi:

- banco 1: contiene i registri di **CONTROL**, **ENABLE** e **DATA**;
- banco 2: contiene i registri di **CONFIGURAZIONE** per ogni ingresso;
- banco 3: contiene una copia dei registri di **DATA** che contengono il risultato della capacitance-to-digital conversion (CDC), il valore della soglia e altri parametri numerici.

Come prima cosa vengono configurati i registri del banco 2, poi quelli del banco 1, poi è possibile leggere i dati attraverso interfaccia I2C. Il banco 3 non viene configurato. È possibile vedere la configurazione di tutti i registri in Tabella 1 di [13].

I dati dell'AD7147 vengono acquisiti dalla MTB che esegue una serie di manipolazioni sui dati. La Tabella 1.1 mostra le variabili e le costanti utilizzate nel codice dell'MTB.

Nome del parametro	Dimensione
No_load	12 x 16 bit array
Current_reading	12 x 16 bit array
GAIN	12 x 8 bit array
NOLOAD	8 bit constant
SHIFT	8 bit constant
MAXVAL	8 bit constant
MINVAL	8 bit constant
UP_LIMIT	16 bit constant
BOT_LIMIT	16 bit constant

Tabella 1.1: Variabili e costanti usate dall'MTB

Con il CDC configurato come in Tabella 1 di [13] i dati possono essere letti dalla EDMTB.
L'AD7147 effettua una decimazione e scrive il risultato nel corrispondente registro dello stadio di input e i dati sono letti ogni 11ms. Poichè il periodo di conversione del CDC è di 36 ms, il suo output è sovracampionato di un fattore 3 circa.
L'MTB utilizza invece 43.2 ms di periodo di campionamento, perdendo alcuni dati ogni 5-6 letture. È possibile utilizzare, in alternativa, un meccanismo di interrupt per il quale ogni interrupt indica la fine della conversione di 12 stadi di input e, di conseguenza, la disponibilità di 12 nuovi dati per la lettura dal CDC. Una volta che un interrupt si verifica, il registro STAGE_COMPLETE_INT_STATUS con indirizzo 0x00A viene letto per disabilitare l'interrupt, quindi i risultati di conversione possono essere letti in maniera sequenziale via I2C come mostrato in Figura 1.15.

La prima lettura effettuata è quella di "no load", ovvero una lettura dal CDC senza alcuna pressione applicata sul sensore. Se, per un determinato taxel, una lettura fornisce tutti zeri e la prima lettura (quella di "no load") era diversa da zero, un errore di NAK viene generato. Questo viene interpretato come errore perché, a un certo punto dell'acquisizione, i sensori producono come valore zero, non lavorando correttamente, quindi, quando questo accade più volte, una reinizializzazione della skin è eseguita. Altrimenti, la manipolazione dei dati procede con la compensazione della temperatura (che può anche essere disabilitata) nell'MTB, applicata al pad numero 6. La deriva della temperatura viene calcolata come mostrato in Figura 1.16.

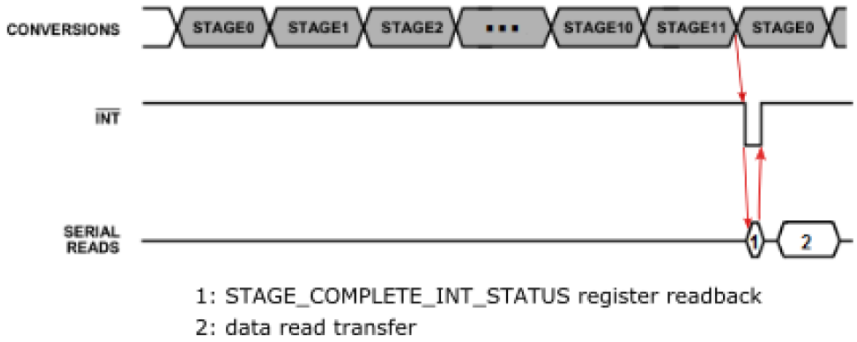


Figura 1.15: Diagramma temporale raffigurante l'interrupt di completa conversione [13].

No_load[6] < Current_reading[6] →

drift = (((Current_reading[6] – No_load[6])>>2)*GAIN[i])>>5;

No_load[6] ≥ Current_reading[6] →

drift = – (((No_load[6] – Current_reading[6])>>2)*GAIN[i])>>5

Figura 1.16: Calcolo della deriva della temperatura[13].

GAIN[i] sono costanti predefinite, derivate dalla caratterizzazione del taxel.

Il valore della lettura è calcolato come:

```
value[i] = Current_reading[i] – No_load[i] – drift
```

Possono verificarsi diverse situazioni:

1. value[i] ≤ – UP_LIMIT → txdata[i] = MAXVAL;
   a. moreover, if value[i] ≤ – (UP_LIMIT << 1) → send out-of-range error;
2. value[i] ≥ BOT_LIMIT → txdata[i] = MINVAL;
   a. moreover, if value[i] ≥ BOT_LIMIT << 1 → send out-of-range error;
3. – UP_LIMIT ≤ value[i] ≤ BOT_LIMIT → txdata[i] = NOLOAD – (value[i] >> SHIFT).

Figura 1.17: [13].

La relazione tra value[i] e txdata [i] è rappresentata in Figura 1.18.

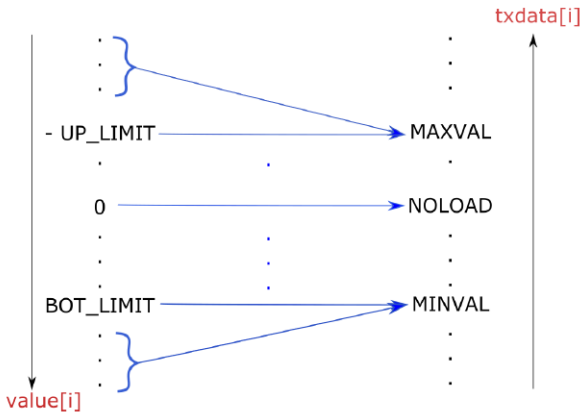


Figura 1.18: Corrispondenza valore - dato trasmesso [13].

La fase finale è quella di selezionare gli 8 bit meno significativi per ogni txdata[i] e usarli come campioni. Se tutti i taxel di una triangolo(insieme di taxel) forniscono una lettura di 0xFFFF, il triangolo è considerato come non connesso e i suoi campioni corrispondenti cono impostati per convenzione su NOLOAD.

Se 5 errori di NAK o out-of-range si verificano sul totale delle letture effettuate (considerando il numero totale di triangoli), una ricalibrazione del sistema viene avviata. Questo processo consiste nello stoppare la conversione corrente e nel riconfigurare tutti i registri secondo Tabella1 [13].

1.7.2 Codifiche a eventi per sensori tattili clock-based

Il metodo che il modulo di generazione di eventi utilizza per generare gli eventi dai dati clock-based è basato sull'attraversamento di soglia [5]: quando il valore del campione acquisito è più grande (o più piccolo) del precedente campione che ha generato un evento, di una certa soglia (Δ), un nuovo evento è generato e spedito. Se il sensore non viene stimolato, il modulo di generazione degli eventi non produce nulla.

Nel sistema considerato i sensori utilizzati non sono a eventi, ma sono sensori standard la quale uscita è discretizzata a frequenza fissa limitata dall'AD7147, settata a 50 Hz. Quando un campione viene acquisito e c'è un cambiamento maggiore di Δ, un evento viene prodotto in uscita, con la stessa frequenza di campionamento del sensore o minore.

Se il segnale cambia più di Δ, la dinamica del segnale di input è più veloce rispetto alla frequenza di campionamento e non è possibile generare eventi con adeguata frequenza. In questo caso, il tempo tra gli eventi non porta informazione sufficiente e il segnale originale non può essere ricostruito fedelmente.
Vista la limitazione del campionamento clock based dei valori dei sensori, vengono descritti due metodi [5] per implementare una lettura senza perdite: "Send Sampled Value on Delta" e "Interpolation and Asynchronous Event-Driven Time Encoding".

Send Sampled Value on Delta

Utilizzando il primo metodo, ogni qualvolta un evento viene generato, si invia il valore campionato dell'acquisizione corrispondente insieme all'indirizzo del taxel attivo.
L'indirizzo del protocollo AS-AER è composto da 32 bit, di cui 26 sono riservati per l'indirizzo del taxel, ma per iCub ne sono richiesti 16 (vedi Figura 1.4), quindi ci sono 10 bit disponibili per codificare i valori dei campioni della forza esercitata sull'elemento sensoriale.
Utilizzando il metodo Send Sampled Value on Delta, per valori di Δ abbastanza piccoli e in assenza di rumore, si ottiene una compressione senza perdite. Tuttavia un taxel non stimolato produce ancora un segnale abbastanza grande, quindi bisogna cercare un compromesso per Δ tra la reiezione al rumore in condizioni di inattività e la sensibilità di generazione degli eventi. Per ovviare a ciò può essere utilizzata una soglia per la reiezione del rumore statico, θ, quindi, solo se $f(t_i) > \theta$ e se $|f(t_i) - f(t_{i-1})| > \Delta$, un evento viene generato.

Interpolation and Asynchronous Event-Driven Time Encoding

Utilizzando il metodo "Interpolation and Asynchronous Event-Driven Time Encoding", prima di avviare l'algoritmo del generatore di eventi, due campioni consecutivi del segnale vengono interpolati linearmente con l'obiettivo di garantire una generazione più tempestiva degli eventi, come se si utilizzassero sensori a eventi.
Se la frequenza di ricampionamento è adeguata, il generatore di eventi riceve in ingresso l'equivalente di un segnale continuo; ogni volta che questo cambia di Δ, un evento è generato.

Utilizzando questo paradigma il tempo tra eventi non è più limitato dalla frequenza di campionamento del sensore ma dalla frequenza di ricampionamento. Ciascun evento è generato appena il cambiamento corrisponde a Δ e il tempo tra due spike racchiude le informazioni per ricostruire il segnale.

L'interpolazione può essere implementata direttamente sul Tactile Hub.
Ciascun AD7147 gestisce 12 taxel e ogni Tactile Hub può essere connesso con un massimo di 16 polpastrelli/patch di pelle; per una frequenza di resampling di 5 MHz, ciascun Tactile Hub deve gestire internamente circa 1 Gsamples/s di dati complessivi.

Dati i vincoli della FPGA, l'implementazione deve fare affidamento su interpolatori di campioni paralleli; un design ben bilanciato potrebbe comprendere, viaggiando a 250 MHz, uno scheduler e 4 generatori di campioni, collegati a 4 generatori di eventi indipendenti e poi tutti i flussi di eventi uniti insieme, dato che la loro velocità di generazione è molto minore dei dati ricampionati.
L'implementazione basata su interpolazione emula meglio il comportamento di un sensore a eventi, in cui la generazione degli eventi è direttamente applicata all'output analogico continuo del trasduttore.

Parte II

Implementazione e test del setup allestito

Capitolo 2

Implementazione e descrizione dei moduli utilizzati

2.1 Tactile Hub

Il Tactile Hub è il modulo hardware che ha il compito di acquisire i dati clock-sampled provenienti dai taxel e generare e trasmettere eventi sul bus seriale AS-AER. Un'altra funzione di questa unità è quella di unire il flusso di eventi generato dai sensori direttamente connessi con il flusso di eventi proveniente da altri sensori.

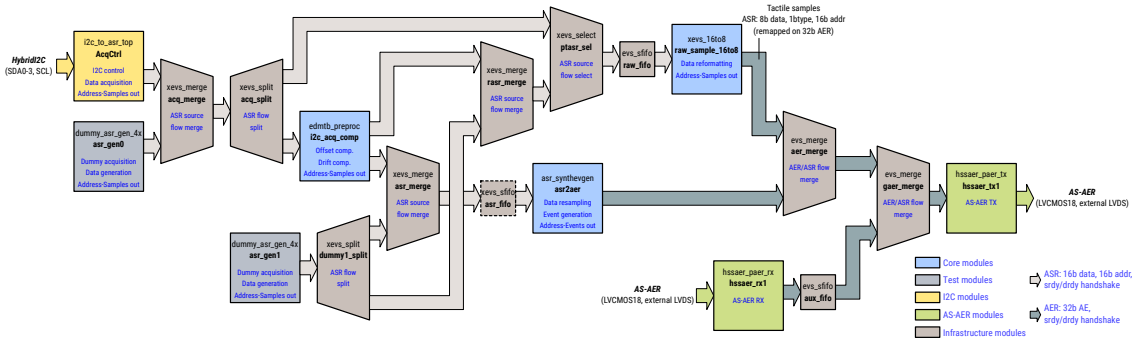


Figura 2.1: Schema ad alto livello del Tactile Hub [18]. In grassetto è scritto il nome dei moduli, in basso a destra è presente una legenda.

In Figura 2.1 abbiamo una rappresentazione ad alto livello del Tactile Hub in cui sono stati utilizzati colori diversi per rappresentare la tipologia di ogni sottomodulo.

Il sottomodulo in giallo, **AcqCtrl**, comunica direttamente con i taxel attraverso una interfaccia ibrida I2C: tale interfaccia è così definita in quanto si utilizzano 4 linee SDA e una sola linea SCL.
Lo standard I2C prevede l'adozione di una linea SCL per ogni linea SDA ma, in questo progetto, per risparmiare fili, la linea di clock è comune a tutti e quattro i fili di dato; questa modifica genera complicazioni a livello di protocollo, quindi è stato utilizzato un apposito controllore.
Il modulo in verde posizionato sull'estrema destra, **hssaer_tx1**, trasmette gli eventi attraverso il protocollo seriale asincrono AS-AER [19] in modalità LVDS direttamente alla testa del robot in cui è presente una HPU responsabile del processamento dei dati.
Dalla HPU verso il Tactile Hub, invece, la comunicazione utilizza il protocollo I2C.
L'altro modulo in verde, **hssaer_rx1**, riceve gli eventi provenienti da altri Tactile Hub.
A ciascun Tactile Hub posso connettere fino a 192 taxel, perchè si hanno a disposizione 4 linee SDA: su ogni linea SDA si possono connettere fino a quattro AD7147, che a loro volta possono gestire fino a 12 taxel. Ciascun AD7147 viene interrogato ogni 11 ms ma, considerando il fatto che ciascun taxel viene ricampionato a 5 MHz, si ottiene un flusso dati di circa 1 Gs/s: utilizzando solamente una pipeline non si riesce a processare questa mole di dati; per ovviare a questo problema sono state utilizzate quattro catene di ricampionamento e generazione degli eventi, ciascuna collegata su una linea SDA. In questo modo è possibile elaborare i dati utilizzando linee con frequenza di clock a 200 MHz per ciascuna catena di ricampionamento.
Tutto il resto del sistema, però, lavora utilizzando una frequenza di 100 MHz, quindi sono state utilizzate delle FIFO asincrone per passare dal dominio dei 100 a quello dei 200 MHz.
Si potrebbe pensare di utilizzare otto catene di ricampionamento al posto di quattro per ottenere una struttura che lavori completamente a 100 MHz.
I moduli in grigio scuro **asr_gen0** e **asr_gen1** sono invece dei dummy generator, ovvero dei generatori interni al Tactile Hub introdotti per scopo di debug in quanto in tal modo è possibile testare delle funzionalità specifiche anche in assenza di sensori connessi attraverso interfaccia ibrida I2C.
I dummy generator possono generare un segnale a dente di sega campionato (quindi a gradini) per il quale è possibile specificare il periodo tra un campione e il successivo utilizzando valori multipli di 10 ns. Il valore di default di questo periodo è 1.100.000, ovvero

11 ms, che corrisponde al tempo di acquisizione dei campioni attraverso il modulo **AcqCtrl**.
I sottomoduli in grigio servono per unificare o dividere i flussi di dati, quelli in azzurro effettuano delle elaborazioni.
Il modulo di **aer_merge** permette l'acquisizione dei campioni, degli eventi o di entrambi, **gaer_merge** permette di selezionare gli eventi provenienti da un eventuale secondo Tactile Hub connesso.
asr_fifo consente di passare dal domino di clock a 100 a quello a 200 MHz, perchè il blocco a valle, **asr2aer** è incaricato di effettuare il resampling e la generazione degli eventi.
Nello schema 2.1 non è presente una seconda parte del Tactile Hub incaricata di gestire l'accelerometro, presente però nel codice HDL utilzzato.
Adoperando **asr_gen0** è possibile iniettare dati a monte della parte di pre-processing, attraverso **asr_gen1** si possono generare dati che è possibile indirizzare in modo diretto al modulo che si occupa del resampling e della generazione degli eventi: è possibile bypassare in questo modo la parte di pre-processing.

