ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

BOLOGNA

INGEGNERIA ELETTRONICA

TESI DI LAUREA

ELABORAZIONE DEI SEGNALI NEI SISTEMI ELETTRONICI

Sistemi di comunicazione real-time ad alte prestazioni tra sistemi embedded basati su FPGA

SARA MOROTTI

RELATORE:
Riccardo Rovatti

CORRELATORE:
Marco Padovani

Anno Accademico 2017/2018

Sessione II

INDICE

INTRODUZIONE

Nel corso di questa esperienza mi è stato richiesto, supportata dall'ufficio ricerca e sviluppo all'interno dell'azienda Sacmi Imola, di sviluppare un sistema di comunicazione real time ad alte prestazioni basato sull'uso dei transceiver. Il sistema richiedeva determinate specifiche, tra cui una banda prossima ai 10 Gbps per canale, una bassa latenza e un'ottima resistenza alle interferenze esterne. Una volta scelta la tecnologia più idonea, si è poi proceduto alla realizzazione di un prototipo con l'ausilio di simulazioni mirate per dimostrare l'efficacia del sistema nella sua totalità. In questo elaborato si presenterà inizialmente una panoramica sulla teoria riguardante le singole parti del sistema di trasmissione progettato, sia per quanto riguarda le schede FPGA, sia per quanto riguarda i cavi di trasmissione scelti e il software di simulazione utilizzato per le prove del progetto, Quartus. Successivamente si presenterà il progetto nella sua totalità e tutte le prove che sono state effettuate in fase di avanprogetto. Infine, si entrerà nel cuore della progettazione vera e propria descrivendo come il progetto finale sia effettivamente stato imbastito e sia stato verificato il suo corretto funzionamento, traendone le dovute considerazioni.

1. SISTEMI EMBEDDED FPGA-BASED

1.1 CENNI SULLE FPGA

Le schede FPGA, o Field Programmable Gate Array, sono dei circuiti integrati le cui funzionalità sono programmabili via linguaggi di descrizione dell'hardware, principalmente VHDL o Verilog. Questi dispositivi possono realizzare funzioni logiche anche molto complesse, e sono particolarmente utili per la loro scalabilità. Inizialmente, le FPGA contavano poche migliaia di porte logiche. Ora, grazie al progresso di questi ultimi due decenni, una FPGA consta di qualche milione di porte logiche per singolo dispositivo. Esistono due tipologie principali di FPGA, ovvero quelle riprogrammabili una sola volta, oppure riprogrammabili numerose volte. Il primo tipo di dispositivo è detto OTP, ovvero One Time Programmable, ed è costituito da componenti il cui stato di funzionamento cambia in modo permanente, permettendo di mantenere la configurazione allo spegnimento del dispositivo. Nel secondo caso invece, i dispositivi sono basati su tecnologia SRAM (Static Random Access Memory), e devono essere riprogrammati ad ogni accensione, avendo una memoria di configurazione volatile. Il vantaggio più notevole delle schede FPGA, è che si tratta di dispositivi piuttosto standard la cui funzionalità da implementare non viene impostata dal produttore. Questo ne consente una produzione su larga scala a basso prezzo. Inoltre, la loro genericità li rende adatti a un gran numero di applicazioni differenti. Esse quindi sono programmate dall'utente finale, consentendo la diminuzione dei tempi di progettazione, di verifica mediante simulazioni e di prova sul campo dell'applicazione. Il grandissimo vantaggio dal punto di vista dell'utente, è che permettono di apportare eventuali modifiche o correggere errori semplicemente riprogrammando il dispositivo in qualsiasi momento. Per questo motivo sono utilizzate ampiamente nelle fasi di prototipizzazione.

Le FPGA generalmente sono montate su PCB boards che usualmente contengono i connettori, che permettono di interfacciare l'FPGA con altri dispositivi esterni (ad esempio un PC per programmarla), un oscillatore al quarzo che genera il segnale di clock e alcuni switches che permettono di selezionare diverse frequenze di clock.

La struttura interna delle FPGA, è composta da una matrice di blocchi di logica configurabile (Configurable Logic Blocks) ,risorse di routing (ovvero interconnessioni programmabili) e blocchi di I/O configurabili. I wires vengono collegati nelle interconnessioni (switch matrix) tramite un dispositivo, che può essere un transistor o un antifuse a seconda della tecnologia. Come detto, il vantaggio della SRAM è che sia riprogrammabile. Tuttavia esso può essere anche uno svantaggio in quanto spegnendo la scheda si perde la programmazione. Viene quindi affiancata a volte una piccola ROM che riprogramma il sistema all'avvio.

