POLITECNICO DI TORINO

Dipartimento di Elettronica e Telecomunicazioni
Corso di Laurea Magistrale in Ingegneria Elettronica

Riduzione dei tempi di simulazione mediante modellazione analogica comportamentale e semplificazione del DSP negli IP SerdDes a 64Gbps

Sessione di laurea: Dicembre 2025

Relatore: Guido Masera
Tutor aziendale: Darjn Esposito
Candidato: Vincenzo Castilletti (s323764)

Abstract

La crescente complessità delle IP PHY SerDes necessaria per garantire elevati data-rate in applicazioni come High Performance Computing (HPC), Data Center e AI/ML, ha reso la simulazione di questi dispositivi sempre più onerosa dal punto di vista computazionale. Questa tesi in azienda, svolta in Synopsys, si concentra sul miglioramento delle velocità di simulazione degli IP Synopsys (PHY), prendendo come riferimento un IP PCIe6. Il lavoro si articola in due principali direzioni: l'ottimizzazione delle simulazioni digitali e lo sviluppo di modelli analogici comportamentali.

La prima parte del lavoro riguarda l'ottimizzazione dei tempi di simulazione digitali, fondamentale per consentire ai customer un'accurata verifica dei loro SoC, contenenti più istanze dei SerDes PCIe6 prodotti da Synopsys. A livello customer, per motivi di protezione della proprietà intellettuale, le simulazioni sono basate su netlist GTECH, sintetizzata tramite una libreria di standard cell technology independent. Nel contesto di verifica delle IP a livello SoC, le simulazioni vengono effettuate utilizzando canali di trasmissione ideali, pertanto l'uso di tecniche di avanzate di DSP non è necessario. Da ciò nasce la possibilità di intervenire, tramite strumenti di Simulation Profiling integrati nei tool di simulazione, per individuare quali blocchi DSP risultano rilevanti per i tempi di simulazione, e semplificarli. Dato che tali modifiche devono essere applicate solo in fase di simulazione, sono state introdotte tramite direttive di preprocessore.

La seconda parte del lavoro riguarda lo sviluppo di modelli analogici utili per la verifica ad high-coverage della logica hardware e del firmware durante la fasi di calibrazione dei circuiti analogici. L'architettura del SerDes presa come riferimento implementa una serie di blocchi logici per l'esecuzione degli algoritmi di calibrazione. L'utilizzo di modelli complessi e dettagliati dei circuiti analogici non renderebbe possibile effettuare queste simulazioni a causa dell'eccessivo tempo di esecuzione. Per verificare il loop di calibrazione, sono stati sviluppati modelli in Verilog che emulano il comportamento dei blocchi analogici durante la fase di calibrazione generando tutti i segnali necessari per l'interfacciamento con il dominio digitale. Questi modelli sono stati sviluppati a partire dalle specifiche funzionali fornite dall'azienda e dall'analisi degli schematici analogici. L'obiettivo è quello co-simulare firmware, RTL e macro analogica, per analizzarne l'interazione in tempi di simulazione ridotti. Anche

in questo caso, l'utilizzo di strumenti di simulation profiling integrati nei tool di simulazione, ha permesso di valutare l'impatto di questi modelli sul tempo complessivo di simulazione, focalizzandosi sull'individuazione di soluzioni semplici ed efficaci. Infine, dall'esecuzione di simulazioni utilizzando i modelli complessi preesistenti e i modelli sviluppati si è evinta una riduzione dei tempi di simulazione. I risultati ottenuti evidenziano benefici sia quantitativi, con tempi di simulazione ridotti, sia qualitativi, grazie alla semplificazione del debug rispetto a modelli analogici complessi.

Indice

Elenco delle figure

Capitolo 1

Introduzione e motivazioni

Il trend relativo alla domanda di prestazioni e alla banda di comunicazione nel mercato dei data center e dell'High-Performance Computing (HPC) mostra una crescita costante. Applicazioni come Artificial Intelligence (AI) e Machine Learning (ML) rappresentano un chiaro esempio dell'aumento dei volumi di calcolo e di trasferimento dati: i modelli moderni richiedono non solo una fase di training intensiva, ma anche elevate capacità di inferenza. Anche l'Internet of Things (IoT), con il numero sempre crescente di dispositivi connessi e il loro continuo scambio di informazioni, contribuisce ad accrescere ulteriormente il fabbisogno di banda e di potenza computazionale. Tutte queste applicazioni si prestano bene al Cloud Computing, ovvero all'elaborazione dei dati in remoto, con un conseguente aumento del carico sui data center.

Parallelamente, anche in settori come l'industria e l'automotive si osserva un incremento delle prestazioni e della banda di comunicazione richieste. Un esempio significativo è rappresentato dagli Advanced Driver Assistance Systems (ADAS), che per eseguire i propri algoritmi necessitano di un elevato numero di sensori e quindi di una grande mole di dati da processare. Queste applicazioni hanno guidato lo sviluppo tecnologico di dispositivi in grado di soddisfare le nuove esigenze del mercato.

Se da un lato lo sviluppo di System On Chip (SoC) allo stato dell'arte ha portato ad uno scaling tecnologico che ha permesso un incremento di prestazioni richiesto, ciò non può essere detto per ciò che si occupa di far comunicare il chip con il mondo esterno, come le interconnessioni chip-to-chip [3]. A causa di limiti fisici, lo scaling delle interconnessioni e dei packaging non è stato efficiente quanto quello sulla logica, diventando il collo di bottiglia principale nello sviluppo di un chip, portando al cosiddetto *pad-limited design*. Questo fenomeno è inoltre amplificato dal fatto che l'incremento di prestazioni si traduce in una maggior mole di dati da processare, stressando maggiormente i limiti delle interfacce di comunicazione.

Come mostrato in figura 1.1 l'incremento della banda di comunicazione cresce in maniera esponenziale, in accordo con quanto accade per la densità di capacità di calcolo dei in un SoC.

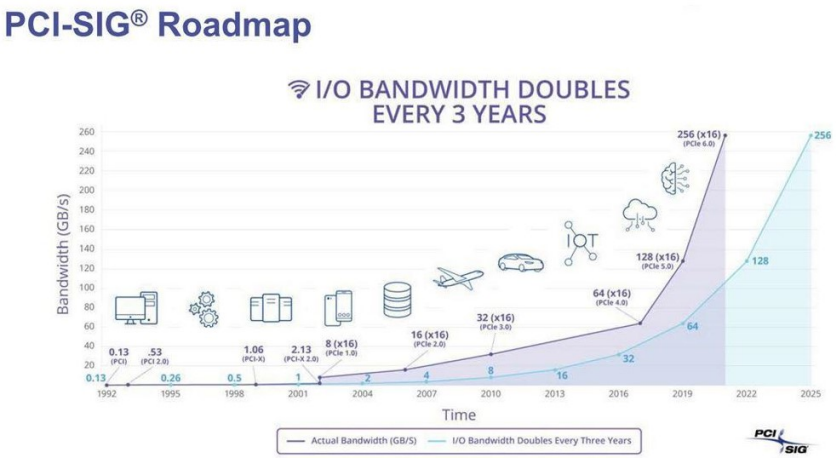


Figura 1.1: I/O Bandwidth vs Year

L'utilizzo di tecniche di trasmissione parallela dei dati non sarebbe in grado di sostenere questo aumento di prestazioni. In questo genere di interfacce di comunicazione, sarebbe necessario agire principalmente sull'incremento del numero di pin e non sulla frequenza di lavoro. L'aumento del numero di pin è una opzione non sostenibile per diverse motivazioni, sia legate all'area e al non efficace scaling dei pin di IO di un die, sia su aspetti legati allo skew a al sincronismo tra i vari bit di un bus. [1] Immaginando infatti applicazioni dove sono presenti bus per connessioni chip-to-chip in un pcb, il routing potrebbe causare problematiche di skew e un non corretto allineamento dei dati da trasmettere, senza parlare di problematiche di crosstalk, potenza, interferenza elettromagnetica ...

Una tipologia di interfacce che permettono di assecondare la crescita di prestazioni senza risultare un collo di bottiglia sono le interfacce seriali, chiamate Ser-Des, attraverso le quali è possibile gestire l'incremento di velocità aumentando la frequenza di trasmissione. Lo stato dell'arte di questi dispositivi possiede una complessità non indifferente, al fine di sopprimere i vincoli che caratterizzano queste tipologie di interfacce di comunicazione. Ne è di esempio l'interfaccia PCIe, sviluppata e mantenuta dall'associazione Peripheral Component Interconnect - Special Interest Group (PCI-SIG), che al fine di adeguarsi alla richiesta di banda sempre maggiore ha visto un aumento della complessità del protocollo da una generazione alla successiva. Basta citare il passaggio dalla modulazione NRZ a quella PAM-4 [9]. Come possibile intuire, un dispositivo del genere per funzionare correttamente richiede l'implementazione di tecniche complesse, sia dal punto di vista comuninicazionistico, implementando algoritmi di Digital Signal Processing (DSP) ed equalizzazione avanzata per soccombere alle non idealità introdotte dai canali di trasmissione, sia dal punto di vista tecnologico che architetturale, al fine di garantire dei dispositivi che funzionino rispettando i vincoli Power Performance Area (PPA)

richiesti.

La progettazione di queste interfacce richiede architetture allo stato dell'arte, capaci di integrare componenti digitali complessi e blocchi analogici di precisione. Tale complessità si traduce in lunghi tempi di sviluppo e in una fase di verifica particolarmente onerosa. Per semplificare questo processo, i Ser-Des vengono spesso distribuiti sotto forma di Intellectual Property (IP), ovvero moduli riutilizzabili facilmente integrabili all'interno dei SoC, garantendo di accelerare il *silicon bring-up*.

In questo contesto si inserisce il presente lavoro di tesi, svolto in collaborazione con *Synopsys*, con l'obiettivo di incrementare le prestazioni delle simulazioni di sistemi Ser-Des PCIe. Synopsys si occupa della progettazione di IP Phisical Layer (PHY) PCIe 6.0 che consentono connettività real-time a bassa latenza e ad elevato throughput per HPC, storage, e AI SoCs.

L'IP sviluppato da Synopsys sul quale si basa l'elaborato consiste nell'implementazione del PHY, ovvero il layer a più basso livello che comprende la parte digitale e la parte analogica, che chiameremo rispettivamente *macro digitale* e *macro analogica*. É possibile effettuare una distinzione anche tra *soft macro* e *hard macro*. La prima è quella parte di cui è implementata sotto forma di Register Tranfer Level (RTL) sintetizzabile, la cui sintesi e P&R è a discrezione del customer. La seconda è invece è fornita al cliente sotto forma di layout già completamente testato e implementato nella tecnologia di riferimento, utilizzata in quelle parti soggette ad una maggiore complessità implementativa, come le parti di datapath digitali ad alta velocità e la macro analogica.

Per l'azienda risulta fondamentale riuscire a verificare i suoi IP per fornire il miglior servizio possibile ai customer che li acquistano. Da ciò nasce la necessità della prima parte del lavoro di tesi, ovvero lo speed-up di una netlist del circuito digitale, basata su una libreria *technology indipendent*, chiamata Generic Technology (GTECH). Questa tipologia di netlist è spesso fornita ai clienti che richiedono un IP come alternativa ai file sorgente RTL, in modo da proteggerne la proprietà intellettuale. Esse sono utilizzate dai customer per effettuare simulazioni a livello SoC, cioè alto livello, che non richiedono una simulazione completa di tutta l'architettura, ma solamente le parti necessarie a comprendere come il PHY è integrato nella loro architettura.

La crescente complessità nell'architettura di un Ser-Des si traduce in una complessità maggiore della fase di validazione, test e debug dell'IP. Non basta solamente una verifica funzionale della logica digitale, ma risulta fondamentale anche l'integrazione di strumenti che permettano la co-simulazione di firmware e logica digitale ed analizzarne la loro interazione con la macro analogica. Un requisito fondamentale per queste simulazioni è la velocità, è facile intuire infatti in ambito firmware i tempi di simulazione richiesti possano essere elevati e utilizzare modelli analogici complessi non è sostenibile in questi contesti. In questo contesto è possibile utilizzare dei modelli *flat*, che descrivono la parte

analogica in maniera comportamentale, per velocizzare le simulazioni. Il lavoro quindi che si svolgerà in nella seconda parte della tesi è quello di ampliare questi modelli comportamentali, focalizzandosi sullo sviluppo tutti quegli aspetti necessari a supportare la co-simulazione delle procedure di calibrazione. L'obiettivo quindi di questi modelli è quello di fornire un degli strumenti aggiuntivi di supporto al team, accelerando i tempi di debug e rendendo possibile l'analisi di corner-case critici.

Per guidare nella lettura della tesi in seguito è riportata una descrizione di come è stato strutturato l'elaborato:

- il capitolo 2 presenta una panoramica dello stato dell'arte delle architetture SerDes, illustrando le principali strategie adottate sia nella parte digitale sia in quella analogica. L'obiettivo è chiarire sfide e complessità tipiche di queste architetture e fornire una base per gli sviluppi affrontati nei capitoli successivi.

- Il capitolo 3 descrive gli interventi effettuati per ottenere uno speed-up sulle simulazioni della netlist GTECH, spiegando il contesto, le modifiche RTL apportate e i benefici ottenuti in termini di performance simulativa.

- Il capitolo 4 approfondisce il tema delle calibrazioni nella progettazione analogica e descrive la realizzazione dei modelli flat comportamentali. In questo modello si parte da una introduzione su come i meccanismi di calibrazione agiscono in un dispositivo di questo tipo, descrivendo le tecniche implementative e le soluzioni utilizzate. Al termine del capitolo mostra i risultati ottenuti e quanto questi modelli possano portare a miglioramenti effettuando dei confronti con altre soluzioni e modelli.

- Infine, il capitolo 5 di conclusione che riassume i risultati ottenuti e gli sviluppi futuri.

Capitolo 2

Panoramica sui Ser-Des e PCIe

Lo scopo di questo capitolo è fornire una panoramica sul mondo degli High Speed Ser-Des (HS Ser-Des), al fine di chiarire le principali circostanze e sfide che caratterizzano un sistema di trasmissione ad alta velocità. Il capitolo è strutturato in una prima parte dedicata ai concetti generali, seguita dalla descrizione di un'architettura base di trasmettitore e ricevitore. Vengono illustrate le criticità principali e introdotte le motivazioni che giustificano la presenza di specifici blocchi di calibrazione, i cui modelli analogici sono implementati nella seconda parte del presente lavoro di tesi.

2.1 Ser-Des

Un Serializer-Deserializer (Ser-Des) è un blocco che si occupa di serializzare e deserializzare i dati in comunicazioni chip-to-chip ad alta velocità [8]. Nel caso in cui è necessario effettuare una connessione tra due Application Specific Integrated Circuit (ASIC), il Ser-Des interviene effettuando la conversione di dati da parallelo a seriale, la trasmissione sul canale e la riconversione in parallelo per consentire all'ASIC a valle di poter interpretare i dati (figura 2.1). Il ruolo principale di un serdes è quello di minimizzare il numero di pin I/O di un IC, parti che hanno un rilevante impatto al fine di contenere i costi e l'area di Integrated Circuits (ICs), senza compromettere le prestazioni e introdurre colli di bottiglia nella trasmissione dei dati.

In generale, in un Ser-Des è possibile identificare una struttura composta da un numero variabile di *lane* (figura 2.2). Ogni lane rappresenta un canale di trasmissione dati seriale indipendente ed è costituita da una coppia di ricetrasmettitori, al fine di consentire una connessione *Full Duplex*. L'utilizzo di più lane in parallelo consente di aumentare la velocità di trasmissione suddividendo i dati tra più canali fisici.

Un esempio ne è lo standard di comunicazione seriale PCIe che permette di ottenere una connessione di tipo point-to-point per connessioni ad alta velocità.

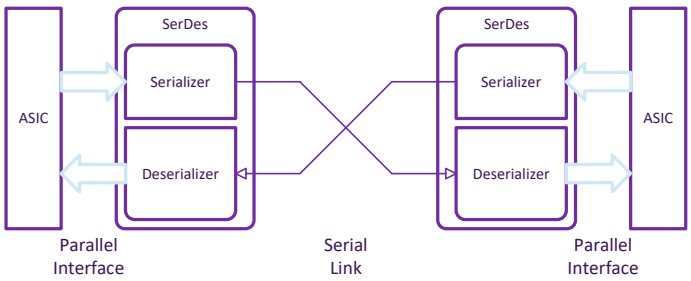


Figura 2.1: Struttura base di un Ser-Des

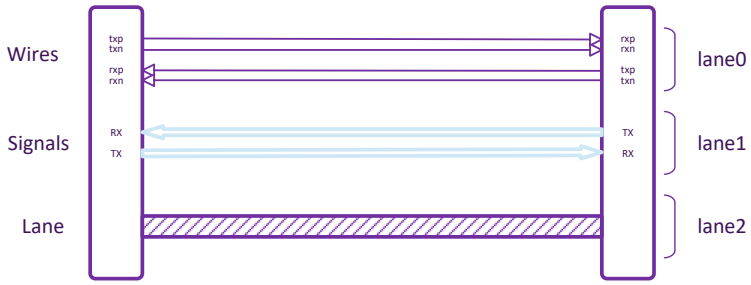


Figura 2.2: ogni lane è composta da un segnale differenziale per i dati da trasmettere e da ricevere, nel complesso sono presenti due flussi di dati, uno per il ricevitore e uno per il trasmettitore.

Nasce come una evoluzione della sua antenata interfaccia parallela Peripheral Component Interconnect (PCI) ed è stata sviluppata dal consorzio PCI-SIG.

Lo standard PCIe è composto da una serie di layer di astrazione [9], come mostrato in figura 2.3:

- **Application Layer:** questo è il layer più esterno, esso è chiamato anche host layer e non è definito dallo standard, ma dipende dal tipo di applicazione. Nello specifico è il blocco che genera le richieste di comunicazione e interpreta le risposte;
- **Transaction Layer:** Questo è il primo vero e proprio la gestisce la configurazione del dispositivo e la trasmissione dei dati da e verso la memoria;
- **Data Link Layer:** questo layer si occupa della gestione e della creazione dei pacchetti, nonché della verifica della loro integrità. In questa fase è possibile applicare tecniche di codifica dei dati, come la Forward Error Correction (FEC), che permettono di migliorare l'affidabilità della trasmissione. Nelle generazioni più recenti, come la Gen6, l'adozione

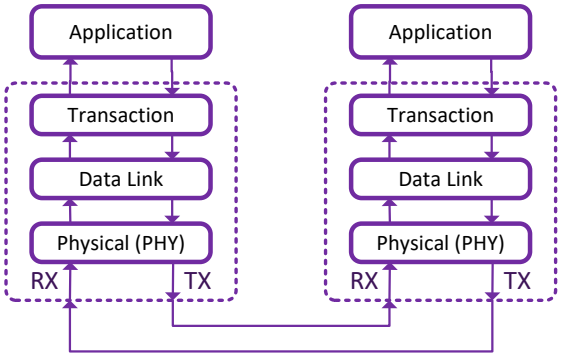


Figura 2.3: PCIe Layers

di meccanismi di correzione degli errori risulta fondamentale per ridurre il tasso di errore. Inoltre, è a questo livello che viene stabilito se un pacchetto contenente errori debba essere ritrasmesso;

- **Phisical layer (PHY):** è il livello più basso del protocollo e consiste in tutte le componenti analogiche e digitali che permettono di trasformare i bit in arrivo dai layer superiori in un segnale elettrico da trasmettere e viceversa riuscire a trasformare il segnale elettrico in informazioni da inviare ai layer superiori.

Per avere chiarezza delle prestazioni di un IP PCIe 6 è possibile analizzare le seguenti specifiche:

- velocità di trasferimento fino a 64 Gbps per singola lane (da considerare come raw bits);
- da x1 a x16 lane, per una velocità di 128GBps in configurazione x16;
- retrocompatibilità con tutte le altre generazioni;
- passaggio dalla modulazione NRZ a PAM-4, per incrementare le prestazioni senza aumento di banda;
- introduzione di codifica FEC per la mitigazione del BER;

Modulazione PAM-4

A differenza delle precedenti generazioni, per poter garantire la velocità di trasmissione richiesta delle specifiche, viene implementata una modulazione Pulse Amplitude Modulation - 4 (PAM-4). Grazie a essa è possibile raddoppiare il bit-rate senza modificare il boud-rate, e quindi mantenendo invariati

i requisiti di banda del canale di trasmissione. Questa codifica permette di poter codificare con un singolo simbolo 2 bit, su quattro livelli di tensione differenti(figura 2.4).

La presenza di 4 livelli differenti di tensione fa si che, a differenza di una codifica NRZ, ci siano ben 3 soglie per la decisione del simbolo.