2.1.1 Protocollo I2C

Questo protocollo permette di collegare, sullo stesso bus, un numero elevato di periferiche, ognuna delle quali caratterizzata da un indirizzo.
Il bus utilizza due linee seriali, SDA e SCL, più una linea per la massa; la linea SDA serve per il passaggio dei dati, mentre SCL viene utilizzato per trasmettere il segnale di clock, basilare per sincronizzare una trasmissione.
La peculiarità del bus I2C è quella di consentire la connessione di più periferiche, ma la comunicazione può avvenire solamente tra due dispositivi per volta.
Per ogni comunicazione instaurata abbiamo un master e uno slave: il master è il dispositivo che avvia e termina una comunicazione, lo slave, a discrezione del master, può ricevere o trasmettere. È possibile avere più master su di uno stesso bus, ma solo uno per volta può ricoprire questo ruolo.
Le linee SDA e SCL devono essere implementate tramite uscite open drain oppure open collector, quindi è necessaria una resistenza di pull-up tra ogni linea e l'alimentazione; questo implica che quando le due linee non sono utilizzate sono a livello alto.

2.2 AS-AER2UART

Il modulo AS-AER2UART viene utilizzato come unità connessa al trasmettitore del Tactile Hub attraverso il protocollo seriale asincrono a eventi AS-AER [19]. In Figura 2.2 è presente una rappresentazione ad alto livello del progetto.

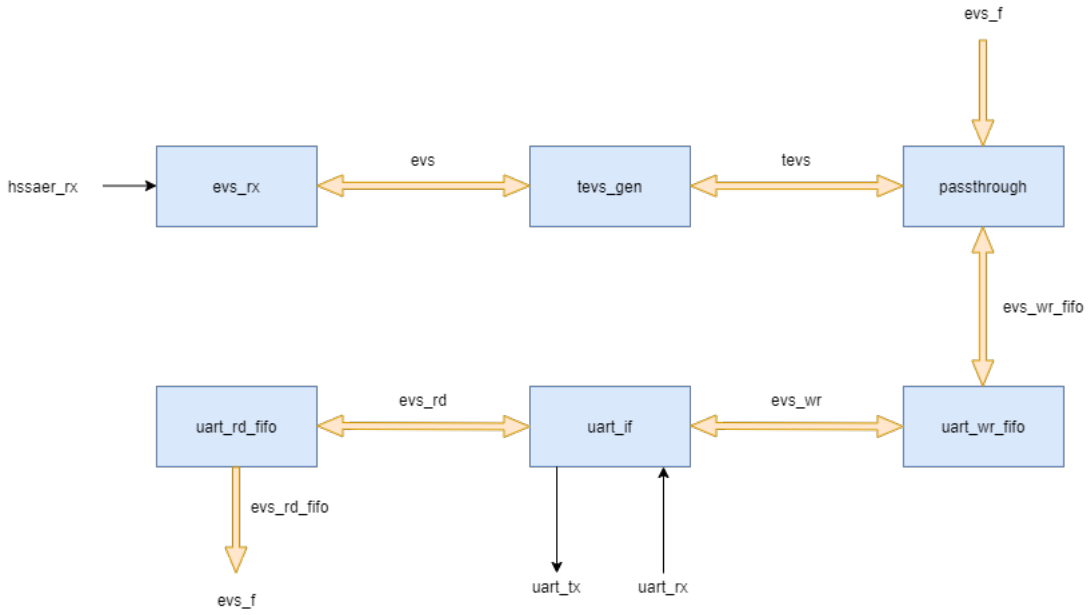


Figura 2.2: Rappresentazione ad alto livello del modulo AS-AER2UART.

Il progetto è caratterizzato da sei sottomoduli, ognuno dei quali svolge un compito specifico.
L'istanza **evs_rx** è il ricevitore AS-AER, preposto alla ricezione degli eventi trasmessi dal Tactile Hub, **tevs_gen** è incaricato di generare e applicare un timestamp agli eventi ricevuti mentre **passthrough** è un blocchetto che abilita il passaggio dei dati tra il blocco tevs_gen e i successivi.
Le istanze **uart_wr_fifo** e **uart_rd_fifo** sono due FIFO che confluiscono nel modulo **uart_if** che è un trasmettitore/ricevitore UART.
Tutto il sistema lavora con un clock di 100 MHz, eccetto il ricevitore che utilizza due clock a 100 MHz, rispettivamente sfasati tra loro di 180°, e due clock a 300 MHz, anch'essi sfasati tra lori di 180°.
La velocità di trasmissione della UART utilizzata è di 12Mb/s, velocità non standard, per consentire la trasmissione dei dati al PC senza generare overflow della FIFO.

2.3 Protocollo di handshake

Tra tutti i moduli dei due sistemi presi in esame per questa tesi viene utilizzato un protocollo di handshake utile per la sincronizzazione che utilizza due segnali: `srdy` e `drdy`, chiamati rispettivamente source ready e destination ready.
Ogni modulo possiede un segnale di destination ready, per comunicare, lato input, che il modulo è disponibile a ricevere dei dati, e un segnale di source ready, lato output, per comunicare al modulo a valle che il modulo è disponibile alla trasmissione di un nuovo dato. Quando entrambi questi segnali, scambiati tra due moduli, sono asseriti può avvenire la trasmissione del dato da sorgente a destinazione come descritto in Tabella 2.1.

In Figura 2.3 sono mostrati due moduli che si interfacciano utilizzando il protocollo descritto.

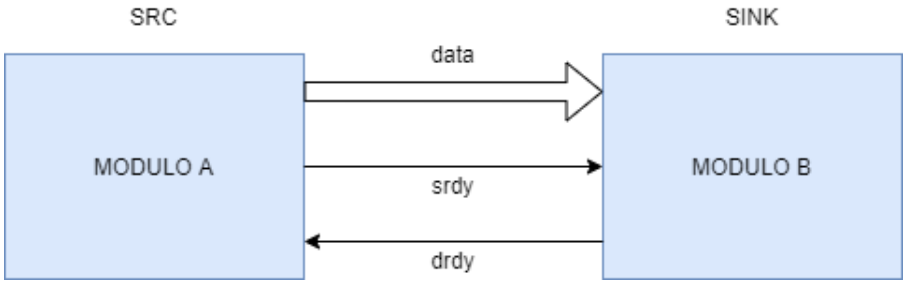


Figura 2.3: Protocollo di handshake tra due moduli, rispettivamente sorgente e destinazione

SRDY	DRDY	SIGNIFICATO
0	0	SRC e SINK non pronti
0	1	SRC non pronto, SINK pronto
1	0	SRC pronto, SINK non pronto
1	1	SRC e SINK pronti

Tabella 2.1: Combinazioni dei segnali di handshake e relativo significato

2.4 Hardware utilizzato

La strumentazione hardware utilizzata per tutto il lavoro di tesi comprende una board TE0712 connessa a una carrier board TE0701-05 e ad una debug card XM105, rispettivamente raffigurate in 2.4,2.5 ,2.6 e un modulo FTDI 2232H mini module rappresentato in 2.7.
Sul modulo TE0712 è presente una FPGA Artix-7 (XC7A200T), mentre la carrier board e la debug card forniscono diverse interfacce e pin di I/O per connettere la FPGA a dispositivi esterni. Il modulo FTDI è necessario per leggere/scrivere i registri del Tactile Hub tramite protocollo I2C.

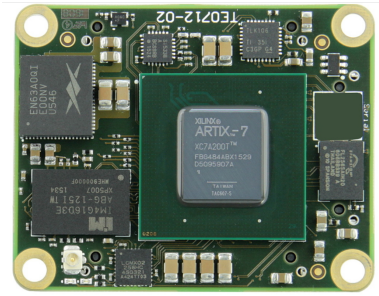


Figura 2.4: Modulo TE0712 della Trenz

Figura 2.5: Modulo TE0701-05 della Trenz

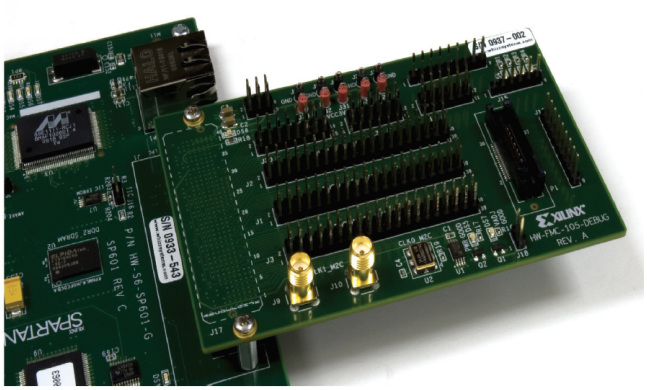


Figura 2.6: Xilinx debug card FMC XM105

Figura 2.7: Modulo FTDI 2232H mini module

2.5 Implementazione del setup sperimentale

È stato generato un unico design che comprende sia il modulo del Tactile Hub sia AS-AER2UART, il quale utilizza due MMCM per la generazione dei clock di entrambi i moduli. In particolare la parte di generazione dei clock dei due moduli è stata unificata eliminando i segnali doppi, come rappresentato in Figura 2.8.

Dopo aver testato il modulo AS-AER2UART singolarmente attraverso un generatore di dati interno è stato allestito il setup rappresentato in Figura 2.9, obiettivo di questo lavoro di tesi, necessario anche per testare il Tactile Hub, il protocollo di comunicazione e i dati trasmessi al PC attraverso UART.

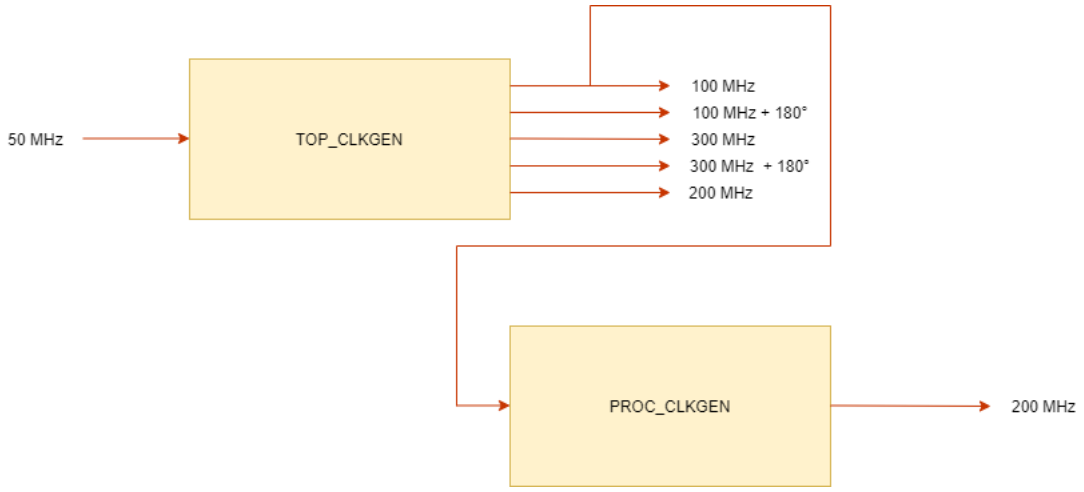


Figura 2.8: MMCM globale per i moduli Tactile Hub e AS-AER2UART.

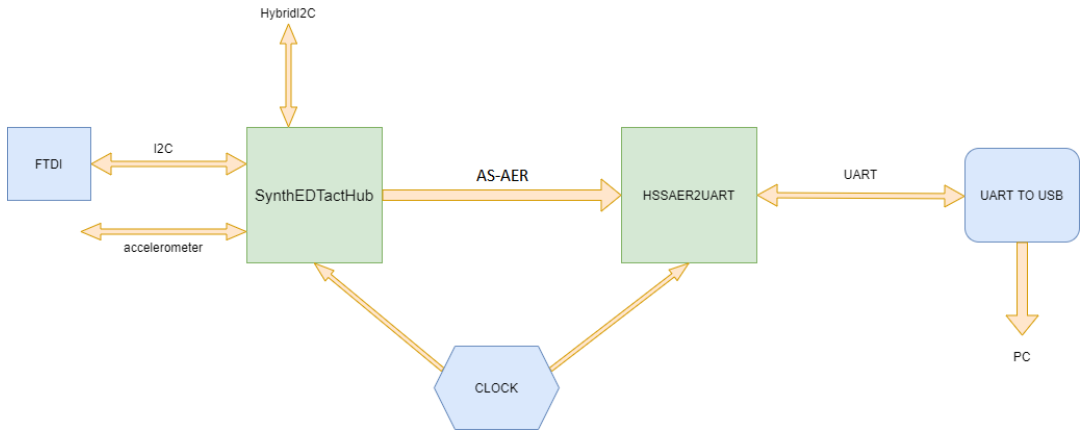


Figura 2.9: Schema del setup sperimentale allestito che comprende Tactile Hub, AS-AER2UART, modulo FTDI, trasmettitore UART e PC.

Obiettivo finale del lavoro di tesi consiste nell'allestire il setup funzionante di Figura 2.9 utilizzando al posto di sensori tattili uno stimolo prodotto da uno o più generatori interni al Tactile Hub; per abilitare e impostare questi generatori è stato utilizzato un modulo FTDI che, attraverso il protocollo I2C scrive e legge i registri del Tactile Hub.
Possono essere selezionati diversi percorsi per i dati generati all'interno del Tactile Hub programmando opportuni registri e, per questo lavoro di tesi si è testato il percorso che trasmette in uscita i campioni generati senza alcuna elaborazione.

È anche possibile programmare il modulo in modo da effettuare del pre-processamento sui campioni e per generare gli eventi.
I campioni in uscita dal Tactile Hub sono poi trasmessi al modulo AS-AER2UART che applica un timestamp e spedisce, tramite UART, i campioni comprensivi di timestamp al PC.
Dopo i test effettuati e descritti nel capitolo 3 si è riscontrata una incompatibilità del protocollo di comunicazione AS-AER con il setup allestito; per questo motivo il trasmettitore e ricevitore AS-AER sono stati sostituiti con quelli progettati basati sulla codifica data-strobe, descritti nel dettaglio in 4. L'implementazione del trasmettitore sul Tactile Hub lavora a una frequenza di 100 MHz per garantire i 100 Mbps di velocità di trasferimento mentre quella del ricevitore lavora a 200 MHz per poter campionare più di una volta i bit di dato.

2.5.1 Registri del Tactile Hub

	7	6	5	4	3	2	1	0	Default value
0x00	DUMMYGEN_SEL			DUMMY_ACQ_EN	FORCE_CALIB		I2C_ACQ_SEL	I2C_ACQ_EN	0x81
0x01	DUMMYGEN_EN7	DUMMYGEN_EN6	DUMMYGEN_EN5	DUMMYGEN_EN4	DUMMYGEN_EN3	DUMMYGEN_EN2	DUMMYGEN_EN1	DUMMYGEN_EN0	0X10
0x02	ASR_FILTER_TYPE	ASR_FILTER_EN			EVGEN_NTHR_EN	PREPROC_SAMPLES	PREPROC_EVGEN	DRIFT_COMP_EN	0X0e
0x03					SAMPLES_SEL	AUX_TX_EN	SAMPLES_TX_EN	EVENTS_TX_EN	0xfb
0x04	EVGEN_WR_SEL			NEURAL_EVGEN_ENABLE_MASK			EVGEN_SELECT		0x00
0x05	ACCEL_MERGE_DATA						ACCEL_FIFO_W_ENABLE	ACCEL_OUTNOR_ENABLE	0X03
0x08	Dummy generator period [7:0]								1_100_000
0x09	Dummy generator period [15:8]								
0x0a	Dummy generator period [23:16]								
0x0b	Dummy generator period [31:24]								
0x0e	Dummy generator address [7:0]								0x000f
0x0f	Dummy generator address [15:8]								
0x10	Dummy generator lower bound [7:0]								0
0x11	Dummy generator lower bound [15:8]								
0x12	Dummy generator upper bound [7:0]								255
0x13	Dummy generator upper bound [15:8]								
0x14	Dummy generator increment [7:0]								1
0x15	Dummy generator increment [15:8]								
0x16	Dummy generator decrement [7:0]								1
0x17	Dummy generator decrement [15:8]								
0x3a	TX_KEEPALIVE_EN	DEV_ADDR_HSIDE	DEV_ADDR_DIST		DEV_ADDR_VSIDE	DEV_ADDR_PART		EDMTB_SKIN_PART	0x80

Tabella 2.2: Una parte dei registri del Tactile Hub che include quelli presi in esame

Facendo riferimento alla Figura 2.1, per generare i campioni all'interno del Tactile Hub si è utilizzato il modulo **asr_gen0** composto da 4 dummy generator, prendendo in considerazione il percorso dei dati attraverso i componenti **acq_merge**, **acq_split**, **pta-sr_sel**, **raw_fifo**, **raw_sample32to16**, **aer_merge**, **gaer_merge**, **hssaer_tx1**.
Si può notare come questo percorso non passi attraverso il blocco di pre-processamento né attraverso quello di generazione degli eventi in quanto si vogliono prendere in esame i campioni prodotti senza ulteriore elaborazione.