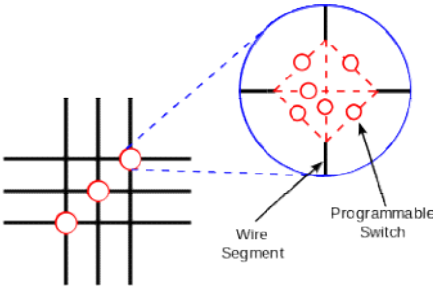


Figura 1 descrizione di una switchmatrix

Come già detto, il cuore della struttura delle FPGA sono le CLB.

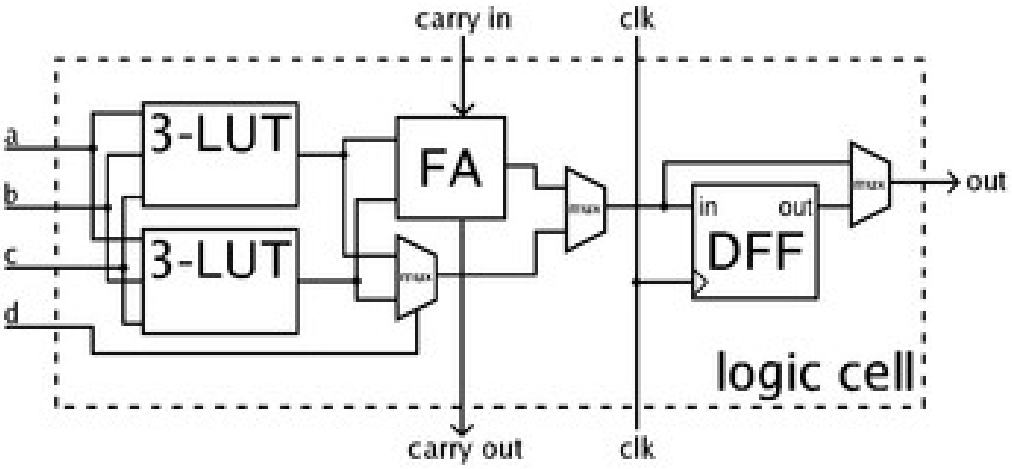


Figura 2 struttura di una CLB

Esse sono composte dalle cosiddette Look Up Tables o LUTs. Ciascuna LUT implementa le funzioni logiche desiderate a n ingressi e un'uscita memorizzando la tabella di verità della funzione stessa. Le CLB dunque possono essere connesse fra loro, permettendo così di realizzare funzioni booleane complesse. Le LUT generalmente sono composte da una memoria SRAM da 16 bit e da un multiplexer a 4 ingressi: una volta configurate possono generare qualsiasi funzione logica a quattro ingressi ciascuna. Vi sono anche le interconnessioni relative alla logica di set/reset e chip enable, ai segnali di clock, e ai segnali provenienti dalle altre slice del dispositivo. La scelta di utilizzare LUT a soli quattro ingressi

risiede nel fatto che la complessità di una LUT cresce esponenzialmente all'aumentare del numero di ingressi, e risulta dunque poco gestibile. Raggruppare insieme alcune LUT in un CLB e connetterle con una rete locale di interconnessioni consente infatti una maggiore velocità, dovuta al fatto che questi tipo di interconnessione è più veloce di quella generale tra blocchi logici distinti. Il CLB è inoltre dimensionato al fine di ottimizzare il numero di connessioni locali e globali in funzione dell'area occupata: CLB troppo grandi implicano che l'area necessaria per le interconnessioni locali superi quella risparmiata grazie al raggruppamento delle LUT contenute in esse.

Sempre per quanto riguarda l'architettura, le FPGA sono composte principalmente da due strati: quello di Building Blocks e quello di Configuration Blocks. Il primo layer, contenente le CLBs, i blocchi IO, le risorse wire e i blocchi RAM, corrispondono al livello superiore, ovvero quello programmabile dall'utente. Il secondo layer invece, quello inferiore, configura le CLBs e le interconnessioni per permettere l'implementazione del circuito progettato dall'utente.