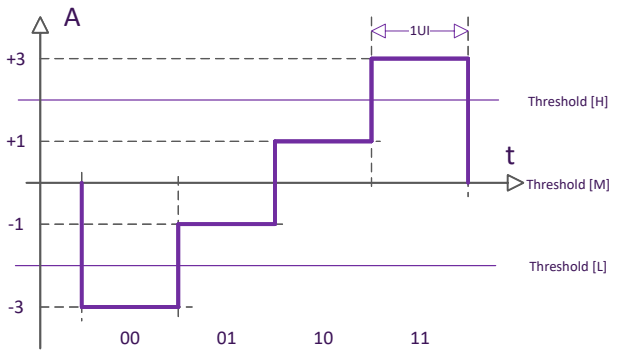


Figura 2.4: Esempio di modulazione PAM-4

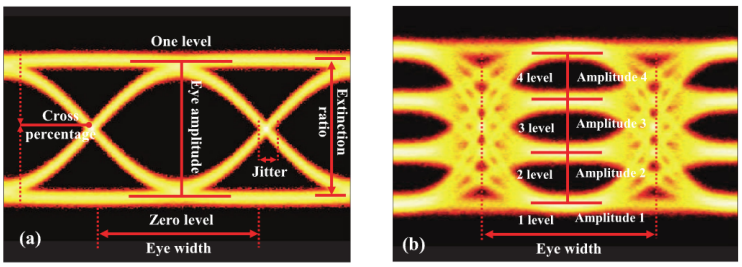


Figura 2.5: a) modulazione NRZ, b) modulazione PAM-4

Il diagramma ad occhio in figura 2.5 mostra chiaramente le criticità e difficoltà che risiedono nell'utilizzo della codifica PAM-4, quella principale riguarda i margini di rumore ridotti e quindi alla prossimità tra le varie soglie. Confrontandolo con la codifica NRZ si nota come l'apertura del diagramma ad occhio sia nettamente diminuita.

L'assegnazione tra l'ampiezza dei simboli da trasmettere ai bit può già essere un primo punto che consente la riduzione degli errori. Una assegnazione come in figura 2.4 presenta infatti una criticità: se ad esempio a causa di un errore il simbolo " $+1$" non è in grado di oltrepassare la soglia "M" ed è riconosciuto come il simbolo " -1" si causa un errore di 2 bit. L'utilizzo di una codifica Gray che implica il cambiamento di un solo bit tra un livello e il successivo, porta ad ottenere un solo bit di errore.

L'architettura di un IP Ser-Des che sfrutta la codifica PAM-4 risulta molto complessa e con numerose tecniche che permettono di ricostruire in maniera accurata il segnale.

2.2 Problematiche di un High Speed Ser-Des

Architetture sempre più complesse usate per ottenere velocità di trasmissione maggiori mettono alla luce una serie di criticità e non idealità durante la progettazione di un High Speed Ser-Des (HS Ser-Des). Ad esempio l'introduzione della codifica PAM-4 porta ad una sensibilità maggiore al rumore, dato che anche una minima variazione sul simbolo porta ad una non corretta discriminazione di ciò che è stato trasmesso. Lo scopo di questa sezione è quello di analizzare le principali sorgenti di errore in un HS Ser-Des per comprendere al meglio come una architettura è sviluppata.

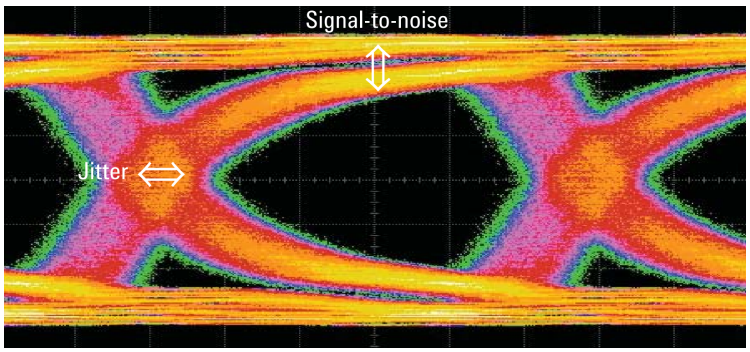


Figura 2.6: Diagramma ad occhio di un segnale affetto da distorsione [10]

Le motivazioni principali risiedono nella presenza di non idealità che derivano da punti di vista differenti. In figura 2.6 è possibile distinguere due principali tipologie di errori: il Signal to Noise Ratio (SNR) (gli errori che influenzano l'ampiezza del simbolo) e il *jitter* (errori dovuti al punto in cui il simbolo è campionato). Entrambi portano al medesimo risultato: il livello di tensione che associato al simbolo nel punto in cui è campionato, è differente da quello ideale, oltrepassando la soglia di discriminazione del simbolo e causando un aumento del Bit Error Rate (BER). Successivamente dono descritte le sorgenti che causano queste non idealità.

2.2.1 Inter Symbol Interfearence

Una importante non idealità su cui bisogna focalizzare l'analisi è quella che riguarda il canale di trasmissione. La non idealità che presenta un generico canale di trasmissione riguarda la sua banda limitata. La presenza di elementi resistivi e reattivi, rende fisicamente impossibile ottenere un canale con una banda sufficientemente grande da essere considerata illimitata.

Come enunciato nei principi di teoria dei segnali se il simbolo presente in ricezione possiede una banda limitata, avrà un supporto temporale illimitato. A titolo di esempio è possibile notare in figura 2.7 come i quattro simbolo

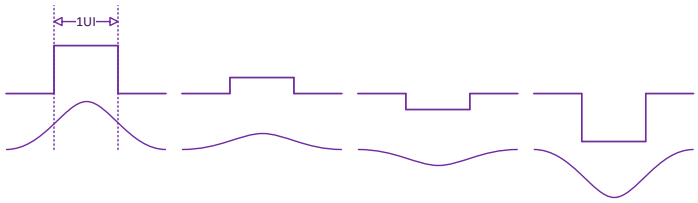


Figura 2.7: Simbolo distorto a causa di banda limitata

di una modulazione PAM-4 ideale risultano in uscita da un canale a banda limitata.

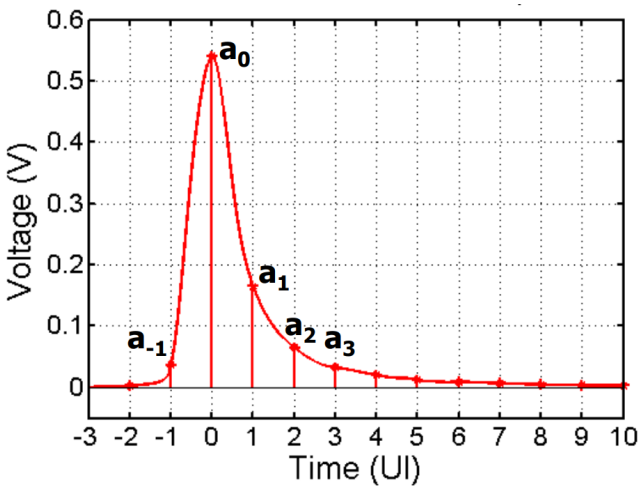


Figura 2.8: Esempio di simbolo in uscita in un canale non ideale

Considerando un simbolo isolato $x[n]$ e trasmettendolo attraverso il canale di trasmissione è possibile notare come anche a distanza superiore ad $1UI$, dove per UI si intende la distanza tra un simbolo e il successivo , siano ancora presenti dei contributi in tensione dovuti a esso (figura 2.8). Questi contributi sono classificati in questo modo:

- **Precursors:** tutti quei contributi che si presentano nei campionamenti precedenti(a_{-1}, a_{-2}, ...);
- **Cursor:** il contributo effettivo dovuto al simbolo attualmente ricevuto, cioe a_0 che si trova a $0UI$;
- **Postcursors:** sono i contributi che andranno ad interferire nei simboli che sono trasmessi dopo il simbolo attuale (a_{n+1}, a_{n+2}, ...).

Da questo si evince poi che trasmettendo una moltitudine di simboli tutti a distanza di $1UI$ i contributi di tutti i simboli continuano a sommarsi rendendo

impossibile determinare il corretto livello del simbolo e generando un errore in ricezione (figura 2.9). Da qui nasce il fenomeno di Inter Symbolic Interfiarence (ISI).

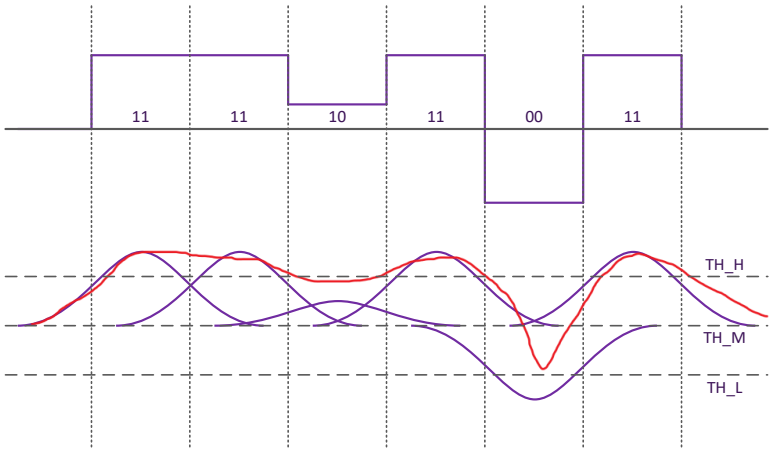


Figura 2.9: Esempio di errore di trasmissione dovuto all'ISI

Tale problematica si considera responsabile di una riduzione del BER e una chiusura del EYE Diagram. In questo caso il livello di ogni simbolo può non rientrare all'interno dei margini di rumore prefissati e quindi può portare all'interpretazione errata. Questo fenomeno emerge maggiormente nelle comunicazioni i cui simboli sono discretizzati con un numero maggiore di livelli e ravvicinati tra loro, come nella codifica PAM-4.

Tramite il diagramma ad occhio in figura 2.10 è possibile notare come l'ISI influenza il valore in cui viene campionato il simbolo.

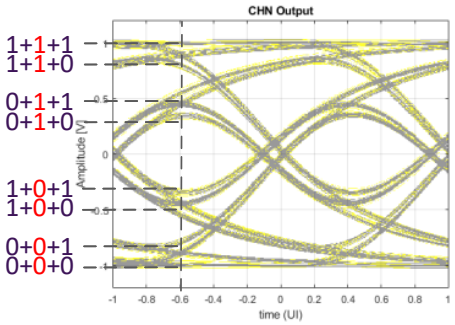


Figura 2.10: Eye Diagram per un canale di trasmissione con ISI

2.2.2 Jitter e Skew

Come stato discusso la motivazione che porta all'analisi del jitter in un sistema di comunicazione ad alta velocità è legato a contenere l'aumento del BER.

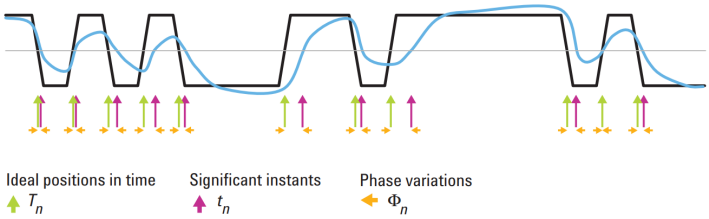


Figura 2.11: Esempio di come il jitter influenza il campionamento del segnale in ricezione

La definizione enuncia che il jitter è definito come variazione di fase rispetto alla sua posizione ideale. Da questo si deduce che il jitter può essere cruciale nel campionamento di un un segnale.

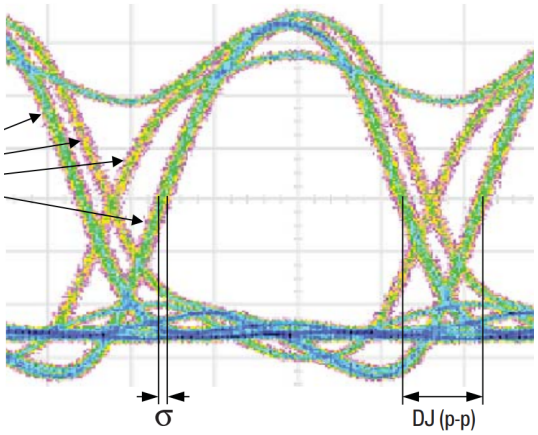


Figura 2.12: Differenza tra jitter casuale e deterministico

Esistono differenti tipologie di jitter, a seconda delle sorgenti e del comportamento, come si evince in figura 2.12:

- **Random Jitter (RJ):** È la tipologia di jitter dovuta a delle sorgenti di rumore casuali, la Probability Density Function (PDF) di questa tipologia di jitter infatti è tipicamente una gaussiana. Le sorgenti di questo rumore sono innumerevoli, possono derivare da phase noise, thermal noise, amplitude noise, ...

  In questa tipologia di jitter risulta impossibile definire una accurata ampiezza picco-picco, ma si utilizzano concetti di media e deviazione standard [10];

- **Deterministic Jitter (DJ):** Questa tipologia di jitter è causata da una sorgenti di anche ampia entità. É definito come deterministico a causa del fatto che, in teoria, conoscendo il sistema sarebbe possibile stimare la sua entità. Esempi di sorgenti di DJ sono interferenze elettromagnetiche, riflessioni sul canale, ISI; come tutte quei fenomeni deterministici che possono portare ad una variazione in ampiezza del segnale, ma che spostano il punto di campionamento del segnale rispetto ad una certa soglia. A sua volta può essere distinto in:
  - **Periodic Jitter (PJ):** quella tipologia di jitter che si ripete in maniera periodica, dovuta ad effetti come *spread spectrum clock* o *feedthrough* del reference clock del Phase Locked Loop (PLL).
  - **Data Dependant Jitter (DDJ):** Soni i contributi di jitter legati ai dati trasmessi, conoscendo:
    - **ISI:** contributo legato alle non idealità del canale di trasmissione e alla sovrapposizione dei simboli
    - **Duty Cycle Distortion (DCD):** in questo caso errori di Duty Cycle portano ad una non corretta durata dei simboli, (esempio sottosezione 2.4.3) .
  - **Bounded Uncorrelated Jitter (BBJ)**: Non dipende direttamente dai dati trasmessi, ma è legato a fenomeni come il crosstalk, generano una variazione in ampiezza che si traduce in una variazione di jitter del segnale dei dati.

2.3 Equalization e Adaptation

In precedenza è stato detto che le non idealità del canale possono portare conseguenze sulla trasmissione delle informazioni. Esistono quindi una serie di strategie utilizzate per mitigare questi effetti, al fine di ottenere una risposta in frequenza costante sul canale.

Le principali strategie introdotte riguardano una serie di filtri analogici e digitali, studiati al fine di ottenere una migliore risposta possibile:

- TX FFE, ovvero un filtro digitale che permette effettuare un boost delle alte frequenze prima di trasmettere il segnale, esso permette di compensare il comportamento passa basso del canale;
- RX CTLE + RX VGA, filtro lineare analogico in ricezione composto che ha un filtro passa alto permette di effettuare un'attenuazione delle basse frequenze, e poi grazie all'amplificatore amplificare a valle tutto il segnale ricevuto per ristabilire i livelli corretti.
- RX FFE, filtro digitale lineare volto alla rimozione di ISI agendo sui termini lineari.

- RX DFE, filtro che permette di compensare anche le non linearità del canale di trasmissione, e che punto agisce in maniera non lineare.

Tutti questi elementi verranno introdotti in seguito. Questi strumenti sono utilizzati al fine di correggere le non idealità lineari e non lineari del canale.

Ognuno di questi elementi presenta dei coefficienti, e il valore di ognuno di questi deve essere scelto in modo da ottenere una risposta in frequenza complessiva che sia costante al variare della frequenza. É possibile dedurre quindi che tutti questi valori dipendano dalla effettiva risposta in frequenza del canale di trasmissione.

In alcuni casi possono esistere dei preset per questi coefficienti, in altri, è necessario implementare determinati algoritmi in hardware, che permettono di valutare il set migliore di coefficienti che portano ad un occhio ben aperto. Queste algoritmi fanno parte delle procedure di adaptation del Ser-Des.

2.4 Phisical Layer: Lane Transmitter

Il trasmettitore è il blocco presente all'interno della lane che si occupa della serializzazione dei dati e della loro conversione nel segnale che verrà effettivamente trasmesso, rappresentando di fatto l'interfaccia diretta con il canale di comunicazione. Esso è composto da una parte digitale, la cui responsabilità principale riguarda l'elaborazione dei dati, e da una parte analogica, che genera il segnale modulato secondo lo schema PAM-4.

Data la complessità e le criticità evidenziate nel capitolo precedente, si rende necessario introdurre alcuni sistemi e tecniche che permettano di mitigare tali problematiche, riducendole al minimo. In figura 2.13 è possibile vedere una tipica implementazione di questo blocco.

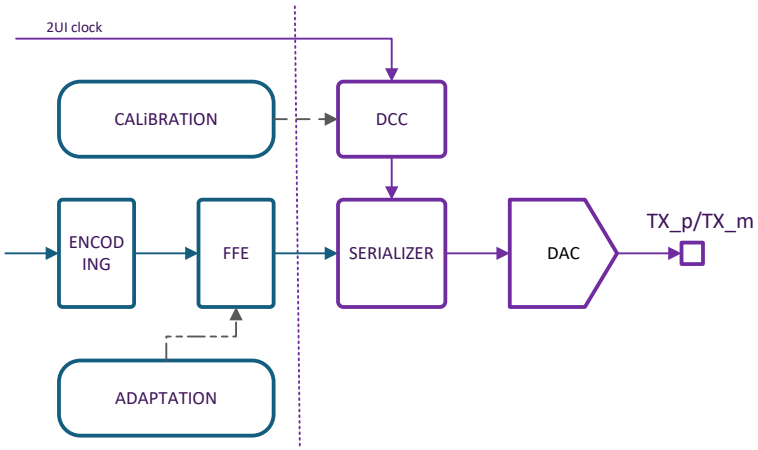


Figura 2.13: Schema a blocco tipico di un trasmettitore

In seguito sono descritti i vari blocchi fondamentali.

2.4.1 Encoder

Lo stadio di encondig può avere un ruolo rilevante durante il progetto di un sistema. Alcune delle motivazioni che richiedono l'introduzione di uno stadio di codifica sono le seguenti:

- garantire il bilanciamento della componente continua (DC).
- assicurare transizioni sufficienti per il clock recovery.
- rilevare e correggere errori.

Esistono diverse tipologie di codifica che presentano differenti vantaggi e svantaggi. Un semplice esempio, usato dalle prime generazioni di PCIe, è

la codifica *8b/10b*, una codifica ridondante, che introduce degli overhead in termini di bit trasmessi. Bisogna precisare che a seconda della generazione gli stadi di codifica possono essere anche presenti in nei layer superiori.

In questo stadio, nel caso di modulazione PAM-4, può essere anche introdotta la *Codifica Gray* che consente di ridurre il numero di transizioni tra i bit di due simboli adiacenti, al fine di ridurre l'errore nel caso in cui un simbolo non venga riconosciuto correttamente.

2.4.2 TX FFE

Come descritto in precedenza, nel trasmettitore è necessario effettuare una predistorsione del segnale in modo da enfatizzare le alte frequenze e garantire in uscita dal canale un segnale con un Eye Diagram che riporta determinate caratteristiche, riuscendo a compensare l'ISI. Il Transmitter (TX) Feed Forward Equalizer (FFE) è il dispositivo utilizzato a questo scopo. La struttura base di questo componente è quella di un filtro FIR digitale, come mostrato in figura 2.14.

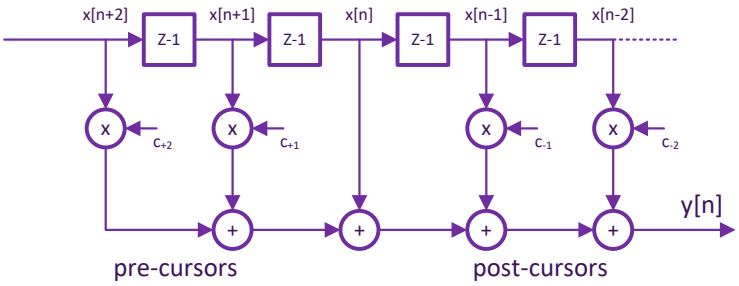


Figura 2.14: Struttura tipica di un fir

L' equazione (2.1) rappresenta la classica equazione di un filtro FIR

$$y[n] = \sum_{i=-N}^{i<M} c_i \cdot x[n+i] \tag{2.1}$$

dove N indica il numero di *Postcursors*, ovvero i taps introdotti per la correzione dell'ISI relativa ai simboli precedentemente inviati, e M indica il numero di *Precursors*, ovvero i taps relativi ai che sono stati inviati dopo il simbolo attuale.

I vari coefficienti c_i saranno calcolati durante la fase di adattamento e dipendono, in maniera intuitiva, da quanto siano elevati i valori residui derivati dai simboli adiacenti a quello attuale.

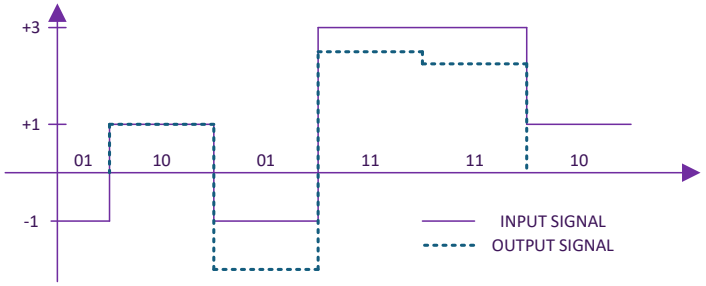


Figura 2.15: Esempio di funzionamento di un FFE

Un esempio di possibile segnale di ingresso e uscita è mostrato in figura 2.15. In questo caso per semplicità è stato considerato un FFE con 2 tap,un postcursor e un precursor. Ad esempio, nel terzo simbolo inviato, "01" la sua ampiezza è stata aumentata visto che i simboli adiacenti possiedono una ampiezza di segno opposto.