Oltre al generatore citato prima ne abbiamo anche un altro, chiamato **asr_gen1**, utile se si vogliono generare dei campioni che non passino attraverso il modulo di pre-processamento **i2c_acq_comp**, ma che passino per **asr2aer**, ovvero per il generatore di eventi.
I dummy generator producono un segnale a dente di sega campionato del quale è possibile impostare la temporizzazione con cui sono generati i campioni, impostando i 4 registri in ordine crescente da 0x08 a 0x0b; di default è impostato un valore per generare campioni ogni 11 ms, tempo d'acquisizione che ho con il blocco **AcqCtrl** attraverso interfaccia ibrida I2C.
Configurando i due registri 0x0e, 0x0f posso impostare l'indirizzo del taxel emulato dal dummy generator, che di default è 12.
Il segnale a dente di sega è caratterizzato da un livello alto e uno basso regolabili anch'essi attraverso 2 coppie di registri, rispettivamente 0x12, 0x13 e 0x10, 0x11.
Abilitando solo un generatore è possibile osservare, in uscita, sempre lo stesso taxel emulato che progressivamente genera dei dati che seguono un certo pattern.
Poiché il protocollo I2C permette la gestione di un massimo di 127 registri e ogni generatore richiede molti registri, avendo in totale 8 dummmy generator, non si riuscirebbe a caratterizzarli tutti se non si utilizzasse un compromesso.
Tale espediente consiste nel selezionare prima, attraverso la scrittura di un registro, il dummy generator, poi nell'impostare tutti i parametri del generatore selezionato.
Per utilizzare solo un dummy generator bisogna disabilitare l'acquisizione dei campioni tramite I2C ibrido scrivendo opportunamente il registro 0x00.
Effettuando una scrittura sul registro 0x03 si può selezionare il percorso dei dati da trasmettere; si può scegliere se effettuare o no il pre-processamento dei dati, si sceglie di non effettuarlo. Sempre in 0x03 è presente un bit per abilitare la ricezione di dati provenienti da un eventuale secondo Tactile Hub.
I dati dal Tactile Hub vengono poi trasmessi ad AS-AER2UART che applica un time-stamp, necessario in quanto i dati ricevuti al PC non sono in tempo reale per la presenza di diversi sistemi che ne rallentano il flusso quali UART, USB, buffer del PC e del sistema operativo.

2.5.2 Scrittura registri con PyFtdi

Attraverso protocollo I2C possono essere scritti e letti i vari registri del Tactile Hub: per far ciò si è utilizzato un modulo FTDI FT2232H Mini Module. Questo dispositivo possiede due interfacce che possono essere configurate come seriali (sincrone o asincrone) o come interfacce parallele FIFO. I due canali possono essere anche configurati in maniera indipendente e questo permette di emulare anche il protocollo I2C. Per configurare questo modulo si è utilizzato Python con l'ausilio di PyFtdi, dei driver per dispositivi FTDI.
Il modulo si comporta da master mentre lo slave è sul Tactile Hub. In Figura 2.10 è riportata la configurazione allestita tra FPGA, trasmettitore UART presente sulla carrier board, modulo FTDI e PC.

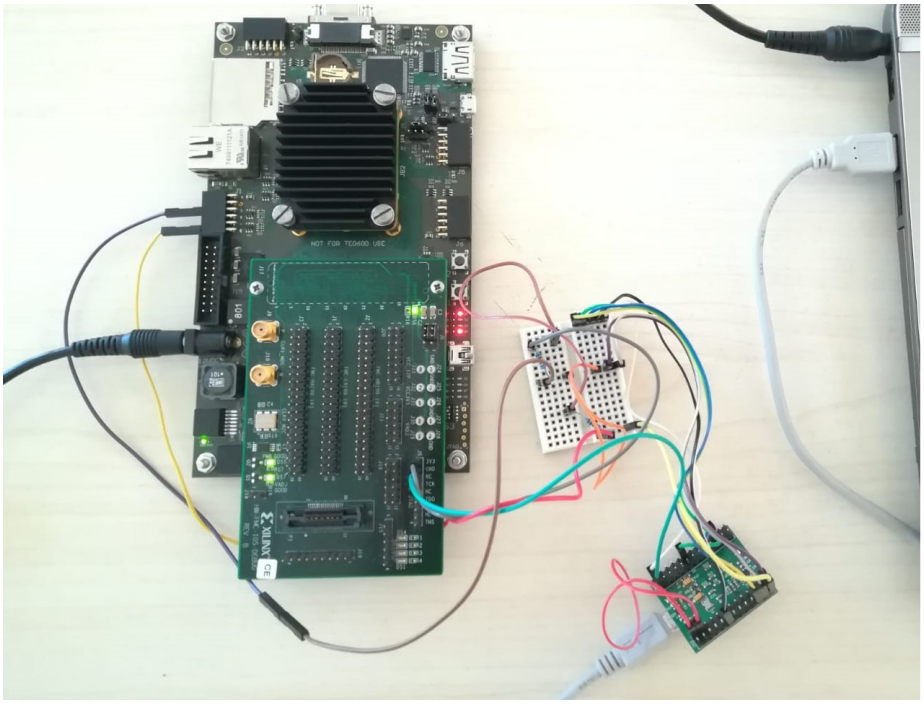


Figura 2.10: Setup allestito che comprende la carrier board con trasmettitore UART, l'FPGA, il modulo FTDI e il PC

Sotto è riportato il codice Python utilizzato per la lettura/scittura dei registri.

```
from pyftdi.i2c import I2cController
from pyftdi.usbtools import UsbTools
```

```
from time import sleep

# istanzio un controller I2C
i2c = I2cController()

# configuro la prima interfaccia (IF/1)
# del dispositivo FTDI come I2C master
i2c.configure('ftdi://ftdi:2232:FTVOU6MM/2')

# porta Tactile Hub (I2C slave)
slave = i2c.get_port(0x1e)

# DUMMYGEN_SEL -> "000", DUMMY_ACQ_EN -> "1"
slave.write_to(0x00,b'\x10')

# abilito solo il dummy generator 0
slave.write_to(0x01,b'\x01')

# SAMPLES_TX_EN -> '1'
slave.write_to(0x03,b'\x02')

# dummy generator address -> '0'
#slave.write_to(0x0e,b'\x00')
#slave.write_to(0x0f,b'\x00')

# disabilito l'accelerometro
slave.write_to(0x05,b'\x00')

# dummy generator increment -> '1'
slave.write_to(0x14,b'\x01')
slave.write_to(0x15,b'\x00')

# dummy generator decrement -> '1'
slave.write_to(0x16,b'\x01')
slave.write_to(0x17,b'\x00')

# per disabilitare TX_KEEPALIVE_EN
slave.write_to(0x3a,b'\x00')

# dummy generator upper bound
#slave.write_to(0x12,b'\xff')
#slave.write_to(0x13,b'\xff')
```

```
# leggo valore registro
value = slave.read_from(0x12,1)
print("0x12 -> ",value)

value = slave.read_from(0x13,1)
print("0x13 -> ",value)

value = slave.read_from(0x10,1)
print("0x10 -> ",value)

value = slave.read_from(0x11,1)
print("0x11 -> ",value)

value = slave.read_from(0x14,1)
print("0x14 -> ",value)

value = slave.read_from(0x15,1)
print("0x15 -> ",value)

value = slave.read_from(0x16,1)
print("0x16 -> ",value)

value = slave.read_from(0x17,1)
print("0x17 -> ",value)

value = slave.read_from(0x0e,1)
print("0x0e -> ",value)

value = slave.read_from(0x0f,1)
print("0x0f -> ",value)
```

Le operazioni preliminari eseguite dal codice sono:

- istanziare un controllore I2C
- configurare un'interfaccia del dispositivo FTDI come I2C master
- impostare una porta I2C slave da pilotare

I comandi fondamentali per leggere e scrivere dai registri sono:

- slave.write_to (**regaddr**, **out**, **relax** = True, **start** = True)
  regaddr $\rightarrow$ indirizzo del registro dello slave da scrivere

out → il buffer del byte da trasferire
relax (bool) → per emettere uno STOP sul bus
start → per emettere una sequenza di start

- slave.read_from (**regaddr**, **readlen**=0, **relax**=True, **start**=True)
  regaddr → indirizzo del registro dello slave dal quale leggere
  readlen → numero di byte da leggere
  relax → per emettere uno STOP sul bus
  start → per emettere una sequenza di start

Il codice Python sopra riportato abilita il dummy generator 0 e il percorso che permette di trasmettere i campioni così come sono stati generati. Viene impostato anche l'indirizzo del taxel emulato, l'incremento e il decremento del generatore dopo un certo periodo di tempo (impostato di default su 11ms) e il valore massimo del generatore; il valore minimo non viene modificato e si utilizza quello di default impostato su 0.
Viene disabilitato anche l'accelerometro e delle commutazioni che servono per evitare problemi relativi all'accoppiamento capacitivo del segnale trasmesso.
Sono stati anche letti diversi registri per verificare che il contenuto fosse in linea con i valori di default.

2.5.3 Pacchetti generati e trasmessi dal Tactile Hub

Il modulo AS-AER2UART trasmette tramite UART al PC i dati generati dal Tactile Hub comprensivi di timestamp. È possibile trasmettere diversi tipi di dati :

- samples (campioni)
- eventi AER
- eventi di errore

In Figura 2.11 è rappresentata la struttura del pacchetto AER, generato e trasmesso dal Tactile Hub e in tabella 2.12 è descritto il significato di ogni campo del pacchetto.

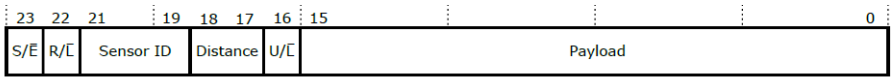


Figura 2.11: Struttura del pacchetto dati AER

Bits	Field	Function
23	S/$\bar{E}$	Packet type: 0: Event 1: Sample
22	R/$\bar{L}$[1]	Body horizontal side: 0: Left 1: Right
21:19	Sensor ID	3-bit Data source ID: 000: – 001: skin taxels 010: accelerometer 011: force-torque 100: microphone 101: encoders 110: optical flow 111: –
18:17	Distance	2-bit Intra-limb/chest localization: 00: far 01: middle 10: near 11: chest
16	U/$\bar{L}$[1]	Body vertical side: 0: Lower 1: Upper
15:0	Payload	See Sections 1.2.1–1.2.3

Figura 2.12: Assegnazione dei bit nel pacchetto dati AER

Ciascun tipo di dato utilizza una struttura specifica per il payload. Per questo lavoro di tesi ci si è concentrati sui samples, quindi di seguito è descritta l'organizzazione dei bit dei pacchetti di questi ultimi.

La trasmissione dei sample richiede due pacchetti consecutivi da spedire; uno per l'indirizzo e l'altro per il reale sample, perchè un singolo pacchetto non ha abbastanza bit per spedire tutte le informazioni in un'unica trasmissione.
Per un dato taxel la sequenza "indirizzo $\rightarrow$ sample" deve essere garantita per evitare ambiguità. Non ci sono ambiguità se il pacchetto dell'indirizzo e del sample sono interlacciati da:

- eventi relativi a un taxel nella stessa FPGA

- pacchetti che provengono da altre FPGA

Questo è vero in quanto i pacchetti possono essere distinti attraverso i bit [23:16]. Qualsiasi situazione errata per cui i dati vengono persi internamente, causando un'associazione errata indirizzo-sample, non viene gestita.

Pacchetti di tipo sample

In Figura 2.13 è mostrata la struttura del payload per un pacchetto di tipo indirizzo, pacchetto che viene sempre spedito prima di uno di sample.

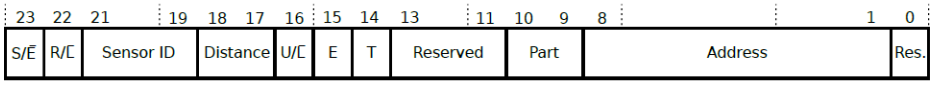


Figura 2.13: Assegnazione dei bit del payload del pacchetto di tipo indirizzo

Il bit 23 è '0', il bit 14 è '1' permettendo la differenziazione con un pacchetto di eventi AER. Il bit 15 è '0', bit 10 e 9 sono usati per identificare ED-MTB multipli localizzati nella stessa area del corpo di iCub [15], i bit [8:1] contengono l'indirizzo del sample del taxel. Sono utilizzati 10 bit per l'indirizzo perchè la più grande patch di pelle ospita 570 taxel.

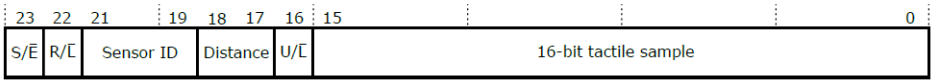


Figura 2.14: Assegnazione dei bit del payload del pacchetto di tipo sample

In Figura 2.14 è mostrata la struttura del payload per un pacchetto di tipo sample; il bit 23 è '1' e tutti i 16 bit del payload sono utilizzati per il sample.

Pacchetti di tipo evento di errore

In Figura 2.15 è mostrata la struttura del payload per un pacchetto di evento di errore.

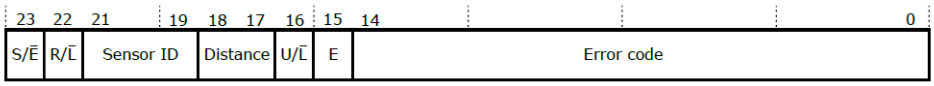


Figura 2.15: Assegnazione dei bit del payload del pacchetto di tipo evento di errore

Il bit 23 è '0', il bit 15 a '1' e i bit [14:0] contengono un codice di errore per identificare il problema che si è verificato.

Pacchetti di tipo evento AER

La Figura 2.16 mostra la struttura del payload per un pacchetto di tipo AER.

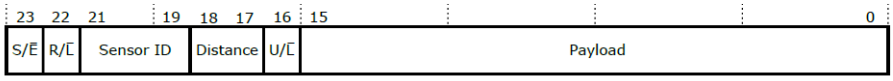


Figura 2.16: Assegnazione dei bit del pacchetto degli eventi del Tactile Hub

Il bit 23, 15 e 14 sono a '0', i bit [10:9] sono usati per identificare ED-MTB multipli localizzati nella stessa area. I bit [8:1] contengono l'indirizzo del taxel. 10 bit di indirizzo sono necessari perchè la patch di pelle più grande ospita 570 taxel. Il significato del bit 13 e 0, che sono i bit di polarità dell'evento sono dettagliati in Tabella 2.3

P_{10}	Polarità
00	DOWN
01	UP
10	CROSS-BASE
11	-

Tabella 2.3: Codifica dei bit di polarità. P_1 è sempre '0' per i dati tattili.

2.5.4 Verifica dei dati ricevuti al PC

Per ricevere ed interpretare i pacchetti di dati si è utilizzato un software scritto in C che acquisisce i dati, trasferiti utilizzando il protocollo UART, attraverso porta USB.
Il software acquisisce dei pacchetti da 32 bit, di cui i 24 bit meno significativi possono rappresentare uno dei pacchetti descritti nella sezione 2.5.3, e li interpreta secondo la struttura mostrata in Figura 2.17.