2.4.3 Serializer

Il serializer è quella parte del dispositivo che si occupa di una trasformazione dei dati da un formato parallelo a seriale (figura 2.16). In ingresso al serializzatore arrivano i simboli in formato parallelo generati in uscita dal FIR.

Questa serializzazione dei dati è effettuata tramite una serie di multiplexer 2 to 1 che seleziona i dati in relazione al clock di ingresso. Si nota come ogni stadio di multiplexer possiede come selettore una versione divisa del clock di ingresso.

In questa configurazione è possibile notare come un Dyty Cycle non ideale possa portare a delle distorsioni sul segnale trasmesso, in particolare sul multiplexer finale che possiede un clock 2UI.

Infatti quando il clock assume valore alto sono riportati in uscita i simboli "pari" e nel caso opposto quelli di ordine "dispari". Se il duty cycle non assume un valore del 50% esiste la possibilità che non tutti i simboli abbiano una durata di 1UI, non garantendo, in ricezione, il campionamento dei dati nel punto ottimale, generando gli errori di DCD jitter. Nei capitoli successivi verrà discusso una particolare calibrazione con l'obiettivo di mitigare questo problema.

2.4.4 Digital to Analog Converter

In questo caso tramite un Digital to Analog Converter (DAC) è possibile riuscire a convertire il simbolo generato in un livello di tensione da trasmettere

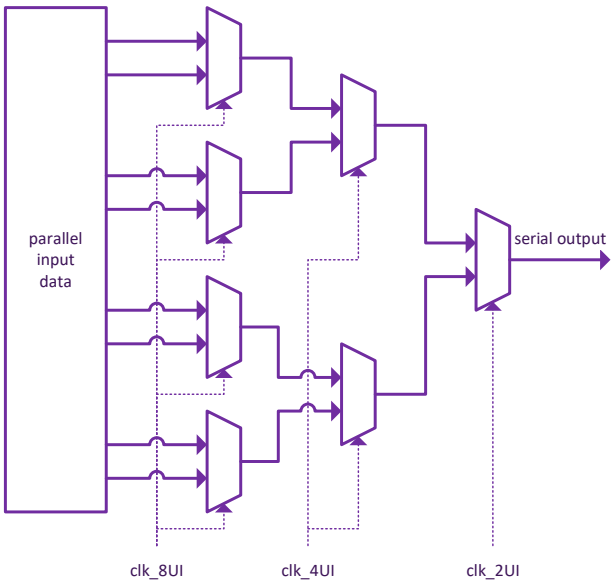


Figura 2.16: Schema di principio di un serializzatore a 8 ingressi

al canale.

Data la presenza dell'equalizzatore FFE i livelli effettivi di segnale generati in uscita non sono solo i quattro possibili ottenibili dalla codifica PAM-4, ma è necessario codificarli con un numero superiore di bit.

Inoltre è necessario ricordare che in uscita ogni lane trasmette il segnale in maniera differenziale.

2.4.5 Duty Cycle Correction

Come anticipato precedentemente il blocco di Duty Cycle Correction (DCC) si occupa di regolare il Duty Cycle del clock necessario al serializzatore, La sua interfaccia possiede un codice di calibrazione che, opportunamente impostato dalla logica digitale, permette una correzione del duty cycle.

2.5 Phisical Layer: Lane Receiver

La struttura base di un ricevitore è solitamente più complicata di quella di un trasmettitore, infatti si ha la necessità di dover ricostruire il segnale. In aggiunta sarà presente anche un blocco di Clock Data Recovery (CDR), il quale permette di allineare il clock di riferimento con i dati.

In figura 2.17 è presente la tipica architettura di un ricevitore, le cui parti verranno dettagliate in seguito:

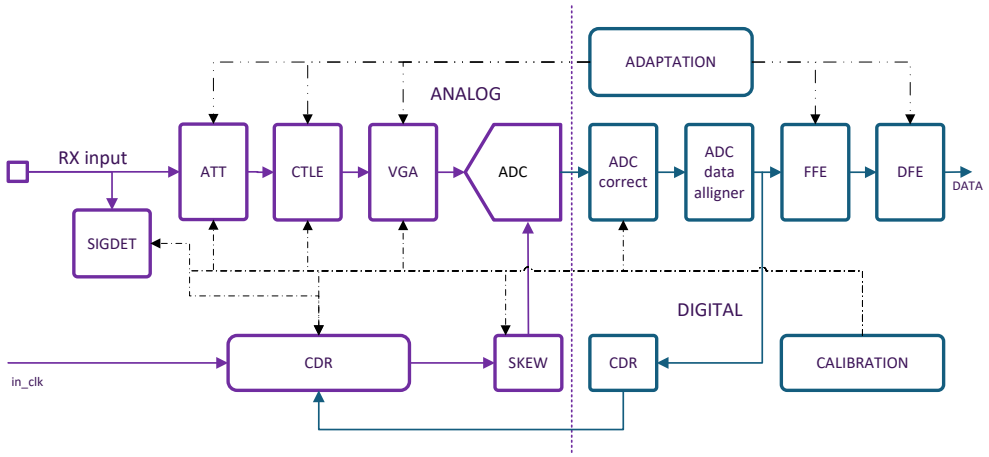


Figura 2.17: Schema di un ricevitore

2.5.1 CTLE

Il Continuos Time Linear Equalizer (CTLE) (nella trattazione chiamato anche HFEQ) è un filtro utilizzato per contrastare l'effetto dovuto al naturale comportamento a banda limitata del canale di comunicazione. Il suo compito principale è quello di amplificare le alte frequenze, che vengono attenuate dal canale. Il comportamento reale di questo filtro è assimilabile a quello di un filtro passa-alto, ovvero attenua le componenti a bassa frequenza per equalizzare a quelle ad alta frequenza.

Dato che il canale non è conosciuto a priori, è necessario effettuare una procedura di *adaptation/equalization* per eseguire il tuning della funzione di trasferimento del filtro.

2.5.2 VGA

Il Variable Gain Amplifier (VGA) è utilizzato all'interno del ricevitore per amplificare il segnale proveniente dal CTLE, infatti il segnale in uscita dal CTLE risulta attenuto, ed è necessario ristabilire un livello di ampiezza adeguato.

2.5.3 ADC

L'Analog to Digital Converter (ADC) presente in questo ricevitore ha la necessità di essere ad elevate prestazioni, data la necessita di avere un elevato throughput. Per questa motivazione l'architettura sfrutta la strategia del *Timing Interleaving*: al fine di massimizzare le prestazione è possibile utilizzare differenti ADC (figura 2.18), che lavorano in modo sfasato e alternato del tempo, riuscendo ad incrementare la frequenza di campionamento complessiva, simulando un ADC complessivo con prestazioni più elevate.

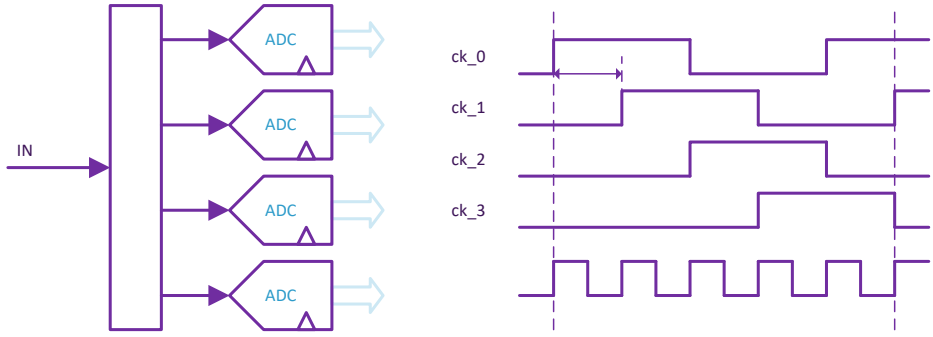


Figura 2.18: Esempio di ADC interleaving

Anche se non sono richieste prestazioni e latenze ridotte per ogni singolo ADC, è comunque presente una certa criticità nei segnali di clock generati, dato che lo sfasamento tra ognuno di essi è legato fortemente alla distanza tra un simbolo e il successivo. Da ciò è possibile dedurre come lo skew tra le varie fasi diventa un elemento determinante, visto che è il responsabile del campionamento dei simboli in ingresso. Avere uno sfasamento non perfettamente ideale causa un campionamento non ottimale dei dati in ingresso, portando ad una non perfetta determinazione dei simboli.

2.5.4 RX FFE

Anche nel ricevitore la presenza di interferenza intersimbolica risulta necessita di essere attenuata tramite un Feed Forward Equalizer (FFE).

Il funzionamento di base è uguale a quello proposto nel TX FFE (sottosezione 2.4.2), solo che in questo caso il calcolo dell'ISI è effettuato considerando i campioni che arrivano dall'ADC.

I valori dei *precursors* e *postcursors* sono valutati durante la fase di adaptation.

Il valore del simbolo attuale viene calcolato compensando il contributo di ISI che ci si aspetta, considerando i simboli precedenti e successivi. Per fare un esempio il contributo $c_{-1} \cdot x[n-1]$ sarà uguale in ampiezza al contributo di ISI presente simbolo attuale, dovuto al simbolo precedentemente inviato, ma avrà segno opposto, e così via con tutti i tap considerati nel'FFE.

È possibile affermare, contrariamente a quanto avviene nel trasmettitore, i simboli non sono più codificati sui 4 livelli tipici del PAM-4, ma su un numero elevato di bit. Questo evidenzia la presenza di alcune problematiche, ovvero che i contributi di correzione di ISI valutati da questo filtro sono calcolati considerando i simboli in ingresso ancora distorti, ovvero contenete del rumore/crosstalk o a sua volta, altri contributi di interferenza intersimbolica[5].

Basti pensare ad esempio che se un simbolo precedente o successivo presenta un contributo di rumore/crosstalk esso sarà anche riportato su tutti gli altri simboli adiacenti.

2.5.5 DFE

Il DFE è un equalizzatore non lineare integrato all'interno del ricevitore, impiegato per consentire la riduzione dell'ISI. La necessità di introdurre un sistema di questo tipo nasce dal fatto che l'utilizzo di un equalizzatore lineare, come l'FFE utilizzato nello stadio precedente, non è sufficiente a compensare i contributi del canale dovuti alla presenza di riflessioni, connettori, ... o altre non idealità presenti nella risposta in frequenza del canale [6].

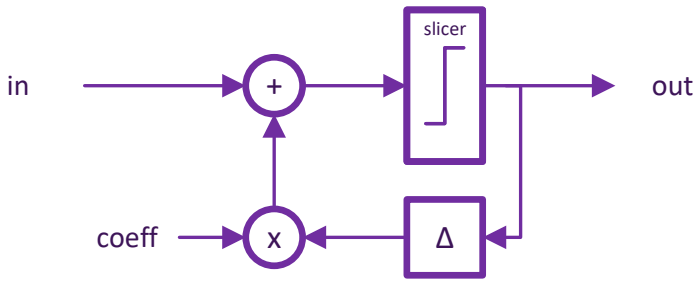


Figura 2.19: Schema di funzionamento di un DFE

Un rilevante vantaggio nell'utilizzare un equalizzatore di questo tipo risiede nella capacità di non introdurre rumore, dato che i dati che arrivano dal feedback sono quelli discretizzati ottenuti a valle del decisore [5] . L'architettura base è mostrata in figura 2.19.

In sostanza il dispositivo in questione agisce modificando i Decision Level (D-Lev), ovvero le soglie di decisione del protocollo PAM-4 in funzione dei dati precedentemente ricevuti quindi delle decisioni prese in precedenza.

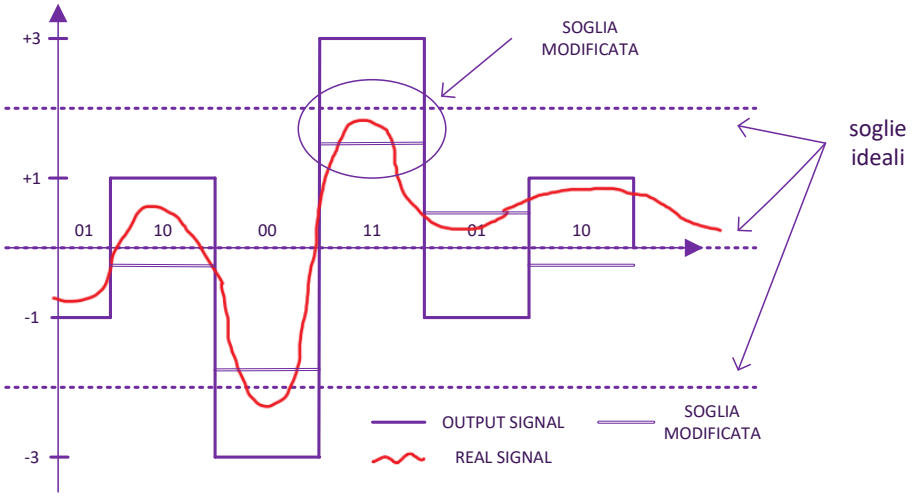


Figura 2.20: Esempio funzionamento DFE

Per chiarire il funzionamento è possibile analizzare il comportamento riportato in figura 2.20: ipotizzando che il simbolo attuale sia ” $+3$” (codifica i bit 11) e il simbolo precedentemente ricevuto sia stato ” -3”, ci si aspetta che il simbolo attuale, abbia un forte contributo di ISI, che agisce riducendo il suo valore effettivo, rendendo impossibile superare la soglia predisposta. Il DFE agisce, tramite l'informazione del simbolo precedentemente ricevuto, introducendo un termine di correzione all'ingresso del decisore che può essere visto come una riduzione delle soglie [7].

Come si nota in figura 2.19, nel percorso è presente un loop, e questo può portare a problematiche nei vincoli di timing, si ricorda infatti che solitamente il DFE è implementato in maniera digitale. Una possibile soluzione, chiamata *unrolled DFE*, consiste nel prevalutare i coefficienti del feedback e poi selezionare quella corretta. In questo modo il contributo di ritardo del sommatore e moltiplicatore nel loop è sostituito dal solo ritardo di selezione del multiplexer, non risultando più un vincolo nell'incremento delle prestazioni. Questa soluzione è possibile grazie alla presenza di solo quattro possibili uscite e alla presenza di un solo TAP, altrimenti prevalutare tutte le altre combinazioni porterebbe ad un impegnativo dispendio di risorse.

Inoltre è possibile notare come il DFE sia quell'elemento nel quale avviene la quantizzazione del segnale. Infatti in ingresso i dati che rappresentano i simboli sono espressi con un numero elevato di bit (dipende dal numero di bit in dell'adc e del FFE) e sono convertiti nei due bit che rappresenta il simbolo

PAM-4.

2.5.6 RX Clock Data Recovery

Una fondamentale parte di una architettura di un ricevitore è quella che si occupa della generazione del clock necessario per campionare i dati. La presenza delle non idealità definite precedentemente rende necessario stabilire con precisione l'istante di tempo in cui ogni simbolo è campionato. Analizzando un classico diagramma ad occhio in ricezione, si nota come anche un minimo disallineamento porta al campionamento dei dati in un punto dove risulta impossibile riuscire a discriminare il valore dei segnali in arrivo. Ancor di più lo è in una modulazione PAM-4 dove ma presenza di soglie ravvicinate tra loro causerebbe un drastico aumento del BER. [4] [2].

In un protocollo asincrono come il PCIe, è necessario creare riuscire a generare il clock di campionamento dei dati a partire dai soli segnali in ingresso. Tutto ciò è effettuato dal blocco di Clock Data Recovery (CDR). Esistono differenti architetture, quella presa in considerazione usa un approccio misto tra il digitale e l'analogico, e che sfrutta l'utilizzo di un Phase Interpolator (PI) e ILO.

A differenza di un sistema come un PLL, che riesce a recuperare il clock da un segnale periodico, la sfida di un CDR è quella di utilizzare un segnale con un generale andamento casuale. Per evitare la problematica di operare con dei dati che presentano lunghe sequenze di uni o zeri, e quindi senza transizioni utili per il recupero del clock, nei layer superiori sono implementate particolari codifiche che permettono di ottenere una distribuzione del segnale casuale.

Phase detector

Come si evince dall'architettura mostrata in figura 2.17 La prima parte del loop del CDR sfrutta i dati in arrivo dall'ADC, per poi andare in ingresso al phase detector. Esso si occupa di generare una informazione riguardante lo sfasamento tra il clock attuale e i dati in ingresso, fornendo una informazione se il clock è in anticipo o ritardo rispetto al punto ottimale in cui campionare i dati.

Un sistema per ottenere questa informazione potrebbe essere quella di sovracampionare il dato, e valutando con precisione dove avviene la transizione. In un HS Ser-Des questa opzione non è implementabile a cause delle elevate frequenze di lavoro. Quello che rimane è sfruttare per queste applicazioni è un *Boude Rate Phase Detector*, che permettono di recuperare l'informazione di fase usando come riferimento un solo campione per ogni simbolo, uno tra questi è il Mueller Muller Phase Detector (MM-PD).

Esso sfrutta le misure di ampiezza dei campioni in arrivo, per cercare di determinare se il simbolo è stato campionato nel suo punto di massimo. Quello che si immagina infatti è che avere dei simboli che presentano un contributo di

interferenza intersimbolica, abbiano un valore di ampiezza non costante, quindi campionando il segnale in un punto differente a quello ideale la sua ampiezza diminuisce.

Come possibile immaginare un sistema di questo tipo non è in grado di funzionare nel caso un cui il segnale di ingresso abbia un comportamento simile a uno ideale, infatti basta pensare che se il simbolo ha un inviluppo quasi costante non è se campionato con un differente sfasamento, il risultato non produce rilevanti differenze e non è possibile capire se anticipare o ridurre lo sfasamento.

Phase Interpolator

Il Phase Interpolator (PI) è il blocco usato per effettuare la correzione di fase. Agisce utilizzando diverse fasi in ingresso sfasate tra di loro e genera un clock in uscita come somma pesata delle varie fasi in ingresso. I pesi di queste fasi sono quelli che permettono di ottenere uno sfasamento del segnale in uscita e possono essere controllati tramite un codice digitale.

Un classico esempio è basato su due fasi di ingresso (componenti I/Q) le quali sono utilizzate per la generazione di una uscita con fase arbitraria ϕ (equazione (2.2)).

$$\begin{aligned} y(t) &= \sin(2\pi f_o t + \varphi) = \\ &= \sin(2\pi f_o t)\cos(\phi) + \cos(2\pi f_o t)\sin(\phi) \end{aligned} \tag{2.2}$$

Come si nota,le ampiezze delle componenti in fase e in quadratura non possono essere decise arbitrariamente. Immaginando di codificare ad esempio ϕ in un codice digitale di controllo, sfruttando la funzione sin e cos si ha la possibilità di ottenere le corrette ampiezze delle due componenti che portano ad una relazione lineare tra il codice e lo sfasamento ottenuto in uscita.

Una ulteriore possibile funzione di mappatura tra sfasamento desiderato e ampiezze delle fasi è quella che sfrutta una approssimazione lineare (figura 2.21) e permette di ridurre la complessità dell'implementazione, ma al consto di ottenere un andamento non lineare tra il codice applicato e lo sfasamento ottenuto.

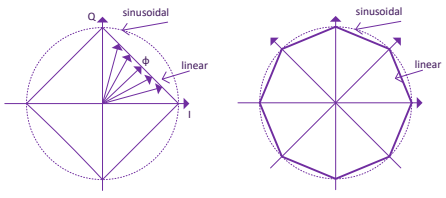


Figura 2.21: Linear interpolator code mapping

In una implementazione reale è utilizzato un numero superiore di fasi, incrementando la complessità ma ottenendo un minor errore utilizzando una valutazione lineare dei pesi.

$$y(t) = \sum_{i=0}^{N-1} A_i \cos\left(\omega t + \frac{2\pi}{N}\right) \tag{2.3}$$

ILO

Come detto, per poter sfruttare il PI è necessario ottenere un numero elevato di fasi equispaziate. Queste fasi sono generate localmente tramite un particolare tipo di oscillatore chiamato, l'Injection Locking Oscillator (ILO).

Un elemento di questo tipo è composto da un oscillatore di riferimento che genera un certo numero di fasi e con una determinata frequenza detta free running frequency (f.r.f.), questo oscillatore può essere un Voltage Controlled Oscillator (VCO), la cui frequenza di lavoro è definita tramite un codice digitale. Al suo ingresso è presente un clock di riferimento, come quello di un PLL. Se le due frequenze sono simili, i due segnali tenderanno a oscillare alla stessa frequenza, attraverso il fenomeno dell'*Injection Locking*.

Questo fenomeno si ritrova non solo in fenomeni elettrici ma anche meccanici. Un esempio è quello di utilizzare prendere 2 pendoli a frequenze leggermente diverse e sfasamenti diversi. Se posti a distanza ravvicinata, a causa di vibrazioni e fenomeni di accoppiamento, tenderanno a convergere alla stessa frequenza di oscillazione e senza alcuno sfasamento. La presenza di questa injection permette di ottenere una sincronizzazione tra due oscillatori.

I vantaggi di una configurazione di questo tipo è rendono più efficace e rapido le fasi di recupero del clock.

L'accoppiamento tra le due frequenze risulta possibile solo nel caso in cui le due frequenze di oscillazione sono ravvicinate. Da questo nasce l'esigenza di effettuare delle calibrazioni che siano in grado di misurare la differenza tra le due frequenze e minimizzarla. Un ulteriore fenomeno si manifesta durante la fase di injection: si genera un errore di sfasamento tra le varie fasi prodotte proporzionale alla differenza tra l'injection frequency e la free running frequency. Si deduce quindi anche la presenza di meccanismi per la correzione dello skew.

Capitolo 3

GTECH Speed-up

Lo scopo di questo capitolo dell'elaborato è quella di chiarire e descrivere le semplificazioni effettuate nella hard macro digitale del PHY, descritta in sezione 2.4 e sezione 2.5, partendo da una introduzione sulla libreria GTECH, sulle metriche utilizzate, come sono state implementate le semplificazioni e i risultati ottenuti.

3.1 Introduzione alla ibreria GTECH

La Generic Technology (GTECH) è una libreria di porte logiche generiche, utilizzata per effettuare la sintesi dell'RTL e generare una netlist. Un possibile esempio della descrizione di queste celle in *Verilog* è mostrato in listato 3.1. Queste celle sono tutte definite a livello comportamentale, risultando completamente prive di informazioni come *strength*, capacità, tempi di propagazione... tipici di una libreria classica standard-cell per la sintesi logica. L'assenza di ritardi di propagazione impedisce l'analisi e l'ottimizzazione del *timing* basata su dati reali da parte del sintetizzatore. In alcuni casi, è possibile modellarle con ritardi unitari, uguali per tutte le celle. Ne consegue che, a differenza di una sintesi basata su una libreria logica standard, non sono presenti file Standard Delay Format (SDF) o Standard Delay Constraint (SDC), contenenti le annotazioni dei vincoli e dei ritardi necessari per la simulazione della netlist.

```
1  module MUX2 ( A, B, S, Z );
2      input   A;
3      input   B;
4      input   S;
5      output  Z;
6
7      assign Z = (S) ? B : A;
8  endmodule
9
10 module NAND2 ( A, B, Z );
11     input   A;
12     input   B;
```

```
13      output  Z;
14
15      assign Z = ~(A & B);
16  endmodule
17
18  module FD2 ( D, CP, CD, Q, QN );
19      input   D;
20      input   CP;
21      input   CD;
22      output  Q;
23      output  QN;
24
25      wire    Q;
26      wire    QN;
27      wire    s_CP;
28      wire    s_D;
29      reg     s_Q;
30
31      assign  s_CP = CP;
32      assign  #(`Thld) s_D = D;
33
34      always @(posedge s_CP or negedge CD) begin
35        if (!CD)
36          s_Q <= #(`Tco) 1'b0;
37        else
38          s_Q <= #(`Tco) s_D;
39      end
40
41      assign Q  =  s_Q;
42      assign QN = ~s_Q;
43  endmodule
```

Listing 3.1: Esempio di porte della libreria GTECH

Come già anticipato, esistono molteplici motivazioni che giustificano l'utilizzo di una netlist di questo tipo. La principale è legata alla necessità, da parte del committente dell'IP, di disporre di un modello simulabile, garantendo al contempo a Synopsys la possibilità di offrire questo servizio senza divulgare informazioni o file coperti da proprietà intellettuale, come l'RTL originale.

3.2 Giustificazioni per la semplificazione della parte DSP

La libreria GTECH è utilizzata come strumento di supporto fornito al customer per facilitare l'integrazione dell'IP nel SoC, analizzandone il comportamento ad alto livello. In tali condizioni non è necessario effettuare delle simulazioni in maniera fedele di ciò che accade a basso livello, ed è possibile simulare scenari di funzionamento ideali. Principalmente l'utilizzo di un canale di trasmissione ideale, quindi non soggetto alle problematiche descritte precedentemente come l'ISI. In questo contesto le tecniche di DSP implementate nel datapath digitale non forniscono alcun vantaggio, ma al contrario, sono elementi che rendono più complessa la simulazione. Un canale ideale infatti implica che i valori dei

coefficienti dei vari filtri siano valutati in maniera tale da rendere trascurabile il loro effetto.

Quanto descritto fin ora è da vedere come l'opportunità che permette la semplificazione della parte DSP dell'architettura durante la sintesi della netlist.

3.3 Metodo di analisi delle performance

Per avere una idea di quali siano le istanze che portano ad un maggiore dispendio di tempo di simulazione e capire su quali elementi è necessario agire, esiste la necessità di effettuare una serie di simulazioni al fine di analizzarne come il tempo di simulazione è suddiviso tra le varie istanze. L'utilizzo di uno strumento come un *Simulation Profiler* diventa indispensabile.

Questo strumento fa parte dell'ambiente utilizzato per effettuare le simulazioni, basato sui seguenti tool di Synopsys:

- **Synopsys VCS:** esso si occupa di compilare i file necessari per la simulazione, l'RTL o la netlist, ed effettua la simulazione producendo il file Fast Signal Database (FSDB) che contiene le informazioni delle forme d'onda ottenute.
- **Synopsys Verdi:** un tool utilizzato per visualizzare o i risultati ottenuti da VCS.

3.3.1 Simulation Profiler

Il *Simulation Profiler* è uno strumento integrato in *Synopsys VCS* utilizzato per l'analisi delle performance di una simulazione logica, sia di RTL che gat-level, al fine di analizzare come il tempo di simulazione è distribuito nelle varie parti.

Durante una simulazione vengono raccolte informazioni sul quanto ogni blocco è utilizzato durante la simulazione, che siano essi `assign`, `always`, `forever`, ...

E' possibile abilitare il *Simulation Profiler* agendo sul comando di *VCS* , come mostrato in listato 3.2.

```
1    vcs -kdb -simprofile ...
```

Listing 3.2: Comando per avilitare il simulation profiler

Al termine della simulazione è poi disponibile un report che presenta tutte le informazioni necessarie al fine di valutare il tempo di simulazione. É possibile visualizzarlo tramite un apposito comando di *Verdi* che permette di avere una visualizzazione più dettagliata (listato 3.3).

```
verdi -profilePath [simprofile dir] -ssf [fsdb dir]
```

Listing 3.3: Comando per visualizzare i risultati del simulation profiler

Ogni report da analizzare è composto da un sommario, che riporta come il tempo totale é suddiviso tra le varie componenti della simulazione, ovvero la simulazione del verilog, le chiamante a funzioni esterne, la generazione dei file di output, il tempo necessario per lo scheduling, elaborazione dei segnali, l'overhead introdotto dal profiler (figura 3.1)...

Questo implica il fatto che se si effettuano delle migliorie per un determinato blocco è presente anche un ulteriore miglioramento dovuto alle altre parti della simulazione, per fare un esempio se alcuni elementi vengono rimossi diminuisce anche il tempo necessario per effettuare il dump delle waveform di uscita.

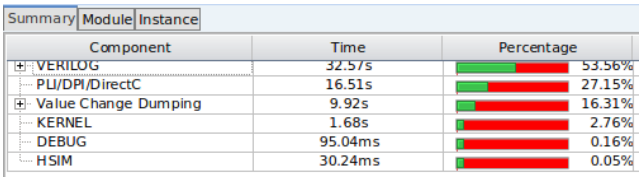


Figura 3.1: Esempio di sommario del Simulation Profiler

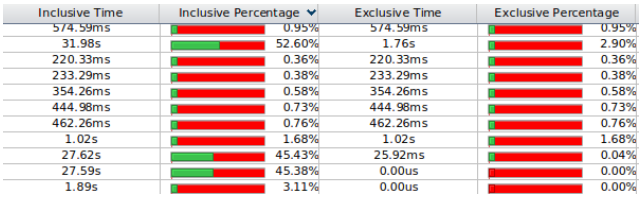
La seconda schermata, figura 3.2, riporta come il tempo di simulazione è impiegato all'interno dei vari moduli istanziati nella gerarchia. Una precisazione da fare riguarda il fatto che i tempi indicati riguardano il solo tempo necessario per la simulazione del Verilog. Nel report sono presenti 4 colonne:

- Inclusive Time/Percentage: indica il tempo in valore assoluto/percentuale per simulare in maniera complessiva quell'istanza e anche tutti i moduli istanziati al suo interno;
- Exclusive Time/Percentage: Indica il tempo necessario per simulare solo il modulo in questione, escludendo tutti i moduli che sono istanziati al suo interno. Utile se è necessario analizzare solamente i processi/-segnali (always, assign) che sono definiti all'interno del modulo stesso, trascurando i moduli istanziati al suo interno.

Nel caso in questione verranno presi come riferimento gli Inclusive Time/-Percentage, dato che la GTECH essendo una netlist, presenta all'interno di ogni modulo delle istanze di altre celle, il che renderebbe non apprezzabile il tempo di simulazione esclusivo di gerarchia superiore.

3.3.2 Testbench utilizzato

Come è possibile intuire, il tempo necessario per completare una simulazione può essere influenzato in modo significativo dal modo in cui il testbench stimola



Inclusive Time	Inclusive Percentage	Exclusive Time	Exclusive Percentage
574.59ms	0.95%	574.59ms	0.95%
31.98s	52.60%	1.76s	2.90%
220.33ms	0.36%	220.33ms	0.36%
233.29ms	0.38%	233.29ms	0.38%
354.26ms	0.58%	354.26ms	0.58%
444.98ms	0.73%	444.98ms	0.73%
462.26ms	0.76%	462.26ms	0.76%
1.02s	1.68%	1.02s	1.68%
27.62s	45.43%	25.92ms	0.04%
27.59s	45.38%	0.00us	0.00%
1.89s	3.11%	0.00us	0.00%

Figura 3.2: Esempio della vista Instance del Simulation Profiler

i segnali del design da simulare. Esistono due aspetti da considerare: il primo riguarda quali segnali vengono effettivamente attivati all'interno del testbench durante la simulazione e quindi quali parti del design vengono stimolate di conseguenza. Il secondo riguarda il *tempo simulato*, ovvero la durata virtuale della simulazione.

Entrambi i due aspetti possono portare infatti ad avere risultati differenti in termini di *tempo di esecuzione* della simulazione (il tempo necessario per elaborare la simulazione), visto che non chiariscono univocamente il carico effettivo per il simulatore.

Ad esempio, un testbench può simulare un intervallo di tempo molto lungo, ma se non genera transizioni sui segnali o non attiva blocchi significativi del design, il carico computazionale sarà ridotto, e quindi anche la durata reale della simulazione sarà breve. Al contrario, in altri casi, una simulazione con un tempo interno breve può richiedere molto tempo per essere completata, a causa dell'elevato numero di segnali coinvolti e delle numerose valutazioni richieste.

Questo comportamento è strettamente legato al funzionamento del simulatore stesso. I simulatori utilizzati per la verifica RTL o netlist sono infatti *event-driven*, e non *time-driven*. Ciò significa che l'elaborazione avviene solo quando si verificano eventi, come un cambiamento di valore su un segnale, piuttosto che scorrere ciclicamente tutti i segnali a ogni passo temporale. Di conseguenza, la complessità della simulazione dipende in larga parte dal numero e dalla frequenza degli eventi generati, e non semplicemente dalla durata del tempo simulato.

Per questo motivo, tutte le simulazioni verranno condotte facendo riferimento a un tempo di simulazione nominale comune, cercando di lavorare in delle condizioni reali, in cui tutti i blocchi fondamentali devono essere attivi che consentano di ottenere dei risultatati simili a quelle delle applicazioni.

Il testbench utilizzato è strutturato in questo modo: una parte preliminare del è esegue la gestione delle procedure di startup, in cui è emulato uno stato di inizializzazione, considerando l'attivazione di tutti i blocchi principali, una seconda parte dove avviene la comunicazione dei dati vera e propria, e infine uno stato di power-off.

Bisogna anche chiarire quali sono i blocchi istanziati all'interno del testbench: oltre ai blocchi principali digitali del trasmettitore e del ricevitore sono

presenti dei modelli analogici, sia del trasmettitore che del ricevitore, come mostrato in figura 3.3.

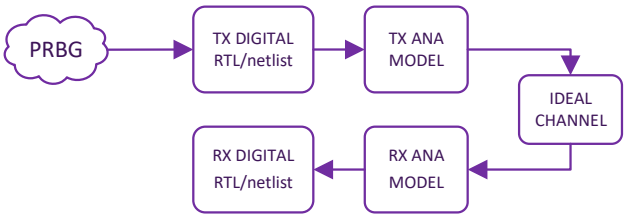


Figura 3.3: Struttura base del testbench

Questo implica che il contributo del tempo di simulazione non sarà da attribuire del tutto alla netlist sintetizzata con la libreria GTECH, ma anche alla presenza di questi modelli analogici.

Tutto questo per ottenere delle informazioni del profiler che siano il più verosimili possibili e che permettano una corretta individuazione dei blocchi responsabili di rallentamenti sul tempo di esecuzione della simulazione. Una precisazione da fare riguarda quindi la durata complessiva della simulazione.

3.3.3 Simulazione di rifermento

Come già detto, prima di procedere con le ottimizzazioni è necessario stabilire una condizione di partenza e ottenere dei risultati del profiler al fine di individuare i blocchi con una maggiore influenza il tempo di esecuzione. I risultati ottenuti sono visibili in figura 3.4 e figura 3.5.

Summary | Module | Instance

Component	Time	Percentage
VERILOG	78.18s	63.81%
PLI/DPI/DirectC	28.51s	23.27%
Value Change Dump...	10.72s	8.75%
KERNEL	4.36s	3.56%
HSIM	676.06ms	0.55%
DEBUG	59.65ms	0.05%
ASSERTION_KERNEL	13.26ms	0.01%

Figura 3.4: Sommario sul tempo di simulazione iniziale della netlist GTECH

Dalla figura 3.4 è possibile ottenere il tempo complessivo di simulazione:

$$T_{ref,tot} = 122.52\,\mathrm{s} \tag{3.1}$$

Questo tempo di simulazione potrebbe sembrare non eccessivo, e si potrebbe pensare che non sia necessario apportare miglioramenti. Tuttavia, si tratta

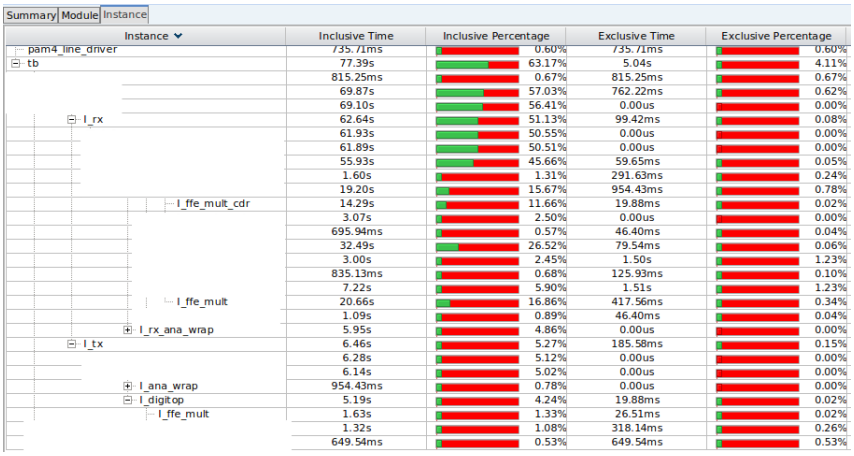


Figura 3.5: Report dettagliato tempi di simulazione GTECH iniziale

solo di una soluzione di riferimento. Nella realtà, infatti, gli algoritmi da testare in queste simulazioni possono avere una durata molto maggiore, anche di diverse ore. Per questo motivo, ottenere dei risultati più rapidamente potrebbe essere particolarmente vantaggioso.

Dalla figura 3.5 si nota come l'elemento critico sia il ricevitore, gli elementi che spiccano sono i due filtri FFE, uno quello utilizzato dal datapath e l'altro quello utilizzato dal sistema di CDR. Entrambi i blocchi sono semplificabili visto che sono legati all'equalizzazione del canale di trasmissione.

Anche se il trasmettitore non è fondamentale per la diminuzione del tempo di simulazione può anch'esso essere semplificato, sempre per quanto riguarda il modulo *tx_ffe*.

In queste analisi si è preferito tenere in considerazione dei valori assoluti dato che, dal momento che una volta che si iniziano ad introdurre dei diverse modifiche può capitare che i valori percentuali possano essere falsati dal fatto che introducendo una modifica e ad esempio riducendo il tempo di simulazione di un blocco, un altro blocco che fino a prima aveva un tempo di simulazione trascurabile poi diventa dominante anche se rispetto alla condizione iniziale non porta alcun miglioramento.

In tabella 3.1 sono riportate le principali instanze semplificabili:

modulo	T_{ref} $[s]$	$\frac{T_{ref}}{T_{ref,tot}}$ $[\%]$
rx_ffe	20.66	16.86
rx_ffe_cdr	14.29	11.66
tx_ffe	1.63	1.33

Tabella 3.1: Tempo iniziale dei blocchi che influenzano la simulazione

3.4 Implementazione delle semplificazioni

A livello di datapath è possibile rappresentare in generale le operazioni svolte, utilizzando come esempio un filtro FIR, il quale ha il funzionamento simile a quello dell'FFE. anche se le istanze su cui si è sviluppata l'ottimizzazione presentano delle implementazioni differenti che rendono molto vantaggioso e andare a rimuover quanto visto.

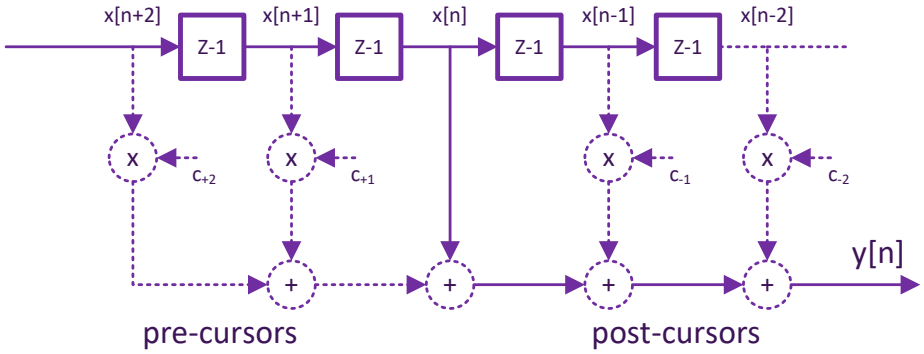


Figura 3.6: Esempio di semplificazione applicata ad un filtro FIR

In figura 3.6 è possibile identificare con linea tratteggiata gli elementi non necessari per la valutazione dell'output del filtro.

Queste modifiche devono essere fatte in modo tale da non influenzare il comportamento del timing e funzionamento del blocco, ma principalmente agendo sulla logica combinatoria e inoltre devono essere inserite in modo tale che non si vada ad intaccare l'RTL sintetizzabile. Per rispettare questi requisiti è stato deciso di inserire una macro che, nel caso in cui venga definita permette, di selezionare il modulo semplificato per la sintesi con la GTECH.

```
1  `ifndef GTECH_OPTIMIZE
2
3      // modulo per la sintesi con GTECH
4      module rx_... (
5          input wire in1,
6          ...
7      );
8          ...
9          //descrizione modulo originale
10     endmodule
11
12 `else
13
14     // modulo per la sintesi con GTECH
15     module rx_... (
```

```
16          input wire in1,
17          ...
18      );
19          ...
20          // modulo semplificato
21      endmodule
22
23  `endif
```

Listing 3.4: Esempio di come sono istanziate le modifiche

3.4.1 RX FFE

Una volta effettuate le semplificazioni ed effettuato una nuova sintesi con la libreria GTECH il report del profiler è mostrato in figura 3.7 e figura 3.8.

E' possibile estrarre l'informazione sul tempo complessivo di simulazione

$$T_{new,tot} = \frac{52.34}{0.6157} = 85.00\,\text{s}$$

con un guadagno in percentuale di

$$= \frac{85.0\,\text{s} - 122.52\,\text{s}}{122.52\,\text{s}} = -30.62\%$$

per quanto riguarda la componente Verilog, il guadagno ottenuto è il seguente:

$$T_{ref} = 20.66\,\text{s}$$

$$T_{new} = 1.61\,\text{s}$$

che si traduce in un guadagno in termini percentuale di:

$$= \frac{1.61\,\text{s} - 20.66\,\text{s}}{122.52\,\text{s}} = -15.54\%$$

Si nota in maniera evidente come in questo caso, una parte considerevole di vantaggio si ottiene anche da tutte le altre componenti di simulazione.