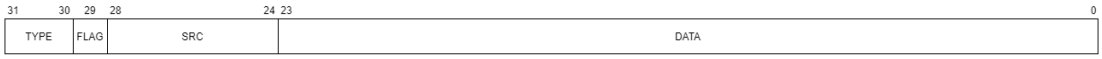


Figura 2.17: Divisione in campi dei pacchetti ricevuti dal PC.

Il programma C presenta un ciclo principale in cui, per ogni pacchetto, vengono analizzati prima i bit del campo **type** e **flag**; per ogni loro combinazione si ha un diverso tipo dato:

- type: ”01” → ERRORE
- type: ”10” (flag: ’0’ → TIMESTAMP, flag: ’1’ → TSWA)
- type: ”11” → CTRL
- type: ”00” (flag: ’1’ → ERRORE, flag: ’0’ → EVENTO)

TSWA è il wraparound ovvero un pacchetto che indica che il contatore del timestamp è arrivato a fine conta.
Se i due bit type sono ”00” e il bit di flag è anch’esso ’0’ si considerano i 24 bit meno significativi del pacchetto su 32 bit, che costituiscono uno dei pacchetti descritti in 2.5.3.

Si possono avere quattro tipi di pacchetto su 24 bit, quindi, determinato quest’ultimo, vengono stampati i suoi campi. In seguito è riportato la porzione principale del codice C.

```
#define EVENT_ID_LIST {’D’, ’U’, ’C’, ’X’}
#define SENSOR_ID_LIST {"None", "Skin", "Accel", " Force", "Mic", "Enc", "Optfl", "Other"}
#define DEV_ADDR_HSIDE_LIST {"Left", "Right"}
#define DEV_ADDR_DIST_LIST {"Far", "Middle", "Near", "Chest"}
#define DEV_ADDR_VSIDE_LIST {"Lower", "Upper"}
#define DEV_ADDR_PART_LIST {"A", "B", "C", "D"}
```

```
static volatile sig_atomic_t run = 1;
const size_t atom_buf_s = ATOM_BUF_SIZE * sizeof( atom);
int atom_buf_off = 0;
atom *patombuf = (atom *) malloc( atom_buf_s);

const char ev_id_s[] = EVENT_ID_LIST;
const char* sensor_id_s[] = SENSOR_ID_LIST;
const char* dev_addr_hside_s[] = DEV_ADDR_HSIDE_LIST;
const char* dev_addr_dist_s[] = DEV_ADDR_DIST_LIST;
const char* dev_addr_vside_s[] = DEV_ADDR_VSIDE_LIST;
const char* dev_addr_part_s[] = DEV_ADDR_PART_LIST;

while(run) {
        int r = read(fd, ((uint8_t*)patombuf) +atom_buf_off, atom_buf_s -atom_buf_off);
        r += atom_buf_off;

        const size_t events = r/sizeof(atom);
        for (register size_t i = 0; i < events ; ++i) {
            register const atom d  = patombuf[i];
            switch(d.type) {
                case 1: //err AE
                    printf( "ERR (%02u): %1x.%06x\n", d.src, d.flag, d.data);
                    break;
                case 2: //ts
                    if( d.flag)
                        printf( "TSWA (%02u): %u\n", d.src, d.data);
                    else
                        printf( "TS (%02u): %u\n", d.src, d.data);
                    break;
                case 3: //ctrl
                    printf( "CTRL (%02u): %1x.%06x\n", d.src, d.flag, d.data);
                    break;
                case 0: //AE
                default:
                    if(d.flag)
                        printf( "AERR (%02u): %u (0x%06x)\n", d.src, d.data, d.data);
                    else
                    {
                        if(0x000000 & d.data)
                            printf("UNK.X: 0x%06x\n", d.data);
                        else{
                        if(0x008000 & d.data) // PACCHETTO EVENTO DI ERRORE
                            printf("ERR: 0x%06x\n", d.data);
                        else{
                        if(0x800000 & d.data) // PACCHETTO ASR SAMPLE
                            printf("AS.SAMPLE (%02u): %u (0x%06x) -> @%s.%s.%s.%s\n",
                            d.src, d.data, d.data,
                            sensor_id_s[(0x380000 & d.data) >> 19],
                            dev_addr_hside_s[(0x400000 & d.data) >> 22],
                            dev_addr_dist_s[(0x060000 & d.data) >> 17],
                            dev_addr_vside_s[(0x010000 & d.data) >> 16]);
```

```
                    else {
                        if (0x004000 & d.data) // PACCHETTO ASR ADDRESS
                            printf("AS.ADDRESS (%02u): %u (0x%06x) -> @%s.%s.%s.%s:%s.0x%02x\n",
                            d.src, d.data, d.data,
                            sensor_id_s[(0x380000 & d.data) >> 19],
                            dev_addr_hside_s[(0x400000 & d.data) >> 22],
                            dev_addr_dist_s[(0x060000 & d.data) >> 17],
                            dev_addr_vside_s[(0x010000 & d.data) >> 16],
                            dev_addr_part_s[(0x000600 & d.data) >> 9],
                            (0x0001fe & d.data) >> 1);
                        else { // PACCHETTO AER EVENTO
                            printf( "AE (%02u): %u (0x%06x) -> %c@%s.%s.%s.%s:%s.0x%02x\n",
                            d.src, d.data, d.data,
                            ev_id_s[(0x000001 & d.data) | ((0x002000 & d.data) >> 13)],
                            sensor_id_s[(0x380000 & d.data) >> 19],
                            dev_addr_hside_s[(0x400000 & d.data) >> 22],
                            dev_addr_dist_s[(0x060000 & d.data) >> 17],
                            dev_addr_vside_s[(0x010000 & d.data) >> 16],
                            dev_addr_part_s[(0x000600 & d.data) >> 9],
                            (0x0001fe & d.data) >> 1);
                        }
                    }
                }
            }
                break;
        }
    }
    atom_buf_off = r % sizeof(atom);
```

Come evidenziato nel codice riportato, per ogni tipo di pacchetto di tipo AER ricevuto vengono interpretati e stampati a video i vari campi che lo compongono secondo quanto descritto in 2.5.3. È possibile anche ricevere dei pacchetti non AER, che differiscono da quelli AER per i bit di type e flag; questi possono essere il timestamp, il TSWA, un pacchetto di errore o uno di CTRL.

Si riporta in Figura 2.18 un estratto dell'output generato dal programma C con i registri del Tactile Hub impostati come mostrato nel codice Python di 2.5.2 in cui è stato abilitato solo un dummy generator.

È possibile notare come la differenza di timestamp tra due pacchetti consecutivi di tipo address o sample sia uguale a 1.100.000, valore di default presente nel registro del periodo del dummy generator. Questo corrisponde a un tempo di 11 ms.
La differenza di timestamp tra un pacchetto di tipo indirizzo e uno di tipo sample facenti

```
TS (00): 12082401
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 12082437
AS.SAMPLE (00): 8912896 (0x880000) -> @Skin.Left.Far.Lower
TS (00): 13182401
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 13182437
AS.SAMPLE (00): 8912897 (0x880001) -> @Skin.Left.Far.Lower
TS (00): 14282401
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 14282437
AS.SAMPLE (00): 8912898 (0x880002) -> @Skin.Left.Far.Lower
TS (00): 15382401
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 15382437
AS.SAMPLE (00): 8912899 (0x880003) -> @Skin.Left.Far.Lower
TS (00): 16482401
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 16482437
AS.SAMPLE (00): 8912900 (0x880004) -> @Skin.Left.Far.Lower
TS (00): 805185
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 805221
AS.SAMPLE (00): 8912901 (0x880005) -> @Skin.Left.Far.Lower
TS (00): 1905185
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 1905221
AS.SAMPLE (00): 8912902 (0x880006) -> @Skin.Left.Far.Lower
TS (00): 3005185
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 3005221
AS.SAMPLE (00): 8912903 (0x880007) -> @Skin.Left.Far.Lower
TS (00): 4105185
AS.ADDRESS (00): 540672 (0x084000) -> @Skin.Left.Far.Lower:A.0x00
TS (00): 4105221
AS.SAMPLE (00): 8912904 (0x880008) -> @Skin.Left.Far.Lower
```

Figura 2.18: Estratto dell'output del programma C che interpreta i pacchetti ricevuti.

parte della stessa coppia address-sample è invece di 36 che corrisponde a 360 ns.

Sono stati effettuati anche test provando ad abilitare più dummy generator in contemporanea evidenziando come le coppie address-sample siano trasmesse in successione, con differenze dei timestamp, tra pacchetti generati dallo stesso generatore, identiche a quelle prima evidenziate.

2.6 Risultati

Dopo aver effettuato i test, riportati in 3, sui vari moduli del setup e sul protocollo di comunicazione, il trasmettitore e ricevitore AS-AER sono stati sostituiti con quelli data-strobe progettati: questi sono stati testati con un apposito testbench su FPGA prima di essere impiegati nel sistema allestito, come discusso in 4.4.
Lo script Python realizzato permette la scrittura e lettura dei registri in modo semplice e veloce utilizzando i driver PyFtdi.

I dati dei campioni, comprensivi di timestamp, inviati dal modulo AS-AER2UART e ricevuti dal PC tramite UART sono stati verificati attraverso un programma C che riceve e interpreta i dati: i pacchetti di tipo address precedono sempre quelli di tipo sample con una differenza temporale tra i due di 360ns. La differenza tra coppie consecutive di pacchetti address-sample rispecchia il periodo di generazione dei campioni programmabile e tutti i valori dei campioni ottenuti palesano quanto programmato nei registri.
In definitiva è possibile affermare che il setup allestito, considerando il percorso dei campioni, si comporta come da specifiche garantendo lo studio del sistema senza la necessità di utilizzare sensori tattili connessi, utilizzando solo un PC con una FPGA e un modulo per la comunicazione seriale per scrivere/leggere i registri.

Capitolo 3

Test

Sono stati effettuati diversi test sui moduli che compongono il setup allestito descritto in 2.5. Per prima cosa è stato testato su FPGA il modulo AS-AER2UART indipendentemente da tutto il resto; successivamente si è testato il Tactile Hub connesso ad AS-AER2UART dopo aver programmato i registri attraverso il modulo FTDI e lo script Python, come discusso in 2.5.2, per poter abilitare i dummy generator e testare il protocollo di comunicazione AS-AER.
Testato il funzionamento del sistema completo si è passati alla sostituzione del trasmettitore e ricevitore AS-AER con quelli data-strobe progettati, descritti in 4, per la non compatibilità dei primi con il setup allestito.
In seguito si sono verificati i dati ricevuti dal PC trasmessi da AS-AER2UART tramite l'utilizzo di un programma C che riceve e interpreta i dati come riportato in 2.5.4.

3.1 Test del modulo AS-AER2UART

È stato caricato su FPGA il bitstream relativo al modulo **AS-AER2UART** per testare il suo funzionamento separatamente dal Tactile Hub, utilizzando, come generatore di dati, un dummy generator interno al modulo, di cui è stato riportato il codice HDL sotto.

```
reg[19:0] cnt;
    reg trg, toggle;
        always_ff @(posedge glsclkp, negedge _grst) begin
                if (~_grst) begin
                        cnt <= 0;
                        trg <= 0;
                        toggle <= 0;
                end
                else begin
                        cnt <= cnt + 1;
                        trg <= ~|cnt;
                        toggle <= ~|cnt ^ toggle;
                end
        end


        assign evs.srdy = trg;
        //assign evs.data = toggle ? 32'h41_42_43_44 : 32'h30_31_32_33;
        assign evs.data = toggle ? 32'h00_00_00_00 : 32'h00_00_00_01;
```

In particolare nella procedura `always_ff` un contatore `cnt` viene aggiornato ad ogni colpo di clock e, in base al suo valore, i segnali `trg` e `toggle`, rispettivamente il source ready e la variabile che mi fa selezionare quale dei due dati trasmettere, variano il loro valore. `evs.data` rappresenta il dato da trasmettere, che può assumere due diversi valori in base alla variabile `toggle`.
Per ogni dato viene trasmesso anche un timestamp, generato all'interno di questo modulo. Sono state testate diverse velocità di trasferimento dati della UART, compresa quella di 12Mb/s, necessaria per i test del sistema completo, ottenendo esiti positivi come mostrato in 3.1, un estratto dell'output del programma in C che riceve e interpreta i dati.
Gli 8 bit più significativi di ciascun pacchetto su 32 bit trasmesso al PC servono per identificare il tipo di pacchetto e la sorgente come descritto in 2.5.4.
Dei dati generati internamente su 32 bit, gli 8 bit più significativi vengono sostituiti da "000", src_id, mentre, per quanto concerne il timestamp, questo è generato su 24 bit e gli 8 bit più significativi sui 32 trasmessi sono "100", src_id. Viene generato anche un segnale di overflow del contatore del timestamp che ha come primi 8 bit "101", src_id.

```
# aeruart-dump initialization ....Ok.
TS (00): 15728641 (0xf00001)
DATA (00): 3224115 (0x313233)
TSWA (00): 0
TS (00): 1048577 (0x100001)
DATA (00): 3224115 (0x313233)
TS (00): 2097153 (0x200001)
DATA (00): 4342596 (0x424344)
TS (00): 3145729 (0x300001)
DATA (00): 3224115 (0x313233)
TS (00): 4194305 (0x400001)
DATA (00): 4342596 (0x424344)
TS (00): 5242881 (0x500001)
DATA (00): 3224115 (0x313233)
TS (00): 6291457 (0x600001)
DATA (00): 4342596 (0x424344)
TS (00): 7340033 (0x700001)
DATA (00): 3224115 (0x313233)
TS (00): 8388609 (0x800001)
DATA (00): 4342596 (0x424344)
TS (00): 9437185 (0x900001)
DATA (00): 3224115 (0x313233)
TS (00): 10485761 (0xa00001)
DATA (00): 4342596 (0x424344)
TS (00): 11534337 (0xb00001)
DATA (00): 3224115 (0x313233)
TS (00): 12582913 (0xc00001)
DATA (00): 4342596 (0x424344)
TS (00): 13631489 (0xd00001)
DATA (00): 3224115 (0x313233)
TS (00): 14680065 (0xe00001)
DATA (00): 4342596 (0x424344)
TS (00): 15728641 (0xf00001)
DATA (00): 3224115 (0x313233)
TSWA (00): 0
TS (00): 1048577 (0x100001)
DATA (00): 3224115 (0x313233)
TS (00): 2097153 (0x200001)
DATA (00): 4342596 (0x424344)
```

Figura 3.1: Output prodotto dal modulo AS-AER2UART, con dati generati dal dummy generator interno, acquisito da PC tramite UART.

È possibile osservare come il timestamp preceda sempre i dati prodotti dal generatore interno. Per ogni riga stampata dal programma viene riportato il src_id fra parentesi e il valore rispettivamente in decimale ed esadecimale eccetto per il TSWA, che rappresenta il segnale di overflow del contatore utilizzato per generare il timestamp.
Quanto ottenuto è in linea con la struttura dei pacchetti: i primi 8 bit di ciascun pacchetto su 32 bit sono utili per codificare il tipo di pacchetto.

3.2 Test del Tactile Hub

Una volta programmati i registri, come descritto in 2.5.1, abilitando un dummy generator, è stato testato il corretto funzionamento del Tactile Hub attraverso la verifica del passaggio di dati tra i diversi moduli della catena presa in considerazione, ovvero quella dei campioni.
La programmazione dei registri effettuata con il codice Python riportanto in 2.5.2 garantisce il trasferimento dei dati tra due blocchi adiacenti ogni 11ms; come metodo di verifica si sono presi in considerazione i segnali di handshake: nel momento in cui il srdy e drdy sono entrambi asseriti c'è passaggio di dati.
Lo spike generato però ha una durata di 10 ns ed è quindi difficile da visualizzare con un oscilloscopio da 100 MHz utilizzato per questo lavoro di tesi; per tale motivo un circuito composto da una serie di flip flop e una porta or è stato utilizzato 3.2 per rendere il segnale più duraturo.