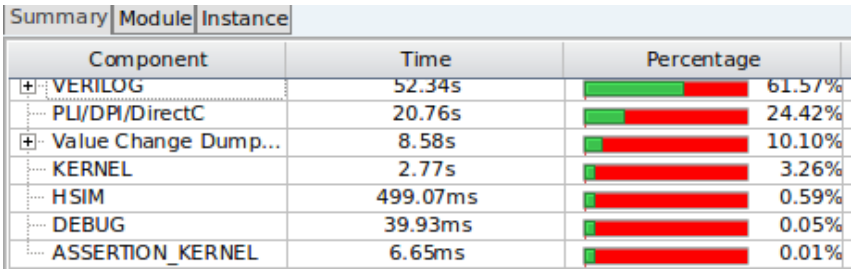


Figura 3.7: Report del tempo di simulazione per *RX_FFE*

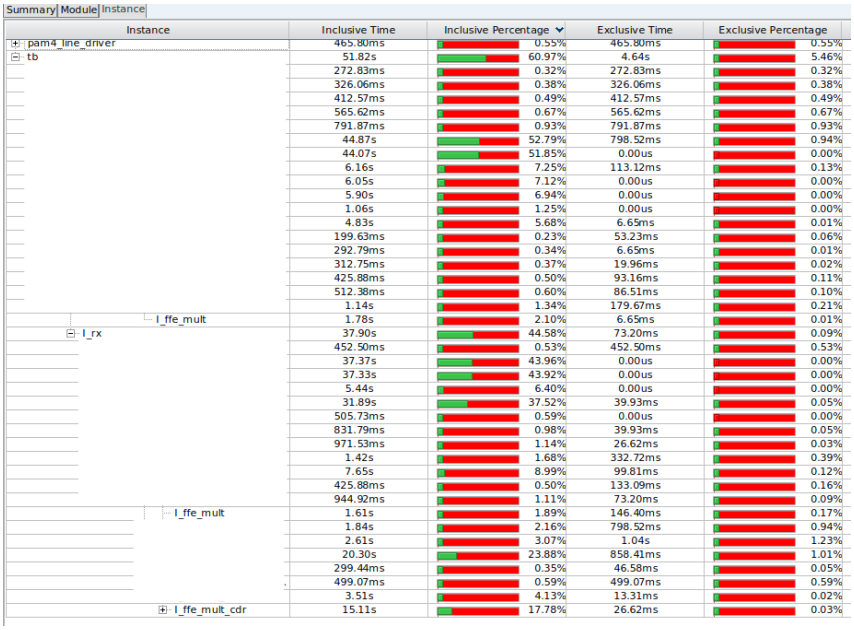


Figura 3.8: Report del tempo di simulazione per *RX_FFE*

3.4.2 RX CDR FFE

Uno dei possibili blocchi in cui è possibile effettuare un miglioramento è una seconda versione del filtro FIR del ricevitore inserito all'interno del blocco di CDR.

Il filtro in questione possiede un numero elevato di elementi combinatori che possono essere esclusi. Purtroppo non è possibile mostrare nel dettaglio quale sia il datapath e come si è andato ad agire, ma dopo aver sintetizzato nuovamente la netlist con libreria GTECH e lanciato la simulazione con lo stesso testbench fatto in precedenza il report ottenuto è mostrato in figura 3.9 e figura 3.10.

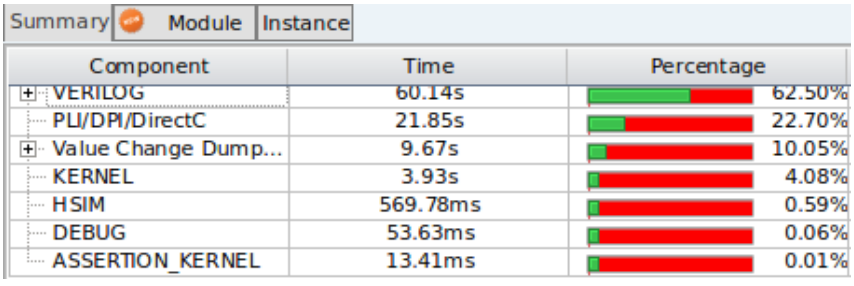


Figura 3.9: Report del tempo di simulazione per *RX_FFE_CDR*

Il tempo complessivo di simulazione risulta:

$$T_{new,tot} = \frac{60.14}{0.6250} = 96.22\,\mathrm{s}$$

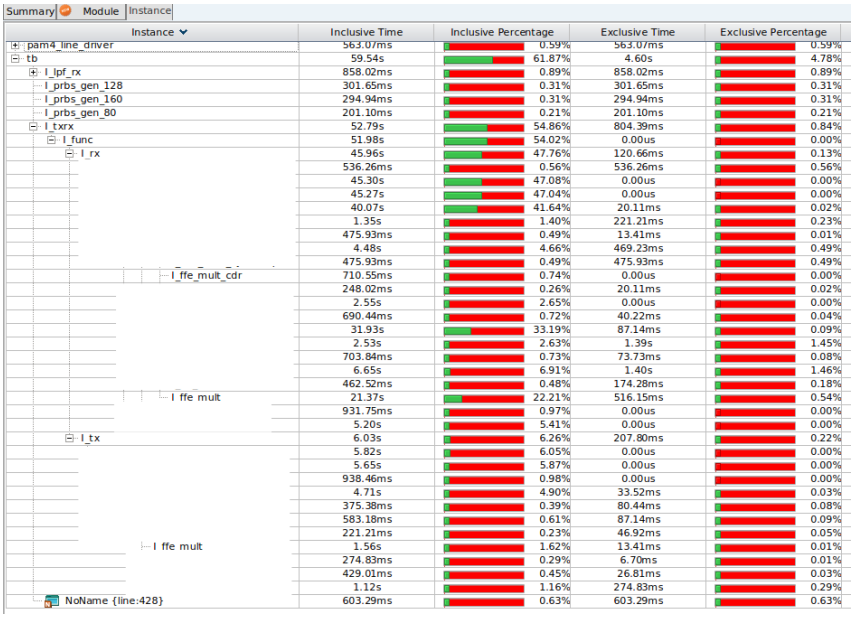


Figura 3.10: Report del tempo di simulazione per *RX_FFE_CDR*

con un guadagno in percentuale di:

$$= \frac{96.22\,\mathrm{s} - 122.5\,\mathrm{s}}{122.5\,\mathrm{s}} = -21.47\%$$

Per quanto riguarda la componente Verilog, il guadagno ottenuto è il seguente:

$$T_{ref} = 14.29\,\mathrm{s}$$

$$T_{new} = 0.71\,\mathrm{s}$$

che in percentuale è espresso come:

$$= \frac{0.71\,\mathrm{s} - 14.29\,\mathrm{s}}{122.52\,\mathrm{s}} = -11.08\%$$

3.5 Analisi risultati ottenuti

Per valutare l'impatto finale delle semplificazioni effettuate è stata sintetizzata una nuova GTECH contenente tutte le modifiche, e successivamente è stata eseguita una simulazione, la quale ha prodotto i report in figura 3.11 e in tabella 3.2.

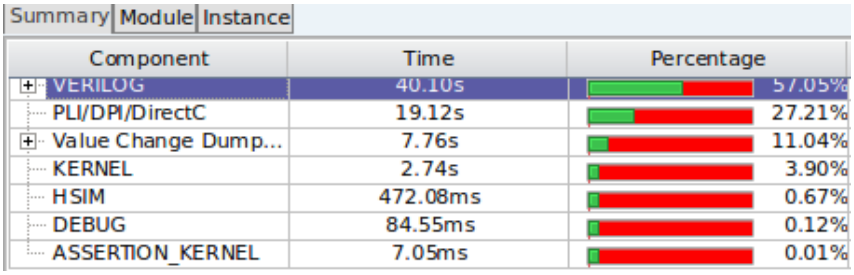


Figura 3.11: FInal Summary

COMPONENT	T_{ref}	T_{new}	$\frac{T_{new}-T_{ref}}{T_{ref,tot}}$
VERILOG	78.18 s	40.10 s	-31.10%
PLI/DPI/DirectC	28.51 s	19.12 s	-7.66%
VCD	10.72 s	7.76 s	-2.41%
KERNEL	4.36 s	2.74 s	-1.3%
TOTAL	122.5 s	70.3 s	-42.5%

Tabella 3.2: Riduzione tempo di simulazione complessivo

Nello specifico la tabella 3.2 evidenzia la differenza tra il valore di riferimento valutato nella simulazione iniziale (T_{ref}) e quello ottenuto in quella finale (T_{new}).

É anche possibile effettuare una analisi più dettagliata vedendo i contributi che forniscono i più evidenti miglioramenti per quanto riguarda i moduli *Verilog* in figura 3.12.

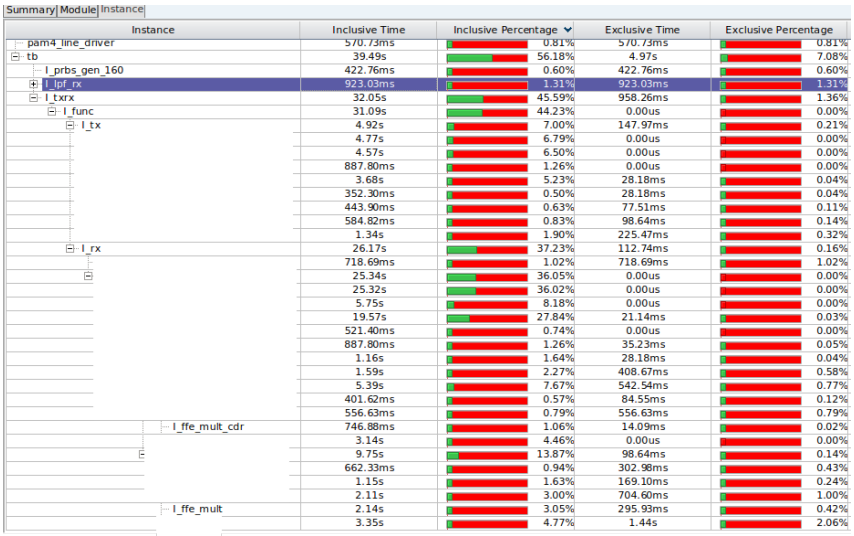


Figura 3.12: Report dettagliato verilog nel caso complessivo

In tabella 3.3 è possibile vedere i miglioramenti ottenuti in termini di tempo di simulazione complessivo per ogni modulo che è stato ottimizzato.

modulo	$T_{ref} \quad [s]$	$T_{new} \quad [s]$	$\%_{new}$
rx_ffe	20.66	2.14	3.05%
rx_ffe_ cdr	14.29	0.75	1.06%
tx_ffe	1.63	< 0.14	$< 0.2\%$

Tabella 3.3: Riduzione tempo di simulazione dovuto ai singoli moduli

I risultati ottenuti riguardano la percentuale di tempo di simulazione ottenuto rispetto al tempo complessivo di simulazione. Per avere una idea più generale sul tempo di simulazione complessivo risparmiato à possibile valutare la percentuale di tempo risparmiato in rapporto al tempo di simulazione iniziale.

$$\Delta\%T_{risp} = \frac{T_{new,el} - T_{ref,el}}{T_{ref,tot}} \tag{3.2}$$

modulo	$\%T_{risp}$
rx_ffe	-15.11%
rx_ffe_ cdr	-11.05%
tx_ffe	-1.3%

Tabella 3.4: Riduzione tempo di simulazione dovuto ai singoli moduli

Capitolo 4

Sviluppo di modelli analogici

In questa parte della trattazione si descrive lo sviluppo di modelli analogici comportamentali, che saranno utilizzati con l'obiettivo di rendere possibili e ottimizzare le co-simulazioni necessarie per analizzare il comportamento del firmware, della logica RTL e della macro analogica durante la fase di calibrazioni degli elementi analogici PHY. Questo capitolo prevede una parte introduttiva sugli algoritmi e meccanismi tipici di una calibrazione, al fine di introdurre il lettore al contesto, una parte successiva riguardante i dettagli implementativi delle calibrazioni nei modelli flat e una parte finale sull'analisi delle prestazioni dei modelli sviluppati.

4.1 Introduzione alle calibrazioni

Prendendo in considerazione la classica architettura del ricevitore descritto nei capitoli precedenti (figura 2.17), si nota come molti dei blocchi analogici posseggano dei segnali di controllo provenienti dal dominio digitale. Il loro scopo è quello della gestione delle calibrazioni, meccanismi utilizzati al fine di mantenere elevate le prestazioni dei circuiti analogici, garantendo il funzionamento del dispositivo durante l'utilizzo in punti di lavoro differenti.

A differenza dei circuiti digitali, la parte analogica risulta molto sensibile alle variazioni Process Voltage Temperature (PVT), che possono generare un'alterazione di grandezze fisiche come tensioni di offset, skew e guadagni, portandole al di fuori del valore nominale e della tolleranza richiesta per il corretto funzionamento del sistema. La variazioni PVT hanno la problematica di non essere costanti durante il ciclo di funzionamento del dispositivo, ad esempio una possibile variazione di temperatura dovuta al cambiamento di power state può portare ad una variazione di tutte le grandezze in gioco, perciò durante il funzionamento del dispositivo gli algoritmi di calibrazione sono eseguiti più volte, sia durante la fase di startup del dispositivo che durante la trasmissione.

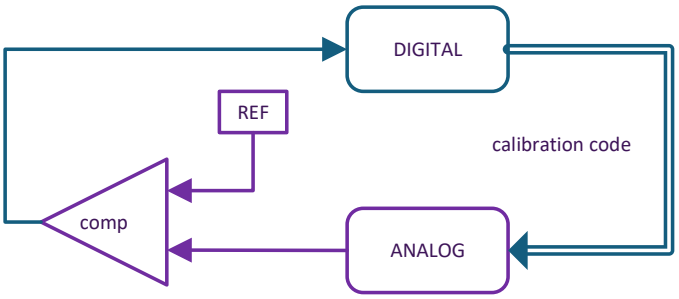


Figura 4.1: Schema di un tipico loop di calibrazione

Oltre all'adozione di tecniche di progettazione volte a mitigare gli effetti di tali variazioni, con le calibrazioni è possibile introdurre un sistema che consenta di controllare il valore della grandezza fisica critica e modificarlo. Un esempio di tale funzionamento è l'utilizzo di un convertitore DAC, impiegato per modificare e correggere la corrente di bias di un circuito analogico, portandola al valore ottimale. L'obiettivo delle procedure di calibrazione diventa quello di individuare il valore ideale del *codice di calibrazione*, in modo da ottenere una perfetta compensazione delle non idealità.

Solitamente gli algoritmi utilizzati per valutare il corretto codice di calibrazione richiedono l'utilizzo di un segnale di feedback proveniente dal dominio analogico, utilizzato per incrementare o decrementare il codice fino ad ottenere una corrispondenza tra il valore da misurare e quello di riferimento. Questo meccanismo funziona tramite un loop di calibrazione, (figura 4.1) che richiede una interazione tra la parte analogica e la parte digitale.

Per chiarire meglio il funzionamento, una procedura di calibrazione che sfrutta un algoritmo lineare per la ricerca del codice ottimale avviene nel seguente modo:

- la calibrazione inizia con un codice di partenza, solitamente un *midcode*, ovvero un valore situato a metà della dinamica del codice;
- un comparatore confronta il valore attuale con quello di riferimento e genera un segnale inviato al dominio digitale usato per la decisione di direzione, ovvero se incrementare o decrementare il codice di calibrazione;
- la Finite State Machine (FSM) che esegue l'algoritmo calcola il nuovo codice di calibrazione in funzione del segnale ricevuto dal comparatore ne aggiorna il valore.
- il codice è nuovamente inviato al blocco analogico per attuare la modifica della grandezza da calibrare;

- il loop è poi ripetuto fino a quando la grandezza da calibrare non raggiunge il valore di riferimento.

Per evitare di introdurre ulteriori errori lungo il loop viene utilizzato un particolare comparatore ad alte prestazioni, con la peculiarità di essere progettato in modo tale da compensare internamente le proprie variazioni PVT,rendendole trascurabili rispetto a quelle degli elementi da calibrare. Data la sua complessità, esso è riutilizzato in molte calibrazioni, rendendo il suo segnale di uscita condiviso tra i vari blocchi digitali.

Come si evince dall'algoritmo mostrato, non si ha la possibilità di effettuare le simulazioni riguardanti le calibrazioni senza coinvolgere entrambi i domini analogici e digitali, rendendo le calibrazioni una fase ostica da simulare dal punto di vista di complessità computazionale nel caso in cui vengano utilizzati modelli complessi e completi della parte analogica.

4.2 Modelli analogici flat con calibrazioni

Un metodo pe la simulazione della parte analogica riguarda lo sviluppo di modelli comportamentali (*flat models*) per la *Macro Analogica*, chiamati `rx_ana_flat` e `tx_ana_flat`, rispettivamente per il ricevitore e trasmettitore. Questi sono dei modelli sviluppati come moduli Verilog e che possono essere simulati all'interno dei simulatori logici per l'RTL. Essi si emulano il comportamento della parte analogica, in modo da garantire una interazione con la parte digitale, quando necessario. Non è necessario riprodurre in modo fedele il funzionamento interno della parte analogica, ma si presta attenzione principalmente ai segnali utilizzati per l'interfacciamento con la parte digitale.

Inizialmente questi modelli non supportavano la possibilità di poter effettuare le calibrazioni, ma emulavano solamente gli elementi principali del datapath analogico. I modelli sviluppati in questo lavoro di tesi saranno istanziati all'interno per favorire una verifica high coverage della del firmware e dell'RTL durante la fase di calibrazione del dispositivo.

Come in generale per i modelli analogici flat, anche per la parte delle calibrazioni lo scopo di questi modelli è di verificare il corretto funzionamento dell'interfaccia tra la parte analogica e digitale, alcuni esempi sono:

- verificare che le tensioni di alimentazione siano abilitate correttamente, ad esempio se durante cambiamenti di power state le tensioni non sono gestite correttamente si potrebbe verificare la perdita dei codici presenti negli elementi sequenziali come registri o latch;

- i codici di calibrazione sia correttamente campionati nei latch, per controllare che i clock siano correttamente forniti agli elementi sequenziali;

- i segnali di enable siano correttamente settati;

- le polarità dei loop di calibrazione siano corretta;
- i segnali di controllo dei moduli analogici, come comparatori, multiplexer, ADC, ... siano corretti.

Questo comporta che per effettuare una calibrazione non sia necessario attuare le correzioni sul modello e sul datapath analogico, ovvero non è necessario che il codice di calibrazione agisca effettivamente sui blocchi analogici, a differenza di alcune eccezioni che saranno dettagliate in seguito.

Questo approccio permette di ottenere calibrazioni indipendenti le une dalle altre, permettendo di poter simulare una sola simulazione e senza che i risultati di ogni calibrazione influenzi le altre.

Il funzionamento vero e proprio del blocco da calibrare non è simulato, ma solamente come la grandezza da calibrare del blocco agisce in funzione dei segnali di stimoli delle calibrazioni, generando i segnali digitali in caso di calibrazione. All'interno dei modelli infatti vengono riconosciute le combinazioni di segnali necessarie ad attuare le calibrazioni e dopo di che il corrispondente blocco è riconosciuto e abilitato, generando l'uscita necessaria alla calibrazione.

4.2.1 Implementazione generale modelli

I seguenti modelli relativi alle calibrazioni sono implementati in Verilog. Le grandezze analogiche sono implementate sotto forma di variabili di tipo `real`, ovvero una variabile floating point a doppia precisione. La correzione è effettuata modificando il valore di queste grandezze fittizie.

Le variazioni PVT delle grandezze da calibrare sono inizializzate in pseudo-casuale, ciò può essere fatto sfruttando delle tipiche funzioni verilog necessarie per la generazione di numeri casuali, le funzioni in questione sono `urandom()` scalati opportunamente in uno specifico range definito per ogni calibrazione, cercando di replicare delle variabili casuali indipendenti.

I modelli vengono generati prendendo come riferimento due fonti differenti: la prima è un documento che contiene tutte le specifiche funzionali di come devono essere progettati i circuiti di calibrazione, la seconda fonte è composta dagli schematici analogici, usati per avere una visuale maggiormente dettagliata su alcuni elementi da andare a modellare, come i le tensioni di alimentazioni specifiche utilizzate per gli elementi di memoria o i segnali di enable effettivamente usati.

Per essere in grado di avere dei valori che siano indipendenti da una simulazione alla successiva è necessario inizializzare un seed ogni qual volta è necessario lanciare *VCS*. Ad esempio è possibile utilizzare come seed la data attuale. Il range di variazione di queste variabili è cruciale per garantire un corretto funzionamento del blocco si di calibrazione. Infatti se la variazione è maggiore della correzione che può essere attuata possono essere riscontrati dei particolari errori di convergenza.

Dato che non è sempre necessario avviare le procedure di calibrazione, esse possono essere attivate o disattivate attraverso direttive di precompilazione (`‘ifdef` e `’define`). Se la macro `‘ENABLE_FLAT_MODEL_CALIB` è definita, i blocchi relativi alle calibrazioni sono istanziati all'interno del modello flat.

4.3 Simulazioni per calibrazioni

Per le simulazioni di queste calibrazioni sono stati adattati alcuni testbench preesistenti. In generale il compito di questo testbench è quello di asserire i segnali necessari alla configurazione e alla gestione delle FSM che eseguono le calibrazioni, emulando il seguente flusso:

- settare i relativi registri di configurazione per ogni calibrazione
- configurare le modalità di calibrazione (coarse, fine o mission mode)
- asserire il segnale di start
- aspettare in polling il segnale di done
- verificare se siano asseriti segnali di errore
- procedere allo stesso modo con le altre calibrazioni

Il testbech utilizzato è basato sullo stesso utilizzato nel capitolo precedente, ma utilizzando l'RTL e introducendo dei test per ogni calibrazione al fine di analizzarne il comportamento. Inoltre ogni simulazione contiene comunque l'abilitazione del Simulation Profiler, usato per verificare le prestazioni dei modelli sviluppati.