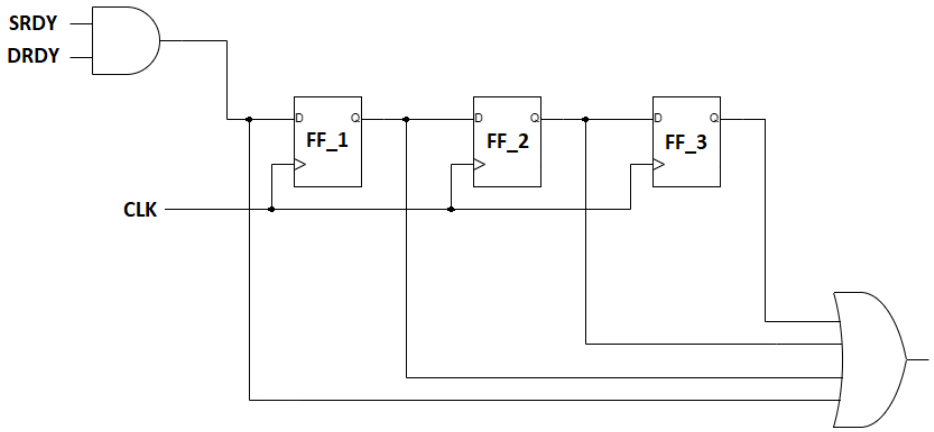


Figura 3.2: Circuito utilizzato per testare il passaggio dei dati tra i moduli del Tactile Hub

Per tutti i moduli presi in esame è stato testato il passaggio dei dati utilizzando il circuito realizzato, collegando ciascuna uscita delle porte or a un pin di I/O della FPGA. Attraverso un oscilloscopio sono stati rilevati gli spike in uscita dai pin di durata di 40 ns, ben visibili, provando il passaggio dei dati nel percorso considerato del Tactile Hub.

3.3 Test del ricevitore AS-AER

Prima di verificare la coerenza dei dati trasmessi dal modulo AS-AER2UART al PC attraverso il programma in C che riceve e interpreta i dati si sono effettuati dei test sul ricevitore AS-AER.
Questi hanno evidenziato, analizzando i segnali di errore con un oscilloscopio, come mostrato in 3.3, l'incompatibilità di questo componente con il setup allestito. Sono stati osservati anche i segnali di handshake tra i primi due sottomoduli di AS-AER2UART dimostrando che non c'era trasmissione di dati con un solo dummy generator attivo: in 3.3 è evidenziato come nessuno spike della and tra il srdy e il drdy è presente mentre in 3.4 si può osservare come il srdy del ricevitore non sia mai asserito.
Si sono sostituiti quindi, ai due moduli AS-AER, il TX e l'RX progettati, descritti successivamente nel capitolo 4. Si è deciso di utilizzare una frequenza di clock al trasmettitore di 100 MHz e una di 200 MHz per il ricevitore per garantire da un lato una trasmissione a 100 Mbps, dall'altro un ricevitore che riesca a campionare più di una volta ciascun bit ricevuto.

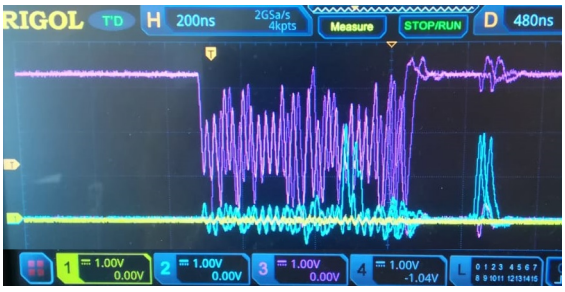


Figura 3.3: In **viola** è mostrato il segnale trasmesso, in **blu** la and di tutti i segnali d'errore, in **giallo** la and tra il srdy e il drdy.

Figura 3.4: In **viola** è mostrato il segnale trasmesso, in **blu** il srdy e in **giallo** il drdy.

3.4 Test del ricevitore data-strobe

Una volta sostituiti il trasmettitore nel design del Tactile Hub e il ricevitore in quello del modulo AS-AER2UART si è testata la ricezione dei dati osservando i segnali di debug generati dal ricevitore descritto in 4.2.
Il segnale di `run_rx`, ovvero il segnale che attesta che il ricevitore è in ricezione, risulta asserito per un intervallo temporale in linea con il tempo di ricezione e ad intervalli regolari, seguendo le differenze temporali tra campioni di tipo sample e address.
I segnali di errore `timeout_error`, `either_error`, `parity_error` sono stati rilevati costantemente inattivi, a dimostrazione del fatto che le ricezioni dei dati trasmessi sono corrette.

3.5 Risultati

Dai test effettuati è stato possibile constatare come i due moduli, presi singolarmente funzionino come dovrebbero, mentre connessi assieme mostrino un funzionamento non corretto dovuto al protocollo di comunicazione non perfettamente compatibile con il setup allestito.
Attraverso questi test è stato possibile quindi determinare il problema nel setup sperimentale e correggerlo utilizzando i due moduli trasmettitore e ricevitore data-strobe progettati inizialmente per effettuare debug su un eventuale futuro Tactile Hub integrato su ASIC.

Parte III

Trasmettitore e ricevitore data-strobe

Capitolo 4

Trasmettitore e ricevitore data-strobe

Sono stati progettati un trasmettitore e ricevitore seriali sulla base della codifica data-strobe, codifica molto semplice che utilizza solo due segnali per comunicare, quello di `data` e di `strobe` per effettuare debug e iniettare/prelevare segnale in generale per i sottomoduli del Tactile Hub.
Il segnale di data è quello che trasporta l'informazione in modo seriale, mentre quello di strobe, composto anch'esso da un solo bit, effettua una transizione ogni volta che il segnale di data non varia rispetto al bit precedente: questo consente di ricreare la temporizzazione della trasmissione dei dati al ricevitore attraverso l'adozione di una semplice porta xor.
In Figura 4.1 è stato riportato un esempio per meglio comprendere il funzionamento della codifica.

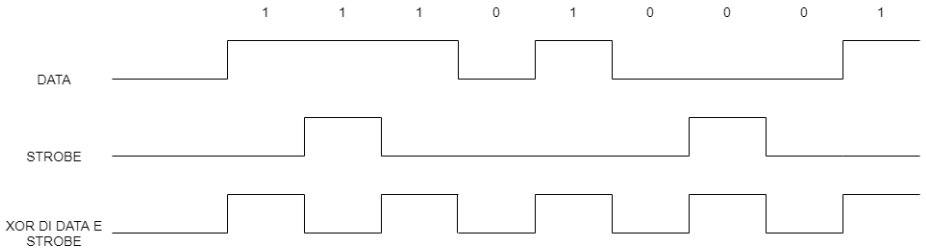


Figura 4.1: Esempio di evoluzione dei segnali di `data`, `strobe` e clock ricostruito (xor dei segnali precedenti) nel tempo.

Tutti i moduli sono stati progettati per gestire un parallelismo variabile dei dati, scelto

a tempo di compilazione. In seguito si fa riferimento al parallelismo dei dati utilizzato per il lavoro di tesi.

4.1 Trasmettitore

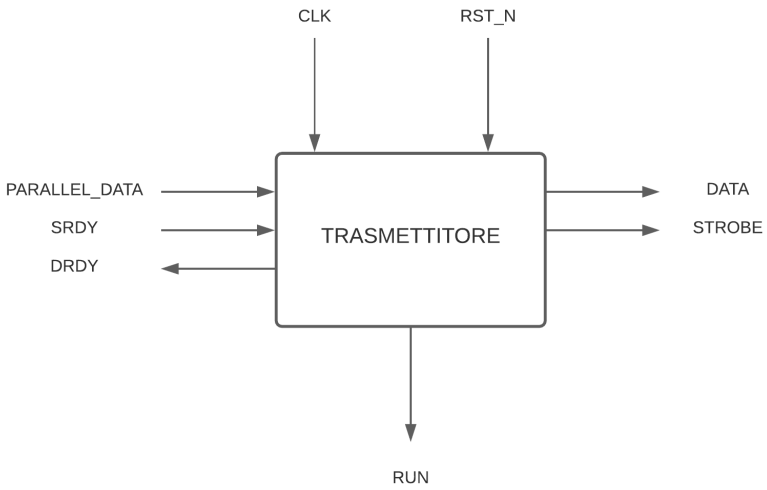


Figura 4.2: Rappresentazione ad alto livello del trasmettitore.

Questo componente, come mostrato in Figura 4.2, si interfaccia da un lato con i segnali `parallel_data`, `srdy` e `drdy`, dall'altro con i segnali di `data`, `strobe`, in più è presente il segnale di `run` per scopi di debug.
Il segnale `parallel_data` serve per acquisire il dato da trasmettere con parallelismo a 32 bit, i segnali di `srdy` e `drdy` servono per il protocollo di handshake tra il trasmettitore e il blocco a monte che fornisce i dati da inviare.
I segnali di `data` e `strobe` sono segnali in uscita dal trasmettitore e portano informazioni sul dato da trasmettere e sulla sua temporizzazione.
Il segnale di `run` indica invece, quando è asserito, che si è in fase di trasmissione.

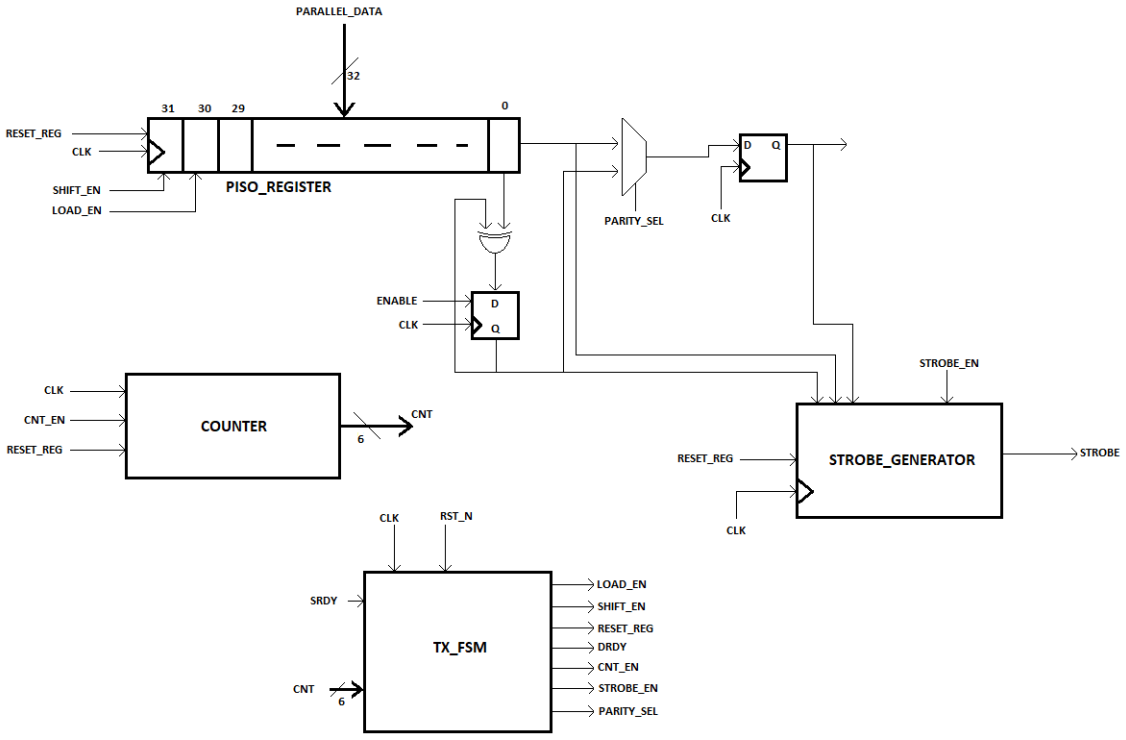


Figura 4.3: Rappresentazione RTL del trasmettitore.

Il datapath, mostrato in Figura 4.3, è composto da pochi e semplici elementi: uno shift register, un generatore del segnale di strobe, un contatore e della logica per generare un bit di parità.

Lo shift register ha il compito di acquisire in parallelo il dato a 32 bit e di trasmetterlo un bit per volta scorrendo ad ogni colpo di clock. Il generatore di strobe invece, ad ogni colpo di clock, quando si è in trasmissione, genera un segnale che è funzione del bit trasmesso in quel colpo di clock e nel colpo di clock precedente.

Il contatore serve per temporizzare la trasmissione e trasmettere esattamente i 32 bit caricati in parallelo nello shift register.

Il flip flop infine serve per ritardare il dato da trasmettere di un colpo di clock per avere i due segnali di `data` e `strobe` perfettamente sincronizzati.

Il bit di parità, trasmesso dopo i 32 bit di dato, viene calcolato mediante l'utilizzo di una porta xor e di un flip flop man mano che i bit di dato scorrono nel registro: la porta xor ha come ingressi l'LSB dello shift register e l'uscita del flip flop, quest'ultimo resettato inizialmente a 0. Nello specifico la parità viene calcolata analizzando uno alla volta i 32

bit caricati in parallelo nel registro e salvando il risultato intermedio nel flip flop; ad ogni shift del registro, anche il flip flop è abilitato.
Tra l'uscita dello shift-register e il flip flop d'uscita è presente un multiplexer, che ha il compito di selezionare o il dato seriale in uscita dal registro oppure il bit di parità.

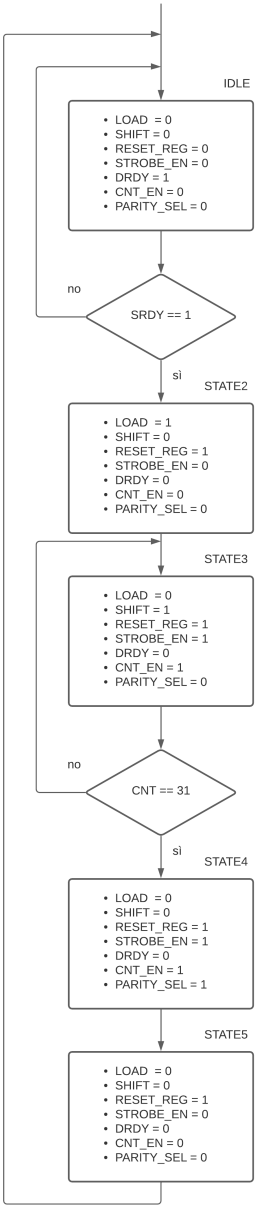


Figura 4.4: Macchina a stati finiti trasmettitore.

L'unità di controllo è stata realizzata con una macchina a stati finiti a cinque stati, come mostrato in Figura 4.4.
Nello stato di `IDLE` vengono resettati tutti i componenti sequenziali e viene asserito il segnale di `drdy` per comunicare al blocchetto posto a monte che il trasmettitore è pronto. Appena viene rilevato il segnale di `srdy` attivo significa che è disponibile il dato da trasmettere, quindi si passa allo `STATO2`, in cui viene asserito il segnale di `load` necessario per il caricamento in parallelo del dato.
Lo stato successivo è lo `STATO3` in cui viene abilitato lo scorrimento del registro, la generazione del segnale di strobe e il contatore. Questo stato viene ripetuto 32 volte, in modo tale da trasmettere tutti i 32 bit presenti nello shift register.
Nello `STATO4` viene modificato infine il selettore del multiplexer in modo da portare in uscita, sul segnale di `data`, al colpo di clock successivo, il bit di parità.

4.2 Ricevitore

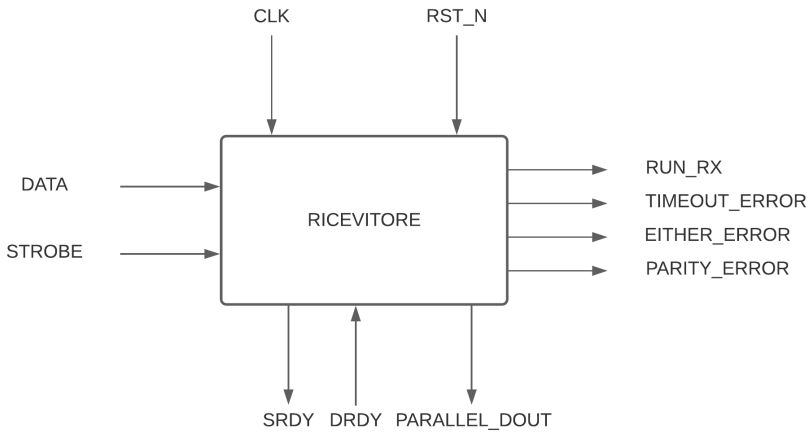


Figura 4.5: Rappresentazione ad alto livello del ricevitore.