4.3.1 Verifica dei modelli

Una volta effettuate le simulazioni tramite il testbench descritto in precedenza per analizzare il funzionamento base e valutarne le prestazioni simulative, i vari modelli sviluppati sono stati inviati al team di verification per i test di verifica funzionale.

Ogni modello è stato importato all'interno dell'ambiente di verifica e per verificarne il corretto funzionamento. Per ogni calibrazione esiste un test, sviluppato in precedenza, utilizzato per la verifica del comportamento del modello non comportamentale già presente. Questi test sono stati riutilizzati e opportunamente modificati al fine di verificare il comportamento dei modelli flat, per vedere se si comportino in modo atteso. Ad esempio all'interno di questi modelli sono presenti dei checker che si occupano di monitorare il funzionamento di determinate grandezze fisiche come tensioni, correnti, o anche codici di calibrazione.

4.4 Calibrazioni RX

In figura 4.2 è possibile analizzare una panoramica sul modello del ricevitore al termine dello sviluppo dei modelli. Lo schematico proposto mostra l'integrazione degli elementi sviluppati nel modello già esistente. Risulta evidente come molti dei moduli istanziati (quelli all'interno del modulo `rx_ana_cal_top`) siano completamente indipendenti dal datapath necessario per la generazione dei paralleli inviati poi al dominio digitale. In questo modo l'introduzione degli errori PVT necessari per le calibrazioni non influenza in alcun modo i dati ricevuti, e rende anche le calibrazioni del tutto indipendenti le une dalle altre. Una nota importante riguarda la presenza di un unico segnale di feedback, il quale è condiviso da tutti i moduli sviluppati. Da ciò nasce l'esigenza di generare per ogni calibrazione un segnale di controllo, da utilizzare per la selezione del feedback corrispondente nel multiplexer di uscita.

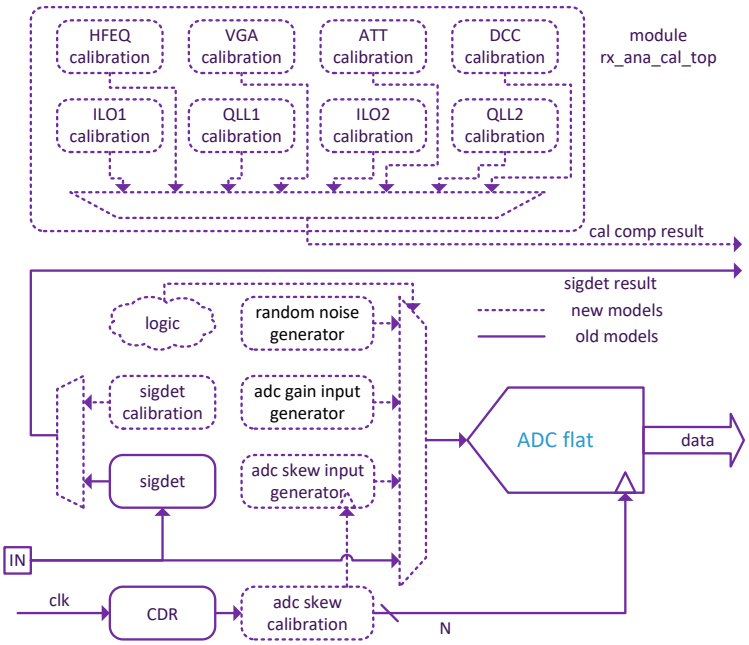


Figura 4.2: Rappresentazione semplificata del modello *rx_ana_flat*

Quanto descritto precedentemente non risulta valido per le calibrazioni concerni l'Analog to Digital Converter (ADC). In questo caso, il dato digitale ottenuto dal convertitore è utilizzato direttamente come feedback per determinare i codici di calibrazione, rendendo indispensabile l'introduzione delle variazioni PVT al suo interno. Tali calibrazioni richiedono la generazione di alcuni segnali e pattern di ingresso, introducendo in questo modo un multiplexer per la selezione dell'apposito ingresso attraverso specifici segnali di

controllo. I vantaggi e le limitazioni di questo approccio saranno dettagliate successivamente.

Nello specifico in figura 4.3 e' possibile vedere come ogni calibrazione possiede tutti i segnali di controllo necessari e sono usati per il segnale di detection.

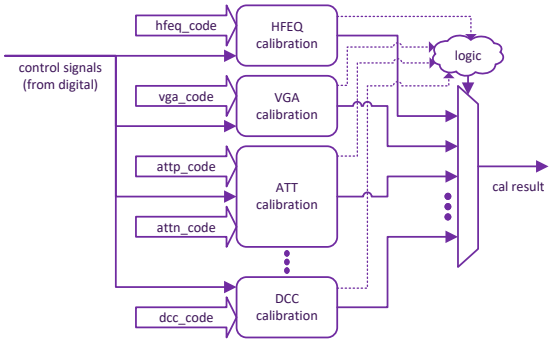


Figura 4.3: Multiplexer per il risultato di calibrazione

4.4.1 RX CTLE, VGA, SIGDET calibration

I modelli di calibrazione di RX CTLE, VGA, SIGDET hanno una struttura molto simile, e sono tutti volti alla calibrazione di una grandezza di offset. Questo perché che come definito nel capitolo introduttivo, questi modelli non emulano il comportamento del funzionamento del blocco, e quindi, dato che la grandezza da calibrare è la medesima, possano essere realizzati sfruttando lo stesso principio. Per descrivere il funzionamento di terrà in considerazione la calibrazione del Continuos Time Linear Equalizer (CTLE).

La calibrazione in questione è implementata con lo scopo di ridurre l'offset in uscita. Nell'implementazione effettiva è possibile misurare questo valore di offset in uscita ed effettuare la misura tramite il calibration comparator. Il funzionamento è il medesimo analizzato nella sezione precedente.

L'algoritmo di ricerca del codice è di tipo lineare. Il codice è inizializzato ad un valore intermedio, detto *midcode* e il suo valore è incrementato o decrementato di un certo valore, a seconda del valore di in uscita dal comparatore. Per spiegare il funzionamento è possibile procedere con il seguente esempio: se il valore di offset è superiore al valore di riferimento il codice è ridotto in modo tale da ridurre l'offset, si procede quindi con il numero necessario di step utili per correggere l'errore. Una volta raggiunto il valore di riferimento il codice inizia a oscillare attorno al valore di riferimento. Una possibile analogia a questo algoritmo può essere trovata nel principio di funzionamento di un convertitore ADC a inseguimento. Lo schema a blocchi del modulo Verilog usato per emulare questa simulazione è disponibile in figura 4.4.

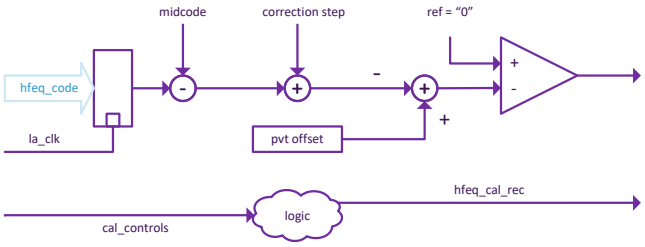


Figura 4.4: Schema di funzionamento modulo HFEQ calibration

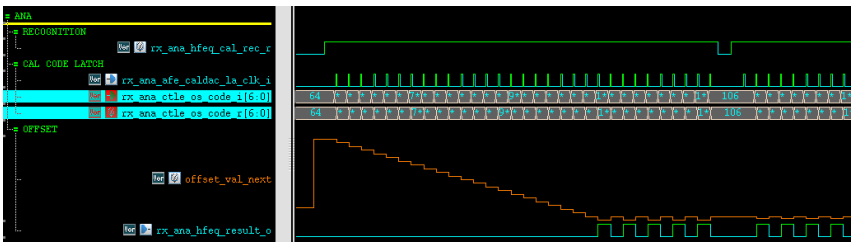


Figura 4.5: Risulatati simulazioni HFEQ calibration

Il sistema di correzione dell'offset è stato impostato come nell'equazione (4.1), se il codice si trova al *MIDCODE* non è presente alcuna correzione, invece un incremento del codice di calibrazione porta ad una riduzione dell'offset.

$$V_{off} = V_{off,pvt} - [CAL\_CODE - MIDCODE] \cdot \delta_{off} \tag{4.1}$$

Per rendere un'idea del funzionamento in figura 4.5 è mostrata una simulazione che mostra come il valore di offset si corregge e tende al valore di riferimento.

Se il codice di calibrazione satura verso il massimo o il minimo valore la macchina a stati asserisce un segnale di errore.

Come già accennato una calibrazione di questo tipo sfrutta il calibraiton comparator e quindi il segnale di uscita è condiviso con altre calibrazione. Una parte importante risulta tutta la parte si logica che acquisisce in ingresso i segnali provenienti dal digitale e in arrivo all'analogica e verifica che i valori attesi siano rispettati. Se tutte le condizioni sono rispettate allora la calibrazione è correttamente riconosciuta e il risultato del comparatore è inviato in uscita. Se le condizioni non sono rispettate la FSM non sarà in grado di effettuare la calibrazione correttamente e il segnale di errore è asserito.

4.4.2 RX ATT calibration

Il segnale in uscita all'ATT è di tipo differenziale, questo rende necessario effettuare la calibrazione dei segnali di offset ad entrambe le uscite. Anche in

questo caso il funzionamento è molto simile a quello precedente. Lo schematico è riportato in figura 4.6.

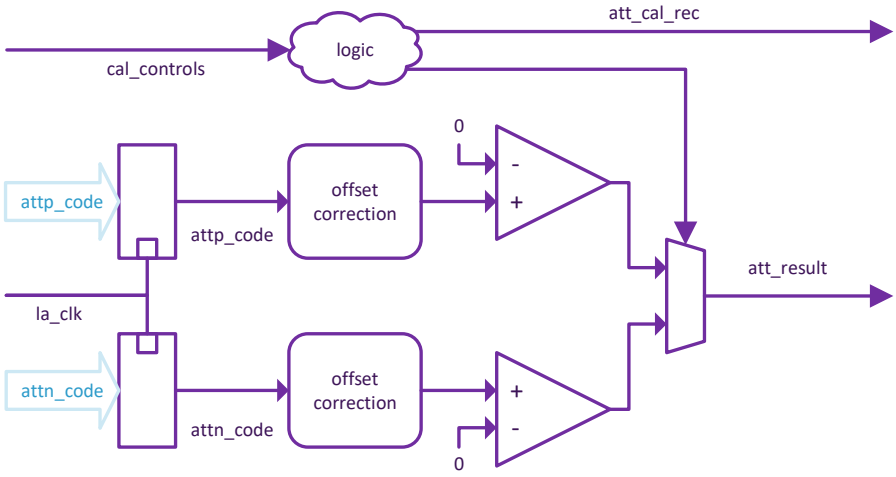


Figura 4.6: Schema di funzionamento modulo ATT calibration

4.4.3 RX DCC calibration

La seguente calibrazione è sviluppata al fine di correggere il Duty Cycle delle varie fasi del segnale di clock generato dall'ILO, nel CDR. Questa calibrazione è effettuata in maniera comportamentale, si è deciso di generare in maniera casuale i valori di δ_c di ogni fase. La presenza di un numero elevato di fasi, e quindi la necessità di un numero elevato di codici porta alla presenza di una bus. Nel modello si è quindi gestita la decodifica dei segnali, verificando in questo modo il corretto funzionamento. Oltre a queste caratteristiche, lo schema di calibrazione è del tutto simile a quelli descritti precedentemente.

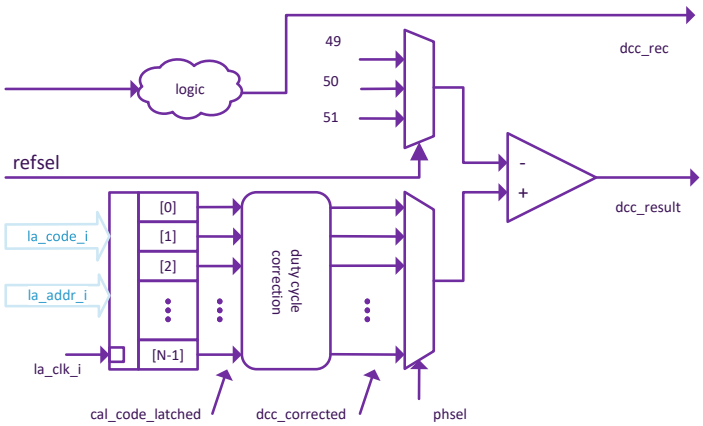


Figura 4.7: Schema di funzionamento modulo RX DCC calibration

4.4.4 RX ADC calibrations

L'architettura di calibrazione implementata in questo Ser-Des, utilizzata per identificare e fornire un feedback per la correzione degli errori PVT provenienti dall'ADC, è digitale. Senza entrare nel dettaglio, i dati digitali forniti in uscita dal convertitore sono utilizzati per l'esecuzione di algoritmi che permettono di individuare le discrepanze tra i valori ideali dei dati dai valori effettivi, potendo così intervenire con delle correzioni, siano esse nel dominio digitale o analogico.

Per questo motivo, diversamente dalle altre calibrazioni, non è possibile modellare la grandezza fisica da calibrare in modo indipendente dal datapath, ma risulta necessario introdurre in esso gli errori di calibrazione. Per questa motivazione si sfrutta il modello di convertitore analogico-digitale preesistente nel modello flat del ricevitore, definendo al suo interno gli errori PVT. Un ADC ad alta velocità realizzato tramite la tecnica di interleaving possiede più di un Successive Approximation Register (SAR) ADC al suo interno e ognuno di essi possiede una propria caratteristica di conversione, la quale può soffrire di non idealità. Gli errori PVT prima citati sono di due tipi: quelli che introducono una variazione di offset e pendenza all'interno della caratteristica di ogni convertitore, e quelli che introducono un errore di timing e campionamento dei vari ADC.

Data la maggiore complessità del modello in questione, queste calibrazione richiedono di monitorare in modo più pedante la presenza di elementi che potrebbero portare ad una riduzione e rallentamento nell'esecuzione delle simulazioni.

Nelle sottosezioni 4.4.5 to 4.4.7 è presente la descrizione di ogni calibrazione implementata.

4.4.5 RX ADC offset

In precedenza è stato accennato che le calibrazioni ADC offset e gain presentano un meccanismo di correzione dell'offset effettuato in digitale. Per quanto riguarda l'offset questo meccanismo è progettato in maniera tale da garantire una correzione inferiore ad 1 LSB.

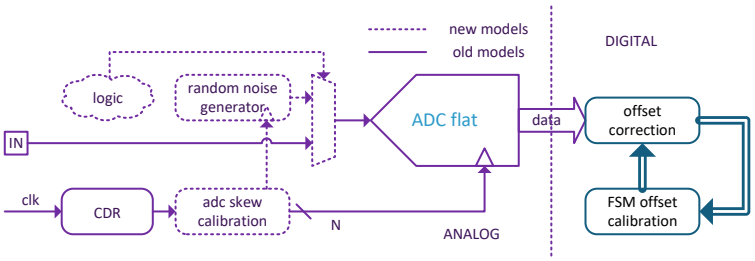


Figura 4.8: Schema ADC offset calibration

Dallo schema in figura 4.8 si nota come la calibrazione in questione non abbia la necessità di introdurre complessi blocchi analogici per la calibrazione, dato che la correzione è effettuata totalmente in digitale. Bisogna solo prestare attenzione al segnale in ingresso all'ADC per la generazione dei dati necessari per l'esecuzione dell'algoritmo.

Nell'algoritmo di calibrazione si applica la strategia del *dithering* attraverso il calcolo del valore medio dei dati per valutare anche gli offset sub-LSB. In un sistema ideale tutto ciò non risulta possibile a causa dell'assenza del rumore, e ciò non porterebbe ad una efficace valutazione dell'offset. Per ovviare a questo problema come segnale di ingresso dell'ADC è utilizzato un rumore con una distribuzione pseudocasuale con valore medio nullo e ampiezza 1 LSB.

```
1     always @(posedge clk ) begin
2     noise <= ((`RANDOM % 11)/10.0 - 0.5) * LSB; //random noise (+-0.5LSB)
3   end
```

Per fare un esempio, se nella seguente caratteristica ideale si inserisce un rumore con valor medio 1.25 ci saranno 75% di probabilità che il valore sia 1 e 25% che sia 2. Effettuando quindi il valor medio del codice di uscita si ottiene proprio il valore richiesto 1.25.

$$code = 0.75 \cdot 1 + 0.25 \cdot 2 = 1.25 \tag{4.2}$$

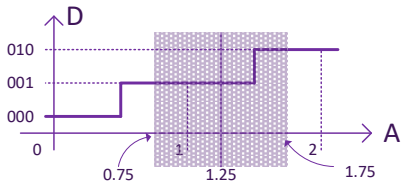


Figura 4.9: Ditherinig sulla caratteristica dell'ADC

4.4.6 RX ADC Gain

Anche in questo caso si nota come l'implementazione proposta in figura 4.10 sia esente da blocchi analogici per la correzione del guadagno, risultando tutto effettuato in modo digitale. La particolarità è che risulta necessario introdurre un DAC per la generazione della tensione DC da fornire in ingresso all'ADC.

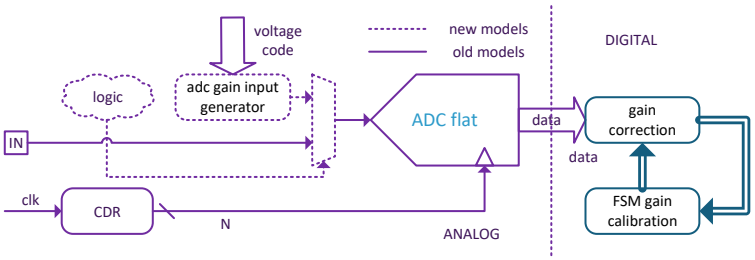


Figura 4.10: Schema ADC offset calibration

4.4.7 RX ADC skew

Come anticipato nella sottosezione 2.5.3 è fondamentale controllare lo skew dei segnali di clock in un timing interleaving ADC al fine di ottenere un ottimale campionamento dei dati ed evitare l'introduzione di jitter deterministici.

Durante il funzionamento reale la variazione PVT di skew associato di ogni fase può essere legata alla somma dei contributi di skew legata alle varie non idealità presenti. Nel modello sviluppato si è deciso di tenere questo contributo completamente casuale e scorrelato dal valore e di generarlo tramite una variabile pseudocasuale.

In figura 4.11 è presente una panoramica di quali sono i blocchi di `rx_ana_flat` coinvolti in questa calibrazione. Il primo è un blocco interposto nel percorso del clock che parte dal CDR e arriva all'ADC che si occupa dell'attuazione della correzione dello skew e il secondo è un blocco che si occu-

pa della generazione di un determinato pattern di dati in ingresso utilizzato dall'algoritmo per la misurazione dello skew di ogni fase.

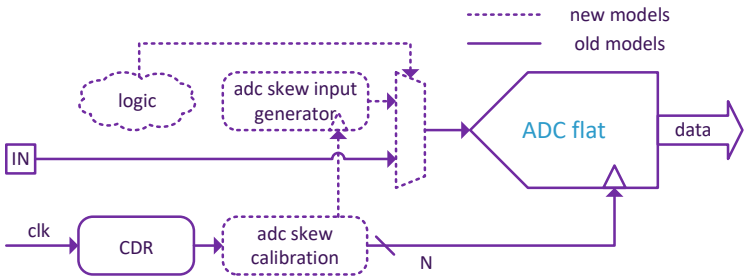


Figura 4.11: Panoramica della calibrazione

In figura 4.12 è mostrata nel dettaglio l'implementazione del primo blocco. Oltre alla classica logica sequenziale e combinatoria per la valutazione del valore di skew si attua anche lo skew nel percorso, come mostrato nel listato 4.1. Anche se da specifiche il range di skew sarebbe dovuto essere positivo e negativo rispetto al valore di riferimento nullo, si è optato per l'inserimento di un valore di skew di riferimento positivo chiamato `t_0` al fine di evitare errori dovuti all'assegnazione di un ritardo negativo in un segnale verilog.

```
1  always @(*) begin
2  for (int i=0 ; i<N ; i++) begin
3    out_clk[i] <= #(corrected_delay[i]) in_clk[i];
4  end
5  end
```

Listing 4.1: assegnazione delay in verilog alle fasi

In una calibrazione di questo tipo si evince tutto lo svantaggio di introdurre le variazioni PVT direttamente nel datapath. Ad esempio se si volesse effettuare la simulazione di questa calibrazione non sarebbe possibile farlo e ottenere una convergenza certa senza prima effettuare la calibrazione di offset e gain. Questo perché gli errori PVT di guadagno e offset presenti sui dati non renderebbero possibile all'algoritmo la corretta discriminazione dello skew, portando a degli errori nella simulazione.