Il ricevitore è un componente leggermente più complesso, in quanto deve ricostruire il segnale di clock legato alla temporizzazione dei dati trasmessi e campionare i dati ricevuti in modo seriale, utilizzando quest'ultimo.
I segnali con cui si interfaccia questo componente hardware sono `data`, `strobe` da un

lato, mentre dal lato di collegamento con il blocco a valle troviamo `parallel_dout`, `srdy` e `drdy` come illustarto in Figura 4.5. Sono presenti inoltre il segnale di debug `run_rx` e dei segnali di errore `parity_error`, `timeout_error` e `either_error`.

I segnali di `srdy` e `drdy` hanno la stessa funzione degli stessi segnali presenti nel trasmettitore, i segnali di `data` e `strobe` sono quelli prodotti dal trasmettitore, mentre `parallel_dout` costituisce il dato ricevuto su 32 bit.
Il segnale di `run_rx` è utile per segnalare all'operatore che è in corso una ricezione, `parity_error`, `timeout_error` e `either_error` segnalano rispettivamente un errore sui dati ricevuti(rilevato attraverso il controllo di parità), un errore generato da un'attesa tra un bit ricevuto e il successivo troppo lunga, la variazione simultanea dei segnali di data e strobe.

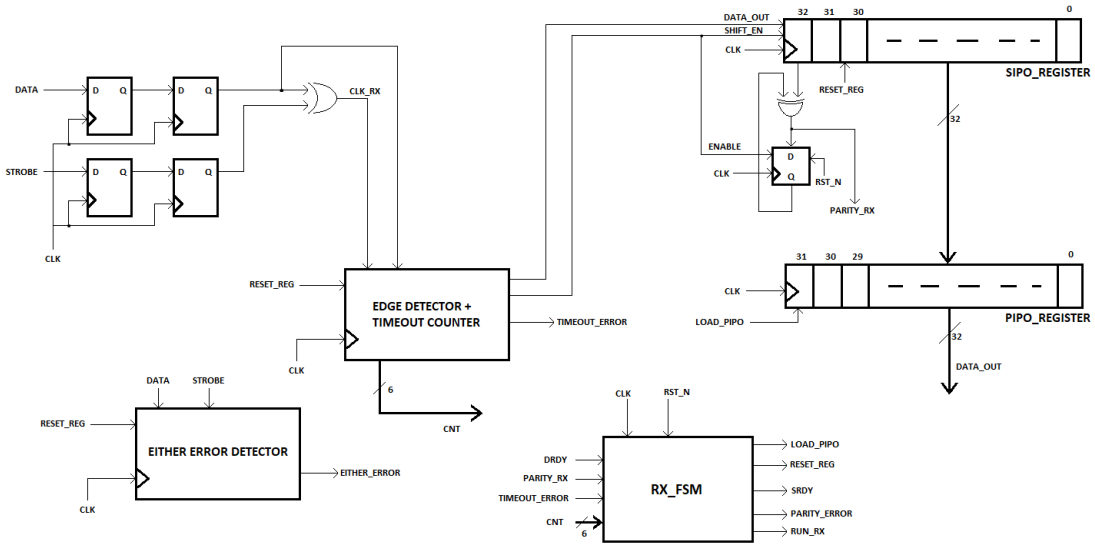


Figura 4.6: Rappresentazione RTL ricevitore

Il datapath del ricevitore, così come la control unit, ha un importante vincolo da rispettare sulla frequenza di clock: questa deve essere almeno uguale o superiore (in teoria) rispetto a quella del trasmettitore per il suo corretto funzionamento; questo perché il ricevitore deve riuscire a campionare almeno una volta il dato in arrivo.
Per poter utilizzare, riducendo la probabilità di metastabilità praticamente a 0, i due segnali provenienti dall'esterno di `data` e `strobe` devono essere campionati attraverso

due flip flop di sincronizzazione in cascata. In uscita dalle due coppie di flip flop c'è una porta xor per generare il segnale di `clk_rx` che serve per capire quando un nuovo dato arriva in ingresso.
Sia il segnale di `data` che di `clk_rx` sono ingressi del blocchetto chiamato "edge_detector + timeout_counter": tale blocco ha due compiti fondamentali, quello di rilevare la variazione del livello del segnale di `clk_rx` per poter campionare i dati in ingresso (campionati sempre attraverso tale blocco) e quello di asserire un segnale di errore (`either_error`) nel caso si attende la variazione del segnale di `clk_rx` per troppo tempo.
Abbiamo poi due registri, di cui un registro di tipo SIPO e uno PIPO.
Il registro SIPO riceve il dato seriale in ingresso direttamente dal blocchetto descritto in precedenza e lo shift viene abilitato anch'esso da quest'ultimo; abbiamo un totale di 33 bit da ricevere in quanto abbiamo 32 bit di dato e un bit di parità.
La stessa logica applicata al trasmettitore è stata replicata anche qui, ma stavolta questa è utile per il controllo di parità e non per la generazione del bit di parità. Essenzialmente se alla fine della ricezione dei 33 bit la parità è 0, la ricezione viene considerata corretta.
Il registro PIPO viene usato per acquisire il dato su 32 bit una volta terminata la ricezione per renderlo disponibile in uscita.
È presente anche un modulo incaricato di segnalare la variazione contemporanea dei segnali di `data` e `strobe`.

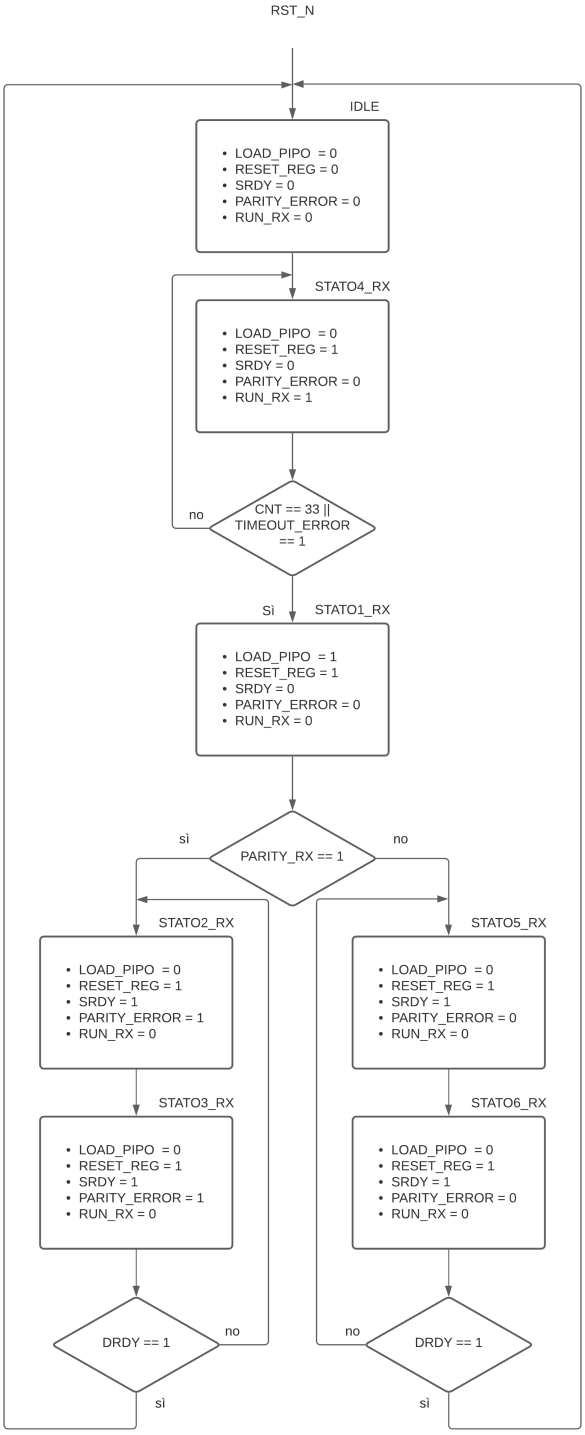


Figura 4.7: Macchina a stati finiti ricevitore

Avendo complicato il datapath attraverso l'adozione del blocchetto "edge_detector + timeout_counter", che è la parte fondamentale del ricevitore, la control unit di conseguenza è molto semplice: abbiamo uno stato di `IDLE`, uno di ricezione, uno di salvataggio dati e altri stati utili a gestire la segnalazione dell'errore di parità.
Nel primo stato, quello di `IDLE`, resettiamo tutti i componenti sequenziali, nello stato `STATO4_RX` siamo in ricezione: fino a quando non si ricevono tutti i 33 bit (32 bit di dato più un bit di parità) oppure non viene asserito il segnale di timeout_error, non si passa allo stato successivo.
Nello stato `STATO1_RX` viene asserito il segnale di `load_pipo` per abilitare il caricamento dei 32 bit di dato nel registro PIPO. Viene controllata anche la parità: se questa è '1', allora c'è stato un errore, altrimenti è '0'. L'errore di parità viene segnalato negli stati successivi attraverso il segnale `parity_error`.
Considerando l'integrazione del ricevitore all'interno del modulo AS-AER2UART e del trasmettitore nel Tactile Hub, si è deciso di utilizzare una frequenza di 200 MHz come riferimento per il ricevitore e una frequenza di 100 MHz per il trasmettitore.
Dopo aver controllato la parità, si hanno quindi stati raddoppiati perché viene asserito il segnale di `srdy`, che deve essere letto dal blocco a valle del ricevitore, che lavora a 100 MHz.

4.3 Testbench Modelsim

In questa sezione vengono mostrati dei frammenti di simulazione del trasmettitore e del ricevitore ottenuti attraverso il software Modelsim. In particolare viene mostrata l'evoluzione di un ciclo completo degli stati delle due FSM con i segnali più rilevanti.
Per questa simulazione sono stati utilizzati un clock di 100 MHz al trasmettitore e uno di 200 MHz al ricevitore, utilizzando gli stessi segnali di temporizzazione delle implementazioni di questi due moduli sul setup allestito.
In Figura 4.8 è mostrata la trasmissione del dato in ingresso al trasmettitore attraverso `parallel_din`; quando viene rilevato asserito il segnale di `srdy` il dato viene acquisito in parallelo dal registro PISO e poi trasmesso un bit per volta tramite il segnale di `data`. È possibile notare come il segnale di `strobe` vari quando il segnale di `data` non varia rispetto al suo valore nel colpo di clock precedente. Viene mostrato il selettore

del multiplexer che varia il suo valore per trasmettere come trentatreesimo bit quello di parità.

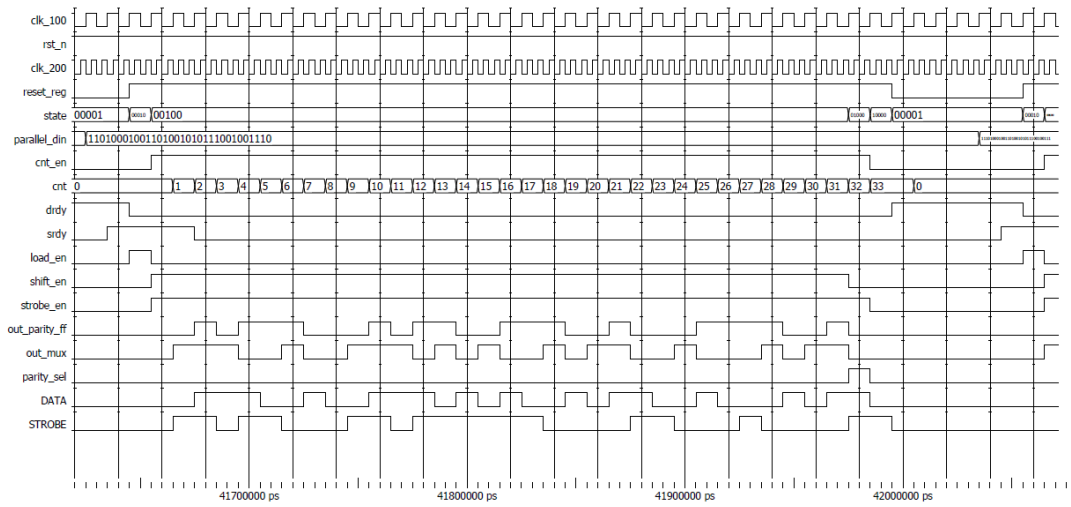


Figura 4.8: Frammento di testbench su modelsim di una trasmissione completa

In Figura 4.9 viene mostrata la ricezione di un dato: non vengono campionati direttamente i segnali di `data` e `strobe` ma quelli di `data_out_sincr2` e `strobe_out_sincr2`, ovvero i segnali in uscita dalle due coppie di sincronizzatori.
Appena viene rilevato il segnale di `clk_rx` a livello logico alto inizia la ricezione dei dati, che continua fino a quando non vengono ricevuti tutti i 33 bit (32 bit di dato e un bit di parità) oppure il segnale di `timeout_error` viene asserito. I dati in ingresso sincronizzati sono campionanti con un clock a frequenza doppia rispetto a quella del trasmettitore e acquisiti solo se il livello del segnale di `clk_rx` è variato.
Una volta ricevuti tutti i 33 bit viene controllata la parità pilotando di conseguenza il segnale `parity_error` e vengono caricati i 32 bit di dato nel registro PIPO.
Viene asserito il segnale di `srdy` per comunicare la disponibilità del dato ricevuto.

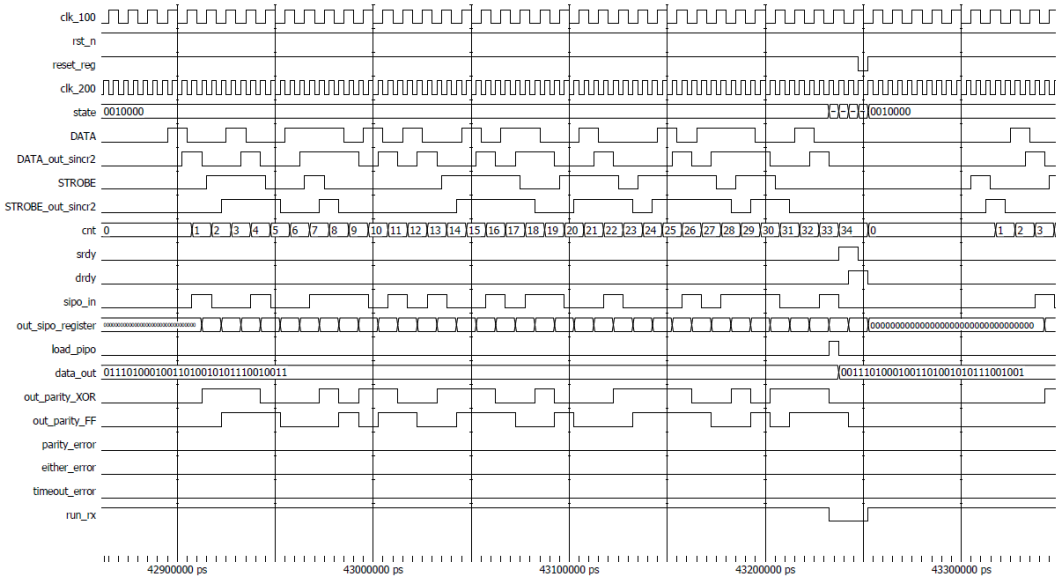


Figura 4.9: Frammento di testbench su modelsim di una ricezione completa

4.4 Testbench su FPGA

Per verificare che i due moduli data-strobe siano stati progettati correttamente, oltre ad eseguire delle simulazioni attraverso il tool Modelsim è stata creata anche una struttura hardware per realizzare un testbench da eseguire su FPGA rappresentata in visione ad alto livello in Figura 4.10.

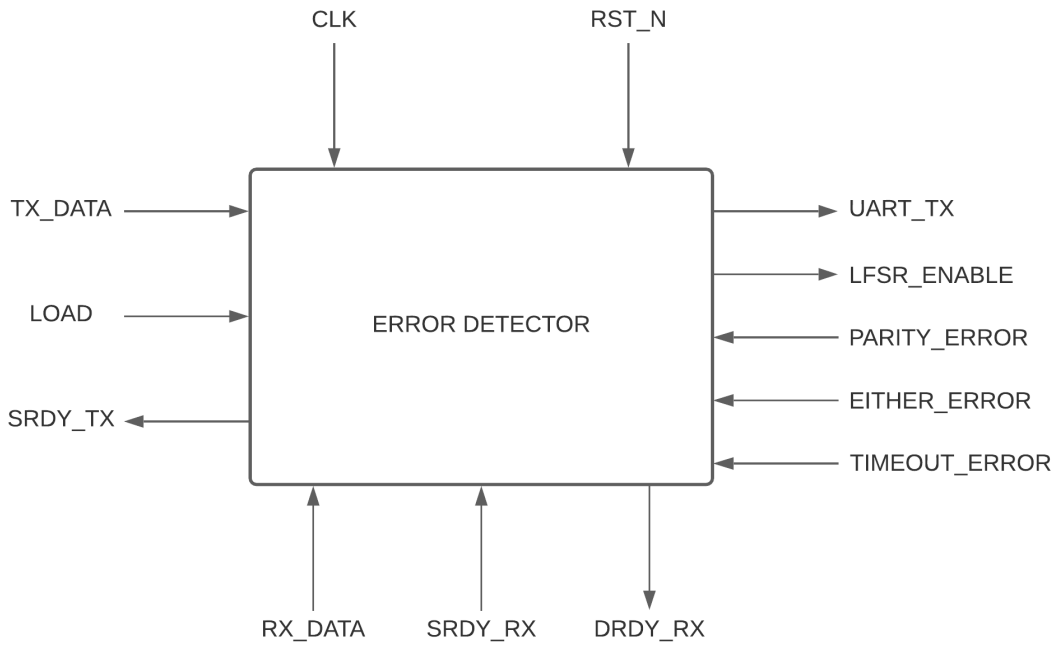


Figura 4.10: Rappresentazione ad alto livello del modulo error detector utile ad effettuare delle statistiche sugli errori rilevati in ricezione.

Questo componente si interfaccia da un lato con il trasmettitore e un registro LFSR modificato attraverso i segnali `tx_data`, `load` e `srdy_tx`.
Il segnale di `tx_data` è collegato direttamente all'LFSR che produce i dati da trasmettere, `srdy_tx`, pilotato da questo modulo, viene utilizzato per avviare il trasmettitore, mentre il segnale di `load` viene utilizzato per constatare l'avvenuto caricamento del dato parallelo da trasmettere.
Nella parte in basso della figura si osserva l'interfaccia con il ricevitore.
Il segnale di `rx_data` è il dato completo ricevuto dal ricevitore, mentre i segnali di `srdy` e `drdy` servono rispettivamente per capire se una ricezione è terminata e per comunicare al ricevitore che il dato ricevuto è stato acquisito dal modulo error detector.
Il segnale di `lfsr_enable` abilita la generazione dei dati pseudocasuali da trasmettere e `uart_tx` serve per la trasmissione di eventuali errori riscontrati durante una serie di 2^{32} trasmissioni.
Infine, i segnali di `parity_error`, `either_error` e `timeout_error` informano il

modulo di diversi errori rilevati dal ricevitore.

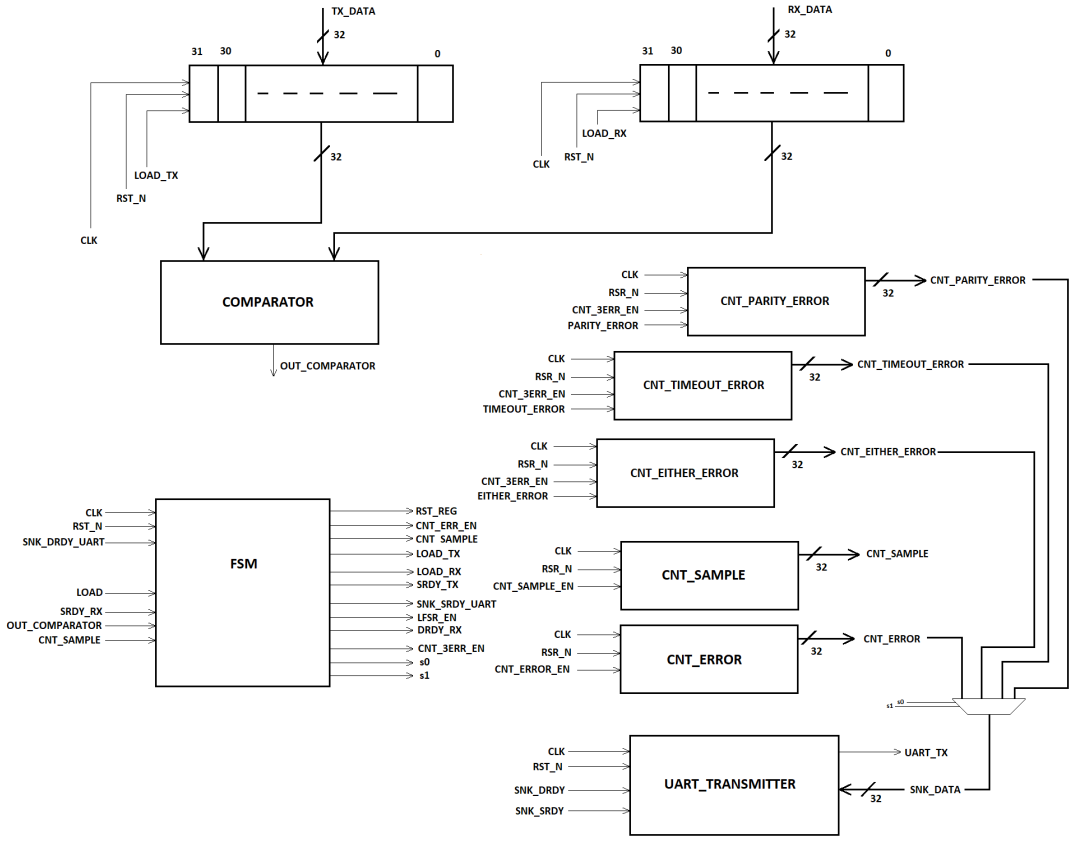


Figura 4.11: Rappresentazione RTL della logica per effettuare il testbench su FPGA

In Figura 4.11 vi è una rappresentazione RTL del modulo error detector.
Si hanno due registri PIPO per salvare il dato da trasmettere prodotto dall'LFSR e quello ricevuto, un comparatore che ha il compito di confrontare i dati presenti nei due registri; in base all'esito della comparazione viene aggiornato il contatore degli errori, che tiene traccia delle differenze rilevate su 2^{32} trasmissioni. È presente un altro contatore per memorizzare il numero di campioni trasmessi per terminare il test dopo 2^{32} trasmissioni. Altri tre contatori sono presenti per memorizzare, sul totale delle 2^{32} trasmissioni, rispettivamente gli errori di parità, quelli di timeout e quelli in cui i segnali di data e strobe sono variati in contemporanea.

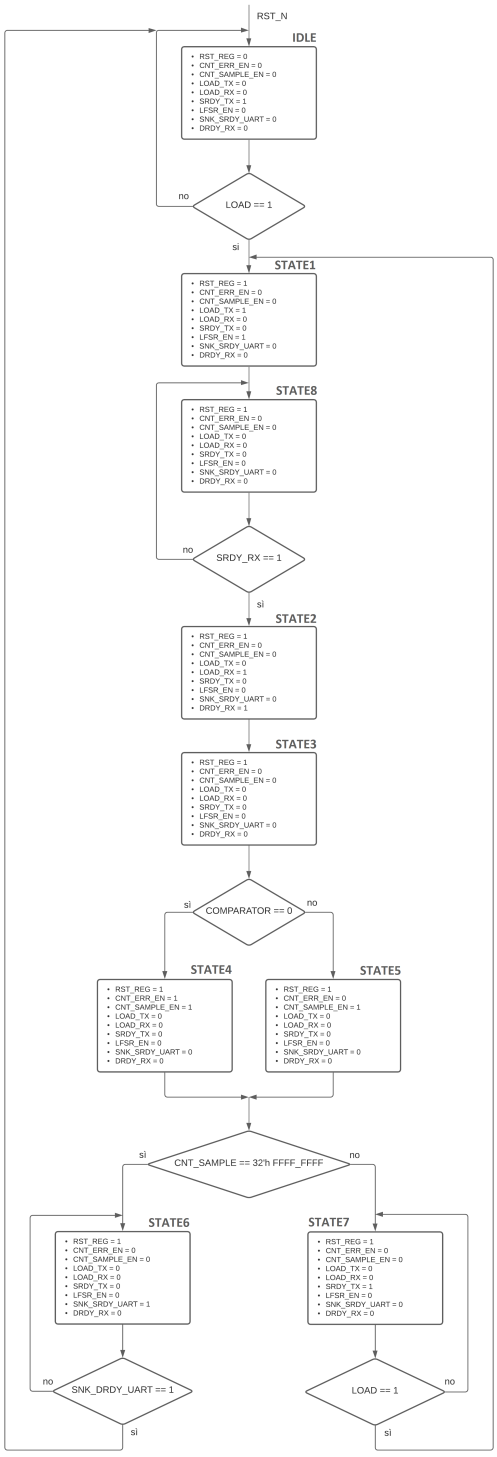


Figura 4.12: Macchina a stati finiti testbench su FPGA

La macchina a stati finiti del testbench è rappresentata in Figura 4.12.
Lo stato di `IDLE` serve per resettare tutta la logica sequenziale, inoltre asserisce il segnale di `srdy_tx` per abilitare il trasmettitore ad avviare una trasmissione. Si attende il segnale di `load` dal trasmettitore per capire quando è stato caricato il dato da trasmettere nel registro PISO, a questo punto, nello `STATE1` viene asserito il segnale di `load_tx` per caricare nel registro PIPO del modulo per il testbench, il dato da trasmettere generato dall'LFSR.
Nello stato `STATE8` si attende il segnale di `srdy_rx` generato dal ricevitore per capire quando la ricezione è terminata.
Lo stato `STATE2` asserisce l'abilitazione dei contatori per gli errori, eccetto cnt_error; viene inoltre asserito il segnale di `lfsr_en` necessario per generare il campione pseudo-casuale successivo dell'LFSR e il segnale di `drdy_rx`, utile per far tornare nello stato di `IDLE` il ricevitore, per essere pronti ad una nuova ricezione.
Vengono poi confrontati i due valori memorizzati nei registri PIPO e, se questi risultano differenti, il segnale di `cnt_err_en` viene asserito per abilitare il contatore cnt_error ad incrementare il suo valore. Inoltre viene anche abilitato il contatore che tiene traccia del numero dei campioni analizzati.
Si trasmettono successivamente i valori dei contatori di errore tramite UART al PC se si sono raggiunte le 2^{32} trasmissioni oppure la FSM ritorna allo `STATE1` per iniziare un altro ciclo.
Il segnale `srdy` del trasmettitore e quello di `drdy` al ricevitore vengono controllati da questo circuito per riuscire a temporizzare correttamente tutte le operazioni da eseguire.

Shift register con retroazione lineare (LFSR)

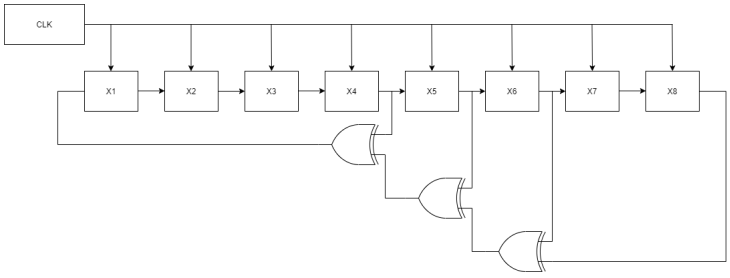


Figura 4.13: LFSR 8 bit

Gli shift register con retroazione lineare, chiamati più comunemente Linear Feedback Shift Register (**LFSR**) [1], sono dei registri a scorrimento i cui dati in ingresso sono prodotti attraverso una funzione lineare dei bit memorizzati internamente, che compongono il cosiddetto **stato interno**.La funzione lineare può essere realizzata solamente con porte XOR o XNOR e lo stato iniziale è chiamato **seme**: a partire da questo valore, ad ogni colpo di clock seguono dei valori deterministici dei bit, quindi, dato il numero finito di stati possibili, questi tenderanno a ripetersi con un certo periodo che può essere molto lungo se si sceglie una funzione di retroazione in modo accurato.

I bit del registro che influenzano la generazione dello stato successivo sono chiamati **tap**.
Un LFSR detto **massimale** assume tutti gli stati interni possibili tranne quello per cui l'uscita della funzione lineare produce tutti zeri, a meno che lo stato iniziale del registro non sia composto da tutti zeri.
E' possibile rappresentare la funzione lineare attraverso un polinomio modulo 2 chiamato **polinomio di retroazione** o **caratteristico**: ogni termine del polinomio rappresenta la posizione di uno specifico tap. Abbiamo poi il termine noto ("1") che non corrisponde a nessun tap. Ad esempio,come mostrato in Figura 4.13, utilizzando un LFSR a 8 bit, se i tap sono in posizione 8, 6, 5 e 4 il polinomio dell'LFSR è:

$$X^8 + X^6 + X^5 + X^4 + 1$$

L'LFSR, inoltre, è massimale:

- se e solo se il polinomio è primitivo [1]
- solo se il numero di tap è pari

Come già detto le sequenze di uscita degli LFSR sono deterministiche: se si conosce lo stato attuale, si può prevedere il successivo, quindi questo è un generatore di numeri binari **pseudo-casuale**.

4.4.1 Configurazione allestita per il testbench su FPGA

In Figura 4.14 è rappresentato il testbench su FPGA allestito comprensivo dei blocchetti da testare; è possibile osservare la presenza di diversi flip flop; essi sono necessari per la

sincronizzazione tra diversi regimi di clock, considerati asincroni tra di loro, per i segnali di `load`, `srdy_tx` e `pseudo_random_value`. Le altre due coppie di sincronizzatori, come affermato in precedenza, sono utili alla sincronizzazione dei segnali di data e strobe.

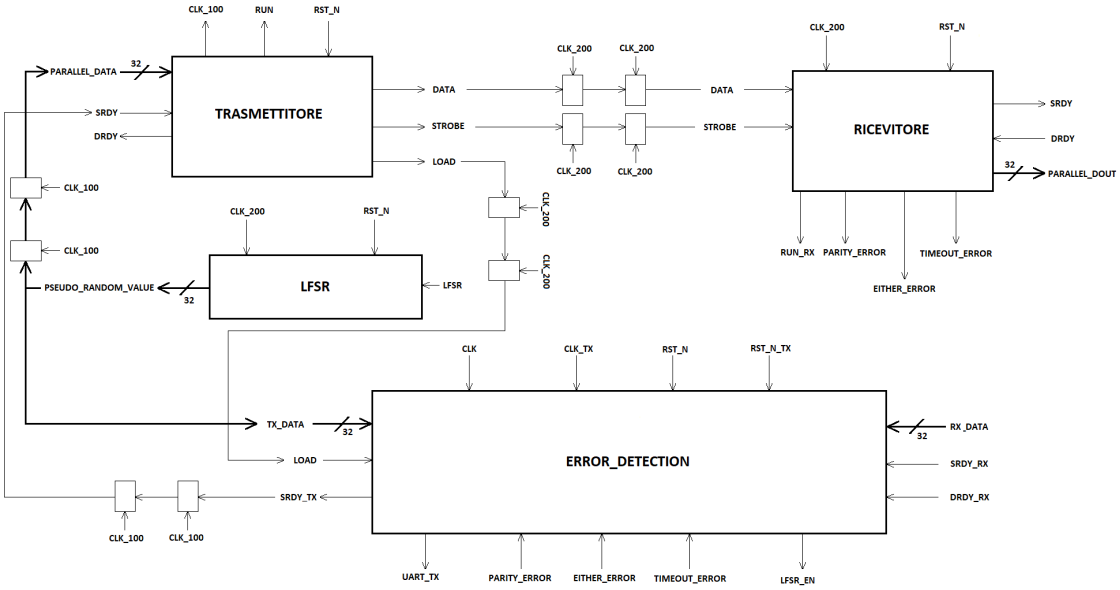


Figura 4.14: Rappresentazione ad alto livello del testbench su FPGA allestito: sono presenti i due moduli di trasmissione e ricezione connessi a un generatore di dati e al di circuito error detection.