Generazione del segnale di ingresso

Una delle parti che ha richiesto una maggiore attenzione nella modellazione di questo blocco di calibrazione, riguarda la generazione del segnale di ingresso. Il

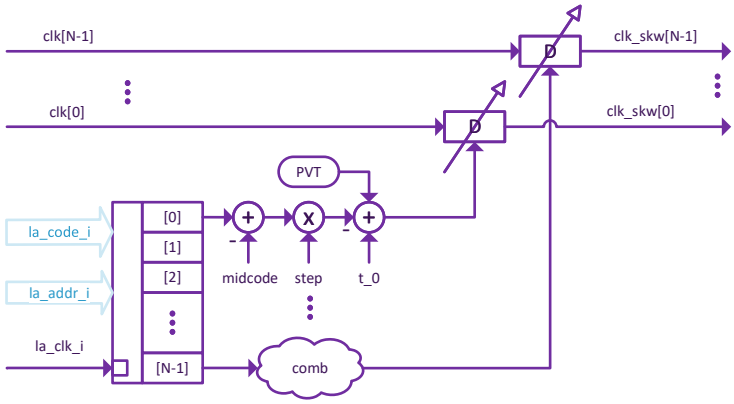


Figura 4.12: Schema di principio ADC Skew Calibration

verilog non nasce come uno strumento per la modellazione di segnali analogici, e per questo motivo la generazione in modo efficace di un segnale analogico può risultare non proprio immediata.

In un algoritmo utilizzato per la calibrazione delle fasi di un interleaved ADC è fondamentale riuscire a captare la minima variazione di fase del segnale. Ad esempio la generazione di un segnale sinusoidale potrebbe essere sviluppata nella modalità presente in listato 4.2, ovvero con la generazione di un segnale di clock che definisce la frequenza di campionamento per la discretizzazione di questo segnale. Se il segnale non è discretizzato in maniera opportunamente fitta esso avrebbe un evidente andamento a gradini, una variazione di ampiezza ΔT dovuto alla modifica dello skew non verrebbe interpretato in maniera corretta e non porterebbe nessun cambiamento nell'ampiezza e il segnale campionato sarebbe sempre lo stesso.

```
1  module sinwave_generator (
2    input wire sine_en,
3    output real sinewave
4  );
5
6  const real  pi = 3.141592653589;
7  const real  sine_ampl=AMPLITUDE;
8
9  real    sine_period= PERIOD;
10 realtime   time_s;
11
12 real resolution= RESOLUTION;
13
14 initial begin
15   clk = 1'b0;
16   forever begin
17     #(1.0*resolution/2.0) clk =  ~clk;
18   end
```

```
19   end
20
21   always @(posedge clk) begin
22     time_s = $realtime*1e-9; // second
23   end
24
25   assign sinewave = sine_en ? (sine_ampl * $sin(2*pi*time_s/sine_period))
         : 0;
26
27   endmodule
```

Listing 4.2: Generazione di un segnale sinusoidale

Una possibile soluzione sarebbe quella di aumentare la frequenza di campionamento, fino ad ottenere uno step di campionamento paragonabile a quello della calibrazione. Questo porta ad una particolare inefficienza, sia dal punto di vista di tempo di esecuzione della simulazione che dal punto di vista di occupazione di memoria. Eseguire una calibrazione completa infatti richiederebbe ad uno spazio di archiviazione necessario non indifferente. L'inefficienza è presente nel fatto che comunque, l'ADC campionerà il segnale con una frequenza di campionamento fissa e quindi molti dei campioni valutati saranno inutilizzati.

La soluzione adottata in questo modello è presente in listato 4.3.

```
1   module sinwave_generator_v2 (
2   input wire sine_en,
3   input wire clk_skw_0,
4   input wire clk_skw_1,
5   input wire clk_skw_2,
6   ...
7   output real sinewave
8   );
9
10  const real  pi = 3.141592653589;
11  const real  sine_ampl=AMPLITUDE;
12
13  real    sine_period= PERIOD;
14  realtime   time_s;
15
16  always @(posedge clk_skw_0 or posedge clk_skw_1 or posedge clk_skw_2 or
        ... ) begin
17  time_s = $realtime*1e-9; // second
18  end
19
20  assign sinewave = sine_en ? (sine_ampl * $sin(2*pi*time_s/sine_period))
        : 0;
21
22  endmodule
```

Listing 4.3: Versione implementata per la generazione di un segnale sinusoidale

La principale differenza è che la sinusoide è direttamente generata con una

frequenza di campionamento definita dalle fasi dell'ADC. Le fasi in questione sono quelle generate dal blocco di calibrazione, dunque sono soggette alle minime variazioni di skew generate dalla calibrazione. Un cambiamento del codice di calibrazione fa sì che il nuovo campione di sinusoide sia direttamente valutato nell'istante di tempo richiesto.

In questo caso il numero di campioni generato in questo caso è il minimo indispensabile, ovvero quello che dipende dalla frequenza di campionamento dell'ADC, ma mantenendo al contempo la massima risoluzione possibile, ovvero quella della variabile `$realtime` intrinseca del Verilog.

4.4.8 RX ILO calibration

Come spiegato nel capitolo introduttivo l'ILO possiede al suo interno un oscillatore locale programmabile e lo scopo di questa calibrazione è quella di ottenere il codice che minimizza la discrepanza tra la sua frequenza e quella del clock in ingresso al fine di favorire il fenomeno di *injection locking*. Come accennato infatti tanto maggiore è la differenza tra le due frequenze tanto difficile risulta ottenere il *locking*.

Come si evince dallo schema in figura 4.13 in questa calibrazione esistono diversi codici e configurazioni da dover settare durante la calibrazione:

- `ilo_code`: ovvero il codice del dac che modifica la corrente di ingresso di bias del ring oscillator
- `ilo_config`: ovvero il valore utilizzato per la selezione del della strenght degli inverter usati per la modifica della caratteristica tensione-frequenza dell'oscillatore locale.

Inoltre l'alogoritmo implementato richiede tre differenti feedback:

- `beat_ck/beat_ck_div`: si tratta di un clock la cui frequenza è pari alla differenza della frequenza tra il PLL e la f.r.f.;
- `beat_freq_sign`: indica se la f.r.f. e' maggiore o minore della frequenza del PLL;
- `max_vosc`; un segnale che indica se la tensione `vosc` supera una certa soglia ed è necessario diminuirla.

Per poter eseguire questa calibrazione è necessario conoscere la caratteristica dell'oscillatore dell'ILO, per essere in grado di generare correttamente il valore di beat frequency che andrà in ingresso alla macchina a stati che si occupa della gestione della calibrazione.

Data la caratteristica non ideale e quindi non lineare dell'oscillatore, è stato scelto di effettuare una linearizzazione attorno al punto di lavoro, al fine di mantenere il modello semplice. Le caratteristica dalla quale è stato interpolato

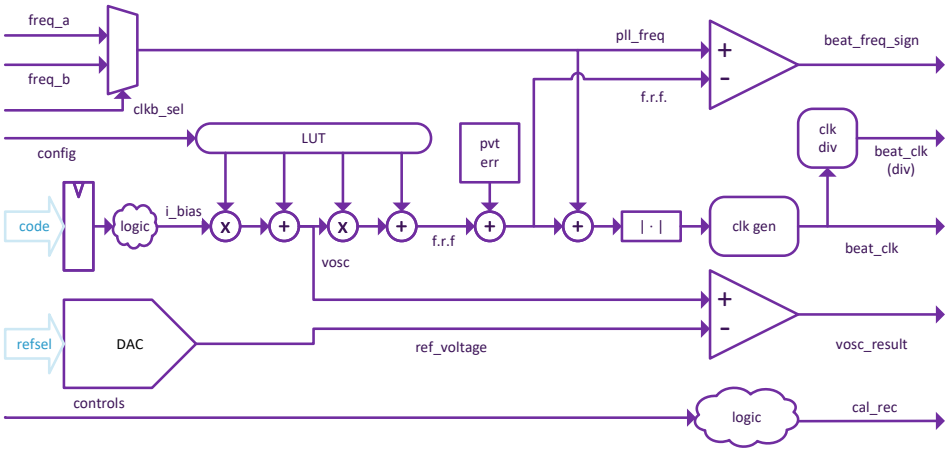


Figura 4.13: Schema calibrazione ILO

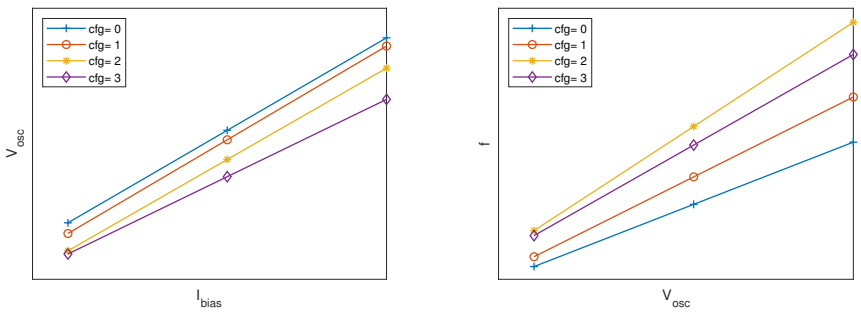


Figura 4.14: Grafico caratteristica linearizzato oscillatore ILO

è stata fornita dal team analogico, effettuando una serie di simulazioni *spice* degli schematici.

Il modello è mostrato in equazione (4.3)

$$\begin{aligned} v_{osc} &= r_{vco} \cdot i_{osc} + r_{0,vco} \\ f.r.f &= k_{vco} \cdot v_{osc} + k_{0,vco} \end{aligned} \tag{4.3}$$

La caratteristica dell'oscillatore è dipendente dal valore del parametro `ilo_config`: esso determina la strength dei transistor utilizzati all'interno dell'oscillatore, permettendo quindi con lo stesso valore di corrente di ottenere differenti frequenze e tensioni diverse. Ad esempio se l'oscillatore si trova in una condizione tale da non riuscire a raggiungere il valore di frequenza con dei parametri ragionevoli, viene selezionato un valore di config differente. Questo comportamento è stato emulato tramite una look-up table, che pilotata dal segnale di `ilo_config` fornisce i valori dei coefficienti per valutare la f.r.f. (equazione (4.4)).

$$\begin{aligned} v_{osc} &= r_{vco}[cfg] \cdot i_{osc} + r_{0,vco}[cfg] \\ f.r.f &= k_{vco}[cfg] \cdot v_{osc} + k_{0,vco}[cfg] \end{aligned} \tag{4.4}$$

Una volta determinato il valore di frequenza è generato il segnale di beat clock(listato 4.4), tenendo presenti gli accorgimenti necessari per ottenere un valore di ritardo che sia sempre positivo.

```
1     initial begin
2     beat_ck = 1'b0;
3     forever begin
4        #(1.0/freq/2.0) beat_ck =  ~beat_ck;
5     end
6   end
```

Listing 4.4: Generazione del beat clock

La variazione PVT è modellata tramite una variabile casuale che agisce direttamente sulla beat frequency.

4.4.9 RX QLL calibration

Questa calibrazione, come quella dell'ILO, mantiene lo scopo di controllare il codice che gestisce la frequenza dell'oscillatore locale dell'ILO, ma effettuando la misurazione in una condizione differente, ovvero durante l'*Injection Locking*. Qui la frequenza dell'oscillatore locale è forzata a quella del segnale di ingresso ma, a causa del principio di funzionamento dell'oscillatore, la discrepanza tra

le due free running frequency si traduce un una variazione di skew tra le fasi generate. In figura 4.15 è mostrato un esempio di come cambia lo skew in un oscillatore con quattro fasi prima e dopo la condizione di injection.

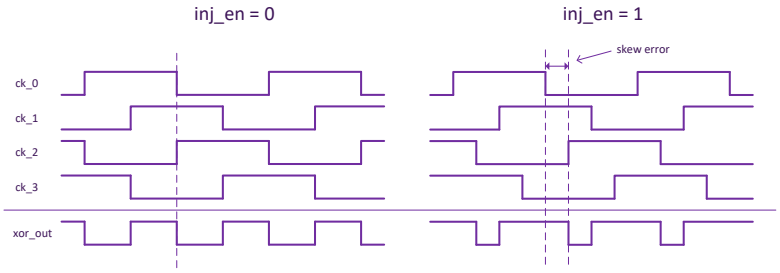


Figura 4.15: Skew tra le fasi in uscita dell'ILO

Differentemente all'ILO calibration vista in precedenza cambia la modalità con cui si ottiene il feedback necessario per la determinazione del codice di calibrazione. La calibrazione quindi agisce nella seguente modalità:

- `inj_en=0`: durante questa fase avviene la misurazione dello skew tra fasi in uscita dall'oscillatore necessario per la generazione del segnale di riferimento utilizzato nel comparatore;
- `inj_en=1`: in questa condizione avviene la vera e propria calibrazione. Il confronto tra lo skew misurato in questa condizione e quello misurato durante la fase di startup fornisce alla FSM l'informazione sull'errore di frequenza dandole la capacità di agire determinando il codice di calibrazione corretto. Minimizzando l'errore di skew tra le fasi si minimizza infatti la differenza tra le due frequenze.

La misurazione dello skew tra le fasi avviene attraverso uno XOR - Phase Detector (XOR-PD). In ingresso sono poste le fasi e in uscita si ottiene un'onda quadra con Duty Cycle proporzionale allo sfasamento (equazione (4.5)). Questo segnale è convertito, attraverso un filtro passa basso, in un segnale DC utilizzato dal comparatore.

$$\delta_c = \frac{\theta}{180^\circ} \tag{4.5}$$

Per mantenere una semplicità del modello si è deciso di non realizzare tutto il meccanismo implementando ogni blocco descritti, ma è stato scelto di modellare il tutto attraverso formule analitiche.

Lo skew misurato durante la prima fase della calibrazione non assume un valore di tipo deterministico, ma è influenzato anch'esso da variazioni PVT, siano esse dovute a variazioni di tensioni di alimentazione, di implementazione

della XOR, ... É stato scelto di utilizzare una prima variabile casuale PVT per modellare tutto ciò.

Per quanto riguarda il contributo di errore di skew introdotto durante la condizione di injection è necessario effettuare delle precisazioni. Come facilmente intuibile, esso dovrebbe dipendere dal risultato della ILO calibration, dato che è quello che determina il valore di differenza di frequenza. Al fine di mantenere indipendenti le calibrazione è stato comunque deciso di descrivere l'errore attraverso una variabile pseudocasuale indipendente opportunamente scalata in un range specifico.

Nel modello si è deciso quindi di modellare il duty cycle complessivo tramite un primo contributo dovuto alle non idealità del sistema di misurazione e un secondo che si abilita durante la fase di injection e che emula il contributo di skew dovuto all'errore di frequenza (equazione (4.6)).

$$\Delta_{\delta_c,tot} = \Delta_{\delta_c,PVT,startup} + \texttt{inj\_en} \cdot (\Delta_{\delta_c,PVT,inj} - \Delta_{correction}) \tag{4.6}$$

Dopo aver definito come è stata implementata la misura del duty cycle è necessario introdurre anche la parte relativa alla correzione dello skew. A differenza dell'ILO calibration questa volta il codice di calibrazione non influenza la frequenza ma lo skew. La relazione che lega il codice dallo skew è di tipo lineare (equazione (4.7)). La natura non deterministica del codice di calibrazione iniziale rende necessario introdurre un registro che memorizzi questo valore ad inizio calibrazione per valutare di quanto il codice attuale si discosta da quello iniziale e valutare la correzione da applicare.

$$\Delta_{correction} = k \cdot (\texttt{CODE} - \texttt{CODE\_STORED}) \tag{4.7}$$

Quanto appena descritto potrebbe causare delle complicazioni a causa di una possibile saturazioni del codice di calibrazione. Per chiarire la problematica è possibile immaginare che se durante la ILO calibration il codice converge ad un valore prossimo alla saturazione, la massima differenza tra `CODE` e `CODE_STORED` potrebbe essere non sufficientemente elevata da consentire una corretta calibrazione. Questo rende evidente la necessità di valutare i massimi range di variazione PVT sia per l'ILO che per la Quadrature Locked Loop (QLL) in maniera non indipendente.

In alcuni casi è necessario effettuare una conversione da skew a gradi, e per questo motivo è necessario introdurre un mux che selezioni il valore corrispondente di frequenza.

$$\theta = 2\pi f \cdot \Delta T \tag{4.8}$$

Lo schema di principio del modello realizzato è mostrato in figura 4.16.

Come è possibile intuire la corretta esecuzione della calibrazione può essere verificata controllando se la condizione $\Delta_{\delta_c,PVT,inj} = \Delta_{correction}$ è verificata.

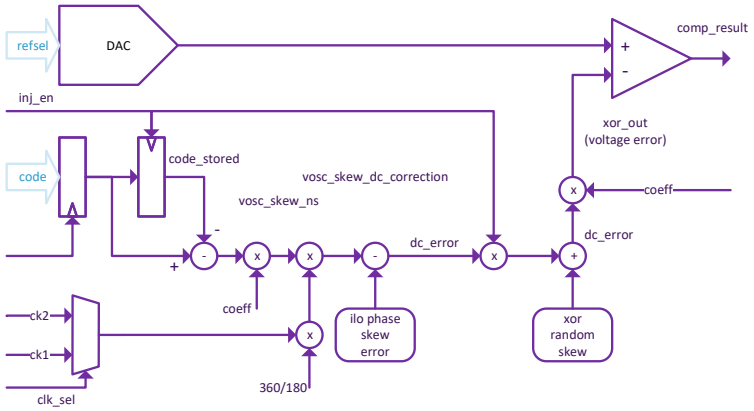


Figura 4.16: Diagramma calibrazione QLL

4.5 Calibrazioni TX

Per quanto riguarda il trasmettitore i blocchi relativi alle calibrazioni sono introdotti sempre all'interno del modello flat della macro analogica (`tx_ana_flat`). In questo caso tutti i blocchi sono indipendenti dal modello preesistente, dato che non non è stata presente la necessità di attuare le correzioni all'interno del datapath analogico (figura 4.17).

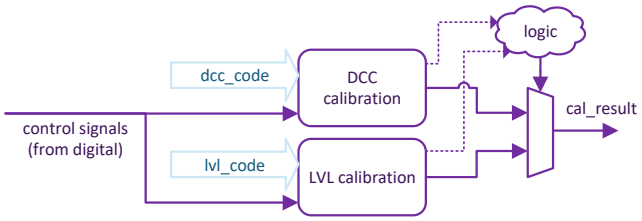


Figura 4.17: Panoramica delle calibrazioni nel modello analogico flat del trasmettitore

4.5.1 TX DCC calibration

L'obiettivo di questa calibrazione risiede nel controllo del duty cycle (δ_c) del clock con periodo $2UI$ utilizzato per la serializzazione dei simboli. Come anticipato nel capitolo introduttivo (capitolo 2), la serializzazione dei dati avviene tramite un multiplexer (figura 4.18): la trasmissione dei simboli pari avviene per tutta la durata del periodo alto del segnale di clock e i simboli dispari sono

trasmessi per tutta la durata del livello basso. Una discrepanza tra la durata del livello alto e del livello basso può causare errori non trascurabili, dato che ogni simbolo trasmesso non presenta la stessa durata e ogni simbolo potrebbe non essere campionato in maniera ottimale.

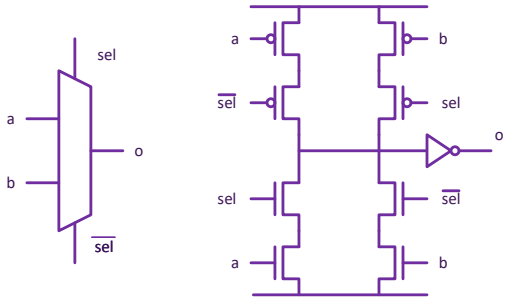


Figura 4.18: Esempio di multiplexer utilizzato per la serializzazione

Un multiplexer come quello illustrato presenta due segnali di selezione, e per questo motivo sono presenti 2 segnali di clock sfasati tra loro di 180°, che da ora in avanti saranno chiamati `clk_0` e `clk_180`.

Nell'architettura del Ser-Des, la misurazione del δ_c avviene utilizzando il segnale in uscita del multiplexer quando in ingresso sono presenti degli specifici pattern. In figura 4.19 si nota come in funzione dei pattern di ingresso del multiplexer (1010 o 0101) il segnale in uscita sia uguale rispettivamente a `clk_0` o a `clk_180`,perciò filtrandolo opportunamente è possibile ottenere un livello di tensione costante proporzionale al δ_c.

Il valore corrispondente a un duty cycle ottimale non è indipendente dai valori della tensione di alimentazione e da altri fattori, per questo è necessaria una fase preliminare volta a determinare il livello di tensione di riferimento a cui corrisponde un δ_c del 50%. Dato che l'uscita è di tipo differenziale, l'algoritmo utilizzato prevede la misurazione di ognuno dei duty cycle rispettivamente da entrambi i canali, `TX_P` e `TX_N`.

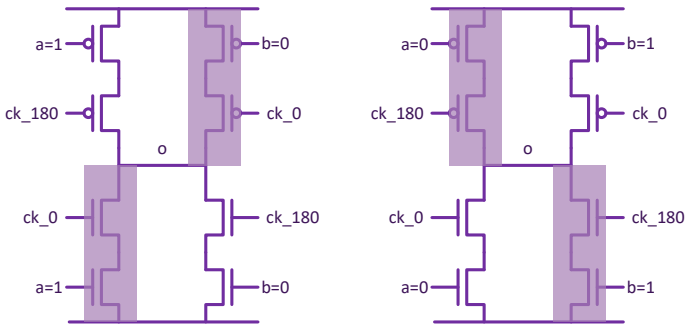


Figura 4.19: Clock in uscita dal multiplexer in funzione dei pattern di ingresso: i rami evidenziati sono quelli attivi che contribuiscono alla generazione del segnale di uscita.