Per generare gli ingressi per il trasmettitore e per il circuito di testbench si utilizza un LFSR massimale a 32 bit con seme diverso da zero con polinomio caratteristico:

$$X^{32} + X^{22} + X^2 + 1$$

Questo LFSR è stato modificato come mostrato nel codice HDL sotto riportato per includere il caso con tutti i 32 bit a '0', in quanto un LFSR con seme diverso da zero non ammette tale valore.

```
module LFSR_register(clk, rst_n, pseudo_random_value, lfsr_enable);
input clk;
input rst_n;
```

```
input lfsr_enable;
output [31:0] pseudo_random_value;
reg [31:0] temp;
reg out_xor1, out_xor2, out_xor3;
    always @ (posedge clk/*, negedge rst_n*/)
        begin
            if (! rst_n)
                temp = 32'b00000000_00000000_00000000_00000000;
            else
                if(lfsr_enable)
            if(temp == 32'b00000000_00000000_00000000_00000000)
                temp = 32'b00000000_00000000_00000000_00000001;
        else
            temp = {out_xor3, temp[31:1]};
        end
always @* begin
                out_xor1 = temp[0] ^ temp[10];
                out_xor2 = temp[30] ^ out_xor1;
                out_xor3 = temp[31] ^ out_xor2;
            end
assign pseudo_random_value = temp;
endmodule
```

Al reset il valore dell'LFSR è composto da tutti zeri, quando l'enable viene asserito per la prima volta dopo il reset il valore dell'LSB diventa '1', in questo modo è possibile generare valori pseudocasuali contemplando anche il caso con 32 bit a '0'.

In questo testbench il trasmettitore è stato utilizzato con diverse frequenze comprese in un range [70-170] MHz mentre tutti gli altri moduli sono stati fatti funzionare a 200 MHz. Consideriamo per il momento una frequenza al trasmettitore pari a 100 MHz.
In particolare per i segnali di data e strobe vi è il passaggio dai 100 ai 200 MHz, i dati sono sovracampionati con due flip flop in cascata per ridurre quasi a 0 la probabilità di metastabilità sul secondo flip flop di sincronizzazione.
Il modulo di error_detection lavora a 200 MHz, quindi il segnale di srdy_tx necessita di flip flop di sincronizzazione che lavorino a 100 MHz, il segnale di load, invece, è in ingresso a questo modulo quindi i sincronizzatori utilizzati devono lavorare a una frequenza di 200 MHz.
Anche il modulo LFSR modificato non utilizza lo stesso clock del trasmettitore quindi

sono necessari dei flip flop per sincronizzare i dati in ingresso al trasmettitore che lavora a 100 MHz.

L'utilizzo dei sincronizzatori si è dimostrato un metodo efficace per gestire il passaggio dei dati attraverso diversi regimi di clock: Vivado, il programma utilizzato per la sintesi e l'implementazione, infatti, non ha evidenziato percorsi critici che includono le uscite del secondo flip flop di sincronizzazione per ogni coppia utilizzata.

4.4.2 Risultati sperimentali

Sono stati testati i due moduli progettati utilizzando la stessa FPGA con carrier board e debug card utilizzata nel resto di questo lavoro di tesi. Sulla FPGA è stato caricato il bitstream relativo al circuito di testbench completo rappresentato in figura 4.14.
I due segnali di data e strobe dei due moduli sono stati connessi attraverso dei ponticelli esterni alla FPGA, inoltre, per il trasmettitore è stato utilizzato un clock generato attraverso un generatore di forme d'onda Keithley 3390, mentre quello per tutto il resto del sistema è stato generato internamente alla FPGA; è stata prodotta un'onda quadra dal generatore di funzioni a 20 MHz utilizzata come ingresso per un MMCM della FPGA ottenendo un clock non correlato con quello a 200 MHz utilizzato per il resto del sistema.
Per i test effettuati si è generato un clock al trasmettitore nel range [70-170] MHz.
Dalle simulazioni modelsim, collegando insieme solamente trasmettitore e ricevitore, è stato verificato che i dati vengono correttamente ricevuti se la frequenza di lavoro del TX è uguale o inferiore a quella dell'RX.
Sono stati effettuati dei test, ovvero trasmissioni complete di 2^{32} dati pseudocasuali su 32 bit, a diverse frequenze del TX, con frequenza fissa all'RX per verificare le reali prestazioni del protocollo di comunicazione, raggruppati in Tabella 4.1.

Frequenza TX [MHz]	Errori	Parity error	Timeout error	Either error	Errori [%]	Parity error [%]	Timeout error [%]	Either error [%]	Failed Timing
70	3768492473	1432730314	257830345	0	87.74	33.35	6	0	no
71	3505319372	1718867094	66226	0	81.61	40.02	0	0	no
72	3659886418	1231109728	0	0	85.21	28.66	0	0	no
73	2270860320	1513939396	3	0	52.87	35.24	0	0	no
74	2499582139	1666413426	0	0	58.19	38.79	0	0	no
75	278053853	185382482	0	0	6.47	4.31	0	0	no
76	0	0	0	0	0	0	0	0	no
77	0	0	0	0	0	0	0	0	no
78	0	0	0	0	0	0	0	0	no
79	0	0	0	0	0	0	0	0	no
80	0	0	0	0	0	0	0	0	no
85	0	0	0	0	0	0	0	0	no
90	0	0	0	0	0	0	0	0	no
95	0	0	0	0	0	0	0	0	no
96	0	0	0	0	0	0	0	0	no
97	0	0	0	0	0	0	0	0	no
98	0	0	0	0	0	0	0	0	no
99	0	0	0	0	0	0	0	0	no
100	0	0	0	0	0	0	0	0	no
101	0	0	0	0	0	0	0	0	no
102	0	0	0	0	0	0	0	0	no
103	0	0	0	0	0	0	0	0	no
104	0	0	0	0	0	0	0	0	no
105	0	0	0	0	0	0	0	0	no
106	0	0	0	0	0	0	0	0	no
107	0	0	0	0	0	0	0	0	no
108	0	0	0	0	0	0	0	0	no
109	0	0	0	0	0	0	0	0	no
110	0	0	0	0	0	0	0	0	no
111	0	0	0	0	0	0	0	0	sì
112	0	0	0	0	0	0	0	0	no
113	0	0	0	0	0	0	0	0	no
114	0	0	0	0	0	0	0	0	no
115	0	0	0	0	0	0	0	0	no
116	0	0	0	0	0	0	0	0	no
117	0	0	0	0	0	0	0	0	no
118	0	0	0	0	0	0	0	0	no
119	0	0	0	0	0	0	0	0	no
120	0	0	0	0	0	0	0	0	no
130	0	0	0	0	0	0	0	0	sì
140	0	0	0	0	0	0	0	0	sì
145	0	0	0	0	0	0	0	0	sì
146	0	0	0	0	0	0	0	0	sì
147	0	0	0	0	0	0	0	0	no
148	4218041111	2148459740	243969349	0	98.2	50.02	5.68	0	sì
149	4200666156	2147102554	453142224	0	97.80	49.99	10.55	0	sì
150	3346972944	1698516926	1270709	0	77.92	39.54	0.02	0	sì
160	0	0	0	0	0	0	0	0	no
170	4290404135	2149496532	122394252	0	99.89	50.04	2.84	0	no

Tabella 4.1: Errori rilevati nei test per frequenze al TX nel range [70-170] MHz

L'ultima colonna presente in Tabella 4.1 evidenzia un timing non rispettato per alcuni path del trasmettitore ad alcune frequenze dello stesso; sono state effettuate delle modifiche non riportate in questa tesi che non hanno portato a miglioramenti della situazione.
I risultati ottenuti alla frequenza di 160 MHz al trasmettitore risultano discordanti con il resto delle misure effettuate, probabilmente perché, a quella frequenza, non ci sono violazioni di timing.

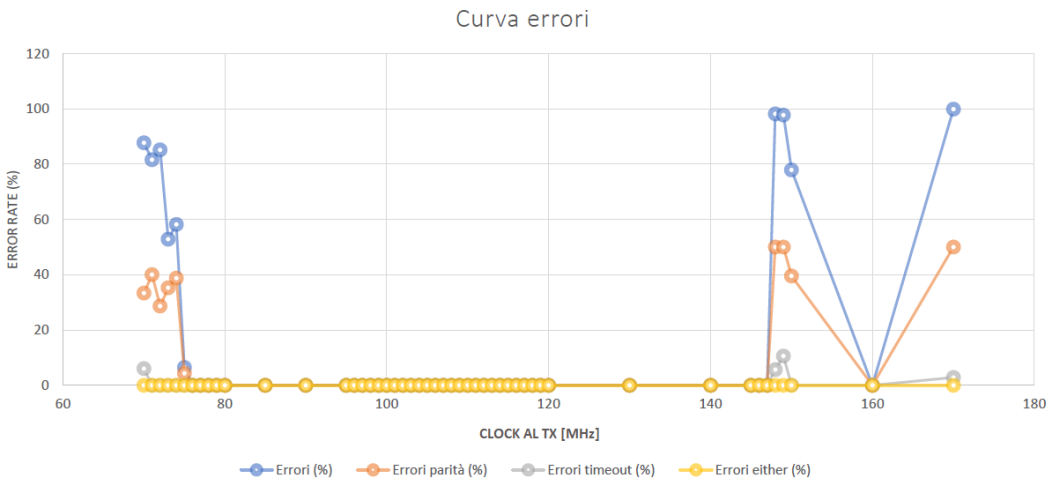


Figura 4.15: Curva degli errori in funzione della frequenza di clock al TX

Per ogni frequenza utilizzata al TX sono state segnalate delle violazioni del timing, ma queste sono relative ad alcuni dei path che includono al loro interno il primo della serie di due flip flop di sincronizzazione utilizzati nel circuito di testbench 4.14. Tali violazioni tuttavia sono da considerarsi regolari in quanto il clock utilizzato dai flip flop di sincronizzazione ha una temporizzazione diversa rispetto ai suoi dati in ingresso.

In Figura 4.15 è riportato un grafico che rappresenta l'occorrenza in percentuale di tutti gli errori considerati in questo testbench in funzione della frequenza utilizzata al trasmettitore.
Si può notare come vengano ricevuti correttamente i dati, senza errori, per un range molto ampio di frequenze al trasmettitore; inoltre la presenza di errori alle basse frequenze può essere giustificata dalla presenza del `timeout counter` al ricevitore il quale viene aggiornato a una frequenza di 200 MHz ogni volta che non viene rilevata una variazione del segnale di clk_rx a ricezione avviata. Quando questo contatore conta poche unità viene asserito un segnale di `timeout_error` il quale blocca la ricezione ottenendo una differenza certa tra il segnale trasmesso e quello ricevuto.
Alle alte frequenze probabilmente gli errori rilevati dipendono dalle violazioni del timing al trasmettitore.

Capitolo 5

Conclusioni e prospettive future

Lo scopo di questa tesi è stato implementare e testare su FPGA un setup che comprende il Tactile Hub e il modulo AS-AER2UART. È stato necessario, per utilizzare una sola FPGA, creare un unico progetto HDL che includesse entrambi i moduli, dopo averli testati separatamente.
Un modulo FTDI 2232H è stato utilizzato assieme a dei driver Python per programmare i registri presenti nel Tactile Hub tramite protocollo I2C; la scrittura di questi registri serve per abilitare dei generatori di campioni del Tactile Hub per simulare stimoli tattili prodotti da eventuali sensori connessi ad esso.
Questi campioni vengono inviati utilizzando il protocollo AS-AER al modulo AS-AER2UART che associa ad essi un timestamp. I campioni generati, comprensivi di timestamp, sono poi trasmessi tramite UART al PC e interpretati da un software C.
Secondo obiettivo di questo lavoro di tesi è stato progettare un trasmettitore e ricevitore seriali basati su codifica data-strobe da utilizzare per scopi di debug all'interno del Tactile Hub in futuro.
Questi sono stati verificati prima attraverso testbench su Modelsim e, successivamente, attraverso una struttura hardware progettata per eseguire un testbench su FPGA ottenendo in entrambi i casi esito positivo. In particolare, il testbench su FPGA ha mostrato il range della frequenza di clock al trasmettitore per cui, con un clock di 200 MHz al ricevitore, non si ottengono errori effettuando la trasmissione di tutti i possibili pattern su 32 bit generati attraverso un LFSR modificato; questo range di frequenze è [76-147] MHz.

Il TX e l'RX AS-AER sono stati sostituiti con quelli nuovi progettati in quanto non completamente compatibili con il setup sperimentale allestito. Tuttavia i risultati ottenuti con questo setup sono in linea con quanto atteso: i sample ricevuti dal PC e prodotti dai dummy generator hanno la giusta temporizzazione e rispettano la struttura dei pacchetti oltre ad assumere il comportamento dettato dai registri che si sono programmati.

Una delle possibili strade future da intraprendere è l'integrazione su ASIC del Tactile Hub per miniaturizzare il modulo e dissipare meno potenza, cosa fondamentale per l'utilizzo in ambiente robotico. In aggiunta, sempre nell'ottica di un possibile ASIC, l'impiego del TX e dell'RX progettati all'interno della struttura del Tactile Hub per poter effettuare test e debug.

Bibliografia

[1] https://en.wikipedia.org/wiki/linear-feedback_shift_register.

[2] http://www.analog.com/media/en/technical-documentation/data-sheets/ad7147.pdf.

[3] Transport-independent protocols for universal aer communications. In *Neural Information Processing*, volume 9492 of *Lecture Notes in Computer Science*, pages 675–684. Springer Nature, November 2015.

[4] Emanuele Baglini, Giorgio Cannata, and Fulvio Mastrogiovanni. Design of an embedded networking infrastructure for whole-body tactile sensing in humanoid robots. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 671–676, 2010.

[5] Chiara Bartolozzi, Paolo Motto Ros, Francesco Diotalevi, Nawid Jamali, Lorenzo Natale, Marco Crepaldi, and Danilo Demarchi. Event-driven encoding of off-the-shelf tactile sensors for compression and latency optimisation for robotic skin. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 166–173, 2017.

[6] Natale L. Nori F. et al. Bartolozzi, C. Robots with a sense of touch. *Nature Mater*, 15:921–925, 2016.

[7] Hans Kristian Otnes Berge and Philipp Hafliger. High-speed serial aer on fpga. In *2007 IEEE International Symposium on Circuits and Systems*, pages 857–860, 2007.

[8] K.A. Boahen. Point-to-point connectivity between neuromorphic chips using address events. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(5):416–434, 2000.

[9] Stefano Caviglia, Luigi Pinna, Maurizio Valle, and Chiara Bartolozzi. An event-driven posfet taxel for sustained and transient sensing. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 349–352, 2016.

[10] Daniel B. Fasnacht, Adrian M. Whatley, and Giacomo Indiveri. A serial communication infrastructure for multi-chip address event systems. In *2008 IEEE International Symposium on Circuits and Systems*, pages 648–651, 2008.

[11] T. Iakymchuk, A. Rosado, T. Serrano-Gotarredona, B. Linares-Barranco, A. Jiménez-Fernández, A. Linares-Barranco, and G. Jiménez-Moreno. An aer handshake-less modular infrastructure pcb with x8 2.5gbps lvds serial links. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1556–1559, 2014.

[12] J.W. Miller J.C. Mallinson. C. *Radio and Electronic Engineer*, 47(4):172–176, 1977.

[13] P. Motto Ros M. Laterza. Ad7147 configuration. *Istituto Italiano di Tecnologia*, 2018.

[14] Marek Miskowicz. Send-on-delta concept: An event-based data reporting strategy. *Sensors*, 6(1):49–63, 2006.

[15] Lorenzo Natale, Chiara Bartolozzi, Francesco Nori, Giulio Sandini, and Giorgio Metta. icub. *Humanoid Robotics: A Reference*, page 291–323, Oct 2018.

[16] Francesco Giovannini Giorgio Metta Lorenzo Natale Nawid Jamali, Marco Maggiali. A new design of a fingertip for the icub hand. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2705–2710, 2015.

[17] Christoph Posch, Teresa Serrano-Gotarredona, Bernabe Linares-Barranco, and Tobi Delbruck. Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output. *Proceedings of the IEEE*, 102(10):1470–1484, 2014.

[18] P. Motto Ros. Synthedtacthub_top_schema. *Istituto Italiano di Tecnologia, v0.5.*

[19] Paolo Motto Ros, Marco Crepaldi, Chiara Bartolozzi, and Danilo Demarchi. Asynchronous dc-free serial protocol for event-based aer systems. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 248–251, 2015.

[20] Carlos Zamarreno-Ramos, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. An instant-startup jitter-tolerant manchester-encoding serializer/deserializer scheme for event-driven bit-serial lvds interchip aer links. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(11):2647–2660, 2011.