Tutti i comportamenti descritti fino a questo punto devono essere implementati anche nel modello analogico comportamentale, come mostrato in figura 4.20. Il modello implementato è stato suddiviso in due differenti parti: la prima parte interviene per la misurazione del livello di tensione di riferimento e la seconda per l'effettiva calibrazione del δ_c, selezionate tramite l'ausilio di un multiplexer.

Come anticipato, il δ_c è valutato in uscita ad entrambi i pad, questo porta a dover modellare in maniera casuale i livelli di tensione, come mostrato nell'equazione (4.9). Questi valori sono utilizzati così come sviluppati al fine di ottenere la misurazione e successivamente sono utilizzati per la conversione del valore di δ_c in un tensione di uscita, la quale è differente a seconda dei pad utilizzati.

$$\begin{aligned} tx_{p,1} &= VH \pm PVT\_ERROR1 \\ tx_{m,1} &= VH \pm PVT\_ERROR2 \\ tx_{p,0} &= VL \pm PVT\_ERROR3 \\ tx_{m,0} &= VL \pm PVT\_ERROR4 \end{aligned} \tag{4.9}$$

Questi livelli di tensione sono utilizzati dal convertitore analogico digitale per il calcolo del valore di riferimento. Una possibile strategia prevede la misura dei 4 livelli di tensione tramite un DAC, modellato nel seguente modo:

$$V_{ref} = \frac{V_{dd}}{N_{bit}} \cdot REF\_CODE$$

La seconda parte dell'algoritmo si occupa della misura del duty cycle. Nel modello implementato saranno quindi presenti due variabili casuali, ognuna utilizzata per modellare le variazioni PVT di ogni segnale di clock. Questi

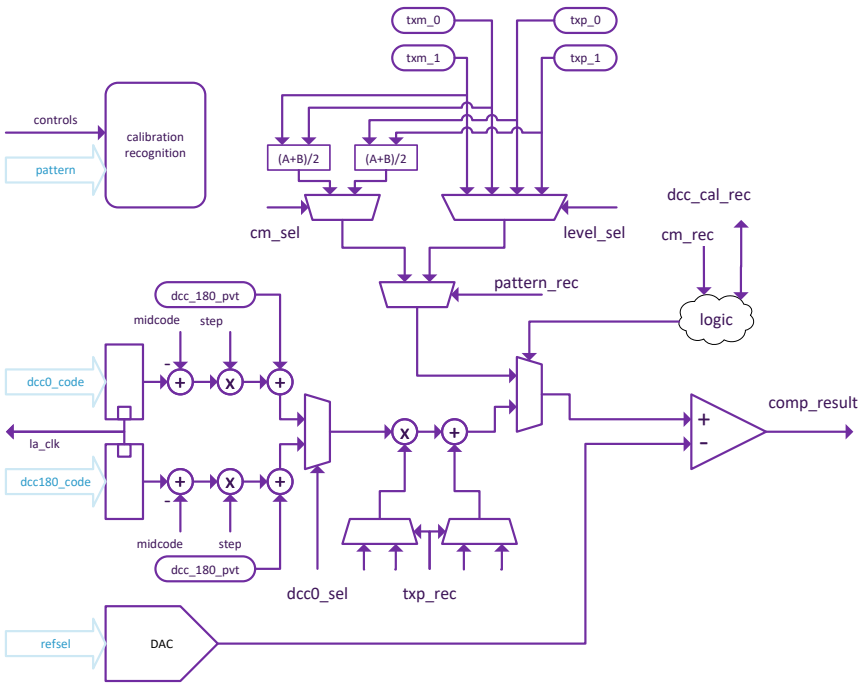


Figura 4.20: Schema di funzionamento TX DCC calibration

valori hanno la necessità di essere convertiti in un livello di tensione contenente anche le informazioni relative a quale canale di trasmissione

Il duty cycle

$$\begin{aligned} v_{out,p} &= tx_{p,1} \cdot \delta_c\big|_x + tx_{p,0} \cdot \left(1 - \delta_c\big|_x\right) \qquad \text{dove } x = 0, 180° \\ v_{out,m} &= tx_{m,1} \cdot \delta_c\big|_x + tx_{m,0} \cdot \left(1 - \delta_c\big|_x\right) \qquad \text{dove } x = 0, 180° \end{aligned} \tag{4.10}$$

Il δ_c è espresso come:

$$\delta_c = 0.5 + \delta_{c,error} \tag{4.11}$$

e quindi

$$\begin{aligned} v_{out,p} &= cm_p + \left(tx_{p,1} - tx_{p,0}\right) \cdot \delta_{c,error}\big|_x \qquad where\ x = 0, 180 \\ v_{out,m} &= cm_m + \left(tx_{m,1} - tx_{m,0}\right) \cdot \delta_{c,error}\big|_x \qquad where\ x = 0, 180 \end{aligned} \tag{4.12}$$

Il modello sviluppato presenta una parte di logica utilizzata al fine di riconoscere se i segnali di controllo assumono i valori necessari della calibrazione,

e successivamente, essi vengono decodificati per la generazione dei segnali di selezione dei multiplexer utilizzati per la calibrazione. Ad esempio, come il segnale `txp_rec` per la selezione dei coefficienti relativi ai pad di uscita.

Per assicurarsi che la calibrazione sia andata a buon fine è necessario analizzare che il valore di `dcc_180_pvt` sia uguale in modulo al valore di correzione. Ottenere una convergenza tra i valori in ingresso al comparatore è solo una condizione necessaria ma non sufficiente per ottenere una calibrazione corretta. Ad esempio, un'errata configurazione dei valori di `refsel` da parte del firmware potrebbe fornire un'uscita apparentemente corretta dal comparatore, ma corrispondente a un livello di tensione che non riflette il duty cycle effettivo desiderato.

4.5.2 TX LVL calibration

Questo modello implementa la calibrazione dei livelli di ampiezza dei pad di uscita del trasmettitore. Attraverso un codice di calibrazione è possibile introdurre delle variazioni di correnti di bias al fine di produrre una variazione della tensione. L'algoritmo utilizzato sfrutta una molteplici misurazioni per la ricerca di un codice ottimale.

Data la forte non linearità presente tra il codice e i livelli di tensione in uscita la scelta implementativa è ricaduta nell'utilizzo di una Look-Up Table (LUT): essa è generata estraendo la caratteristica dalle simulazioni SPICE degli schematici analogici forniti dal team analogico.

Dopo di ché i dati in uscita della LUT sono utilizzati per la generazione dei livelli di tensione dei pad e che a loro volta sono utilizzati per la generazione dei segnali si misurazione richiesti, come il Commmon Mode e lo swing.

Come per gli altri modelli anche in questo caso risulta necessario l'introduzione di variabili casuali, al fine di emulare le variazioni pvt e avere la possibilità di verificare il comportamento in punti di lavoro differenti.

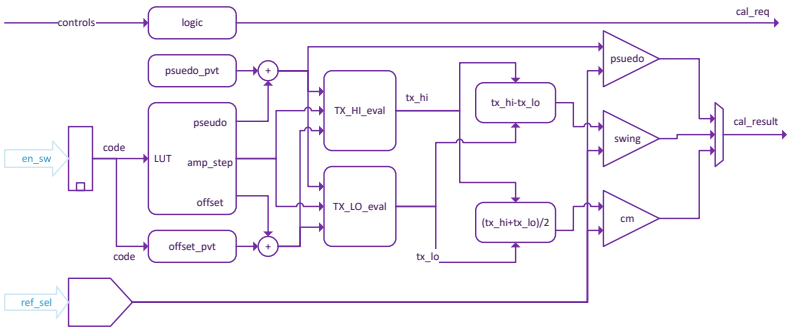


Figura 4.21: Schema di principio LVL calibration

4.6 Analisi prestazioni simulative dei modelli

La fase successiva allo sviluppo di ogni modello consiste nella valutazione dell'impatto *quantitativo* sul tempo di esecuzione delle simulazioni, al fine di verificare che non introducessero colli di bottiglia e che portassero rallentatemi eccessivi delle simulazioni.

L'efficacia dei modelli è stata analizzata attraverso due metriche differenti per avere una visione più completa:

- la prima consiste nel valutare quanto il tempo di simulazione della parte analogica impatti nel tempo di simulazione complessivo, per capire effettivamente quale sia il contributo maggiore tra la parte digitale e quella analogica;
- Il secondo consiste nel valutare il miglioramento rispetto al modello analogico già esistente.

4.6.1 Impatto sul tempo di simulazione complessivo

I risultati ottenuti da questa analisi sono stati effettuati considerando, non i tempi di simulazione dei singoli modelli sviluppati, ma i tempi di simulazione di tutto il modello analogico, questo perché altrimenti non sarebbe stato possibile effettuare una stima realistica, dato che comunque alcune calibrazioni sfruttano anche le altre parti del modello. Inoltre la durata della simulazione è dettata dalla tempo di simulazione dell'intero modello. Per fare un esempio, durante la calibrazione dello skew dell'ADC (sottosezione 4.4.7) è coinvolto il modello flat completo, e questo porta ad avere una misura di quanto sia effettivamente l'overhead complessivo.

In tabella 4.1 sono riportati a titolo di esempio i risultati ottenuti per alcune delle calibrazioni, i dati sono riportati come percentuale del tempo di simulazione della parte analogica rispetto al tempo di simulazione del ricevitore e anche come rapporto tra il tempo di simulazione della parte analogica rispetto a quella digitale, utile nel caso in cui sono presenti elevate discrepanze tra i due.

CALIB	HFEQ	DCC	SKEW	OFST	ILO	QLL	media
$\frac{T_{\texttt{rx\_ana\_flat}}}{T_{\texttt{rx, tot}}}$ [%]	35.6	36.8	39.1	41.7	37.2	38.8	38.2
$\frac{T_{\texttt{rx\_ana\_flat}}}{T_{\texttt{rx, rtl}}}$	0.55	0.58	0.64	0.77	0.59	0.64	0.63

Tabella 4.1: Analisi tempo di simulazione di `rx_ana_flat` rispetto al ricevitore completo.

Da i dati ottenuti da questo metodo di confronto può essere tratta la conclusione che il modello analogico non introduce nessun colli di bottiglia o anomalie

sui tempi di simulazione in nessuna delle calibrazioni, infatti l'overhead della parte analogica risulta inferiore al tempo di simulazione della parte digitale.

Considerando una delle calibrazioni più complesse, dove è presente anche la generazione di clock e altro come la ILO porta una percentuale di simulazione molto ridotta, qui infatti il blocco `rx_cal_top` ha un tempo di simulazione del 0.4% durante una processo di calibrazione completo. Da qui si evince che non bisognerebbe agire nei blocchi ma in tutto il resto cercando di disattivarlo se non necessario.

Considerando invece i modelli per il trasmettitore i risultati ottenuti sono i seguenti:

CALIB	TX DCC	TX LVL	media
$\frac{T_{\texttt{tx\_ana\_flat}}}{T_{\texttt{tx}}}$ [%]	25.6 %	45.9%	35.7
$\frac{T_{\texttt{tx\_ana\_flat}}}{T_{\texttt{tx, rtl}}}$ [%]	0.34	0.85	0.595

Tabella 4.2: Analisi tempo di simulazione di `tx_ana_flat` rispetto al trasmettitore completo (digital + analog).

In questo caso i risultati mostrano degli overhead minori, questa differenza è da rilevare nella minore complessità generale del modello analogico del trasmettitore, che non possiede un elevato numero di componenti (indipendentemente dalle calibrazioni).

Nella maggior parte dei casi analizzati fino ad adesso (escludendo quelli relativi alle calibrazioni dell'ADC) il tempo di simulazione complessivo della parte analogica non è dettato da un elemento in particolare, come la parte della calibrazioni, ma alla presenza di tutti quelle parti necessarie al funzionamento del modello e che non possono essere disattivate, anche se non sono effettivamente coinvolte nella parte di calibrazione.

4.6.2 Confronto con i modelli già esistenti

Al fine di determinare i vantaggi introdotti rispetto gli altri modelli, è possibile effettuare le stesse simulazioni, ma utilizzando modelli differenti. Dalle simulazioni effettuate è risultato uno speed-up complessivo (considerando sia la parte analogica che digitale) di:

$$\frac{T_{\texttt{IP, non flat}}}{T_{\texttt{IP, flat}}} \approx 10$$

dove $T_{\texttt{IP, non flat}} = T_{\texttt{ana}} + T_{\texttt{rtl}}$ e $T_{\texttt{IP, flat}} = T_{\texttt{ana, flat}} + T_{\texttt{rtl}}$, cioè indicano la simulazione complessiva della parte analogica e digitale.

Nell'equazione (4.13) si mostra come sia cambiato il peso della simulazione della parte analogica rispetto la parte digitale confrontandolo con quello valutato nella sezione precedente, considerando che il tempo di simulazione della parte RTL risulta pressoché invariato.

$$\frac{T_{\texttt{ana}}}{T_{\texttt{rtl}}} = 15 \quad \rightarrow \quad \frac{T_{\texttt{ana flat}}}{T_{\texttt{rtl}}} = 0.6 \tag{4.13}$$

Capitolo 5

Conclusioni

Al termine di questo percorso di tesi è possibile evidenziare come entrambi gli approcci, quello relativo alla semplificazione della parte DSP della netlist GTECH, sia lo sviluppo dei modelli analogici comportamentali per le calibrazioni abbiano portato dei vantaggi effettivi per lo speed-up delle rispettive simulazioni.

Nel concreto è possibile affermare che la parte della semplificazione della netlist GTECH (capitolo 3), ha portato a ottenere una complessiva riduzione dei tempi di simulazione di circa il 40%, permettendo quindi di ottenere delle simulazioni a livello di sistema per i customer più efficaci.

Lo sviluppo dei modelli flat ha permesso di analizzare e aumentare la consapevolezza di come lo sviluppo di una parte non sintetizzabile, come un testbench o in questo caso un modello analogico, abbia un impatto rilevante durante la progettazione di un sistema complesso come un PHY HS Ser-Des. L'utilizzo di approcci non ottimali per lo scopo che si deve ottenere può portare a svantaggi durante la fase di test, incrementando anche di molto i tempi di simulazione e debug.

In determinati test dove non si ha la necessità di utilizzare un modello dettagliato l'approccio dell'utilizzo dei modelli flat può essere considerata una soluzione ottimale. In media, come evidenziato dai vari test effettuati su diverse calibrazioni, si è passati da un tempo necessario per simulare la parte analogica che fosse circa 15 volte la parte digitale a circa lo 0.6 del tempo necessario alla simulazione dell'RTL (quella alla quale si è realmente interessati e che non può essere modificato), ottenendo vantaggi significativi e riducendo la durata complessiva delle simulazioni di un fattore 10, ricordando che la durata complessiva delle simulazione è di svariate ore o giorni.

Ulteriori vantaggi sono da riscontrare anche sull'occupazione di memoria del modello flat. In generale, effettuare il dump delle waveform del modello preesistente richiede una occupazione in memoria non indifferente. Con il modello flat, dato l'approccio comportamentale utilizzato, il numero di segnali è molto inferiore e questo permette di risparmiare considerevoli porzioni di spazio di archiviazione.

Questi modelli subito dopo essere stati implementati, sono stati utilizzati da parte del team. Questo è stato utile per comprendere i risultati qualitativi ottenuti, i modelli permettono infatti di avere una semplicità e supporto maggiore durante la fase di debug, fornendo un ulteriore supporto rispetto a quelli già esistenti.

Ovviamente come tutti gli approcci bisogna tenere in considerazione anche la presenza di svantaggi nello sviluppo di modelli di questo tipo: i vantaggi ottenuti in termini di prestazioni simulative hanno avuto il costo di una minore precisione dei modelli. Sebbene questo non sia influente nelle simulazioni e nelle applicazioni prese in considerazione, ci sono altre applicazioni in cui questi modelli non possono essere usati. Infatti non possono sostituire gli altri modelli esistenti, ma possono essere usati come supporto e strumenti aggiuntivi.

Un possibile sviluppo futuro potrebbe riguardare l'implementazione di questi modelli di calibrazione anche per le altre parti della macro analogica, non solo al trasmettitore o al ricevitore, come quella che comprende i PLL per la generazione dei clock di riferimento.

Acronimi

ADAS Advanced Driver Assistance Systems. 1

ADC Analog to Digital Converter. v, vi, 21, 24, 44, 46, 47, 50–52, 54, 56, 67

AI Artificial Intelligence. 1

ASIC Application Specific Integrated Circuit. 5

ATT Attenuator. vi, 48, 49

BBJ Bounded Uncorrelated Jitter. 14

BER Bit Error Rate. 10, 13, 24

CDR Clock Data Recovery. 20, 24, 33, 36, 49, 52

CTLE Continuos Time Linear Equalizer. 20, 47

D-Lev Decision Level. 22

DAC Digital to Analog Converter. 18, 42, 52, 63

DCC Duty Cycle Correction. vi, 19, 50

DCD Duty Cycle Distortion. 14, 18

DDJ Data Dependant Jitter. 14

DFE Decision Feedback Equalizer. v, 22, 23

DJ Deterministic Jitter. 14

DSP Digital Signal Processing. 2, 28, 29, 69

f.r.f. free running frequency. 26, 56, 58

FEC Forward Error Correction. 7, 8

FFE Feed Forward Equalizer. v, 17–19, 21–23, 33, 34

FIR Finite Impulse Resoponse. vi, 17, 18, 34, 36

FSDB Fast Signal Database. 29

FSM Finite State Machine. 42, 45, 59

GTECH Generic Technology. 3, 4, 27, 28, 30, 32, 34–36, 69

HFEQ High Frequency Equalizer. vi, 20, 48

HPC High-Performance Computing. 1, 3

HS Ser-Des High Speed Ser-Des. 5, 10, 24, 69

ICs Integrated Circuits. 5

ILO Injection Locking Oscillator. vi, 24, 26, 49, 56–60

IoT Internet of Things. 1

IP Intellectual Property. 3, 7, 9, 28, 43

ISI Inter Symbolic Interfiarence. v, 12, 14, 17, 21–23, 28

LUT Look-Up Table. 65

ML Machine Learning. 1

MM-PD Mueller Muller Phase Detector. 24

NRZ Not Return to Zero. v, 2, 8, 9

PAM-4 Pulse Amplitude Modulation - 4. v, 2, 8–12, 16, 17, 19, 22, 24

PCI Peripheral Component Interconnect. 6

PCI-SIG Peripheral Component Interconnect - Special Interest Group. 2, 6

PCIe Peripheral Component Interconnect Express. v, 2, 3, 6, 7, 10, 16, 24

PDF Probability Density Function. 13

PHY Phisical Layer. 3, 4, 27, 41, 69

PI Phase Interpolator. 24–26

PJ Periodic Jitter. 14

PLL Phase Locked Loop. 14, 24, 26, 56

PPA Power Performance Area. 2

PVT Process Voltage Temperature. 41, 43, 44, 46, 50, 52, 53, 59, 60, 63

QLL Quadrature Locked Loop. 60

RJ Random Jitter. 13

RTL Register Tranfer Level. 3, 4, 27–29, 34, 41, 43, 45, 69

RX Receiver. vi, 50

SAR Successive Approximation Register. 50

SDC Standard Delay Constraint. 27

SDF Standard Delay Format. 27

Ser-Des Serializer-Deserializer. v, 2, 3, 5, 6, 9, 15, 43, 50, 62

SNR Signal to Noise Ratio. 10

SoC System On Chip. 1, 3, 28

TX Transmitter. 17, 21

VCO Voltage Controlled Oscillator. 26

VGA Variable Gain Amplifier. 20, 47

XOR-PD XOR - Phase Detector. 59

Bibliografia

[1] Herman Eiliya. *Grasp the Ins and Outs of High-Speed I/O*. Accessed: 2025-09-23. URL: https://www.edn.com/grasp-the-ins-and-outs-of-high-speed-i-o/.

[2] M. Hsieh e G. Sobelman. «Architectures for Multi-Gibìgabit Wire-Linked Clock and Data Recovery». In: *IEEE CIRCUITS AND SYSTEM MAGAZINE* (2008).

[3] IEEE. *International Roadmap for Devices and Systems*. Accessed: 2025-11-10. 2022.

[4] Sam Palermo. *Lecture 12: CDRs,ECEN720, Analog & Mixed-Signal Center Texas A&M University*. Accessed: 2025-09-29. 2025.

[5] Sam Palermo. *Lecture 8: RX FIR, CTLE, DFE, Adaptive Eq., Analog & Mixed-Signal Center Texas A&M University*. Accessed: 2025-09-29. 2025.

[6] Behzad Razavi. «The Decision-Feedback Equalizer». In: *IEEE SOLID-STATE CIRCUITS MAGAZINE* (2017). Accessed: 2025-09-29.

[7] David R. Stauffer. «High Speed Serdes Devices and Applications». In: Springer, 2008. Cap. 3.3.2.

[8] Synopsys. *Synopsys - What is a SerDes?* Accessed: 2025-09-23. URL: https://www.synopsys.com/glossary/what-is-serdes.html.

[9] Synopsys. *Synopsys - What is PCI Express (PCIe)?* Accessed: 2025-09-23. URL: https://www.synopsys.com/glossary/what-is-pci-express.html.

[10] Agilent Technologies. «Using Clock Jitter Analysis to reduce BER in Serial Data Applications». In: *Application Note* (2006). Accessed: 2025-09-29.