1.2 TRANSCEIVER AD ALTA VELOCITA'

Il transceiver è un dispositivo costituito da un trasmettitore e un ricevitore che condividono alloggiamento e circuiti. Essi vengono utilizzati all'interno delle PCB boards per la trasmissione e la ricezione dei dati ad alta velocità. Al loro interno infatti, essi sono composti da una serie di blocchi dedicati all'impacchettamento e allo spacchettamento dei dati inviati e ricevuti dalla scheda stessa e da ulteriori blocchi intermedi di equalizzazione e encoder/decoder per ottimizzare il flusso di dati di passaggio. Solitamente ad oggi si utilizzano dei transceiver in grado di operare a delle bit rate seriali superiori a 1 Gbit/s per adattarsi ai flussi di dati raggiunti dalla tecnologia. I dati inviati dalla scheda all'interno dei transceiver vengono rielaborati in modo tale da avere in uscita dei dati con trasmissione seriale partendo da dati invece trasmessi in parallelo. Viceversa, i dati ricevuti in serie vengono spacchettati e ritrasmessi in parallelo alla scheda. I blocchi intermedi si occupano del bilanciamento del canale di trasmissione e della robustezza del dato che si deve trasmettere, in modo da ridurre al minimo la possibilità di errori nella trasmissione.

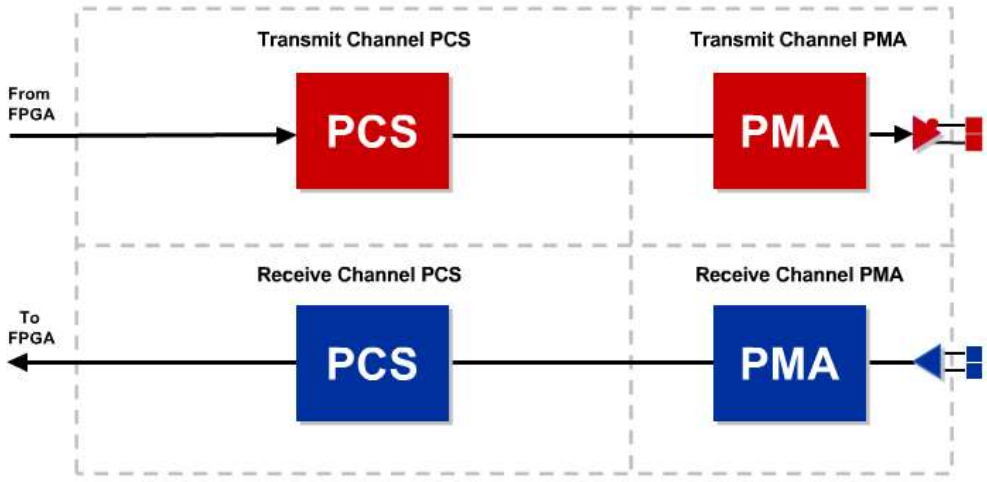


Figura 3 schema generico della struttura interna dei transceiver

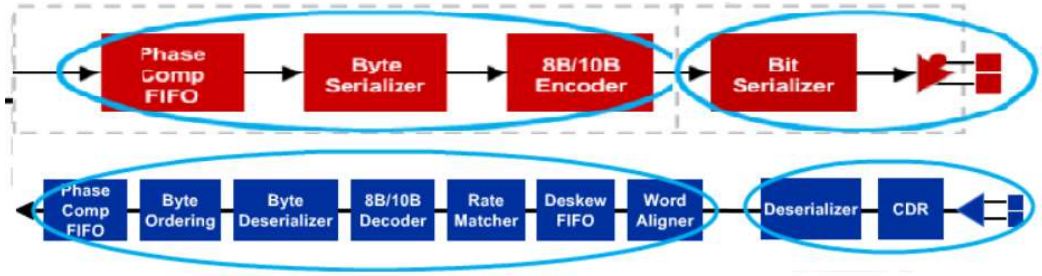


Figura 4 schema dei blocchi che compongono la struttura interna dei transceiver

1.2.1 NATIVE PHY TRANSCEIVER

Nell'ambiente di sviluppo (Quartus) saranno utilizzati dei blocchi base denominati "Native Phy Transceiver" per l'implementazione della comunicazione via transceiver. Questi blocchi permettono un ampio accesso all'hardware di basso livello, consentendo di configurare ed ottimizzare i transceiver in base alle proprie esigenze di progettazione.
Il blocco mette a disposizione anche la possibilità di eseguire una riconfigurazione dinamica che permette al dispositivo di modificare le proprie funzionalità durante il suo normale funzionamento.
Intel mette a disposizione alcune configurazioni "pre-impostate" che permettono di utilizzare alcuni protocolli tra i più comuni, come ad esempio:

- 1G/10 Gbps Ethernet
- 10GBASE-R

- Backplane Ethernet 10GBASE-KR PHY
- Interlaken
- PHY IP Core for PCI Express (PIPE)
- XAUI

In questo progetto partiremo dal blocco base, personalizzandolo in base alle nostre esigenze, creando quindi un "Protocollo Custom".

1.3 DEMO BOARD DI SVILUPPO: INTEL ARRIA 10 GX (10AX115SA)

Per questo progetto, è stata scelta come scheda di sviluppo una demo-board equipaggiata con l'Arria 10 GX appartenente alla fascia medio-alta del portfolio Intel.

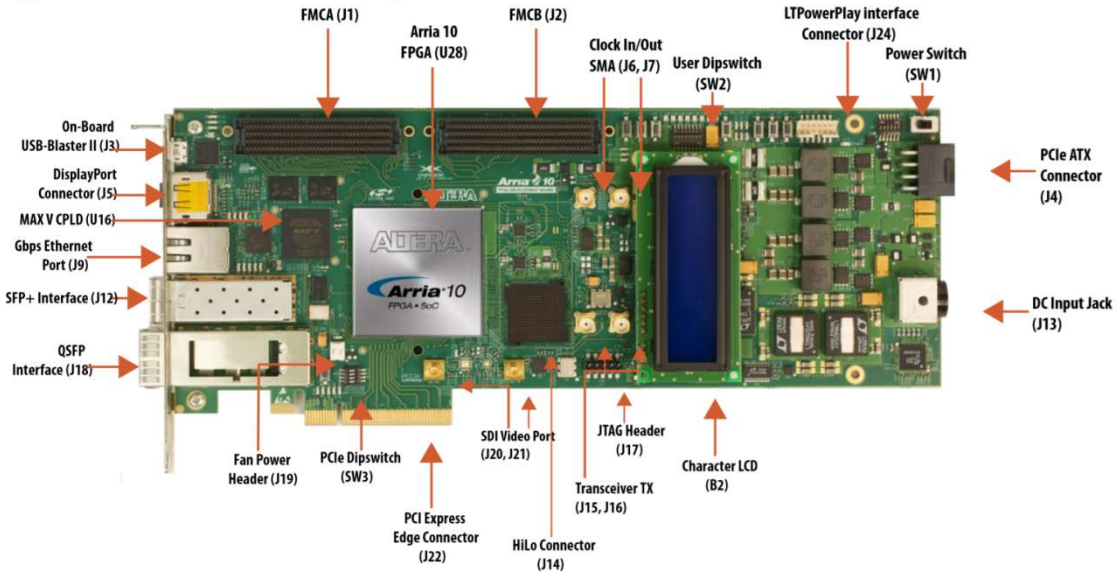


Figura 5 Arria 10 development kit

Come si può osservare in figura, il Development Kit della scheda consta, oltre all' Fpga, di memorie flash programmabili, di switch, pulsanti, led rossi e verdi, connessioni QSFP+ e SFP+, JTAG, e FMCA, pin di alimentazione e entrata USB-Blaster per la programmazione, nonché uno schermo LCD, i CLOCK IN e OUT, i transceiver e l'interfaccia PCIexpress.

Con la scheda sono fornite varie documentazione e librerie base:

- Schema elettrico e layout
- Design files
- Esempi

È stata scelta questa scheda per lo sviluppo del progetto poiché l'FPGA presenta 24 transceiver integrati con ciascuno banda massima di 17.4 Gbps, consentendoci un'elevata flessibilità nel valutare varie soluzioni in termini di architettura del sistema.

Nella demo-board sono presenti connettori SFP+ ("Small Form-factor Pluggable",fino a 10 Gbps) e QSFP ("Quad Small Form-factor Pluggable") che utilizzeremo per testare il nostro canale di comunicazione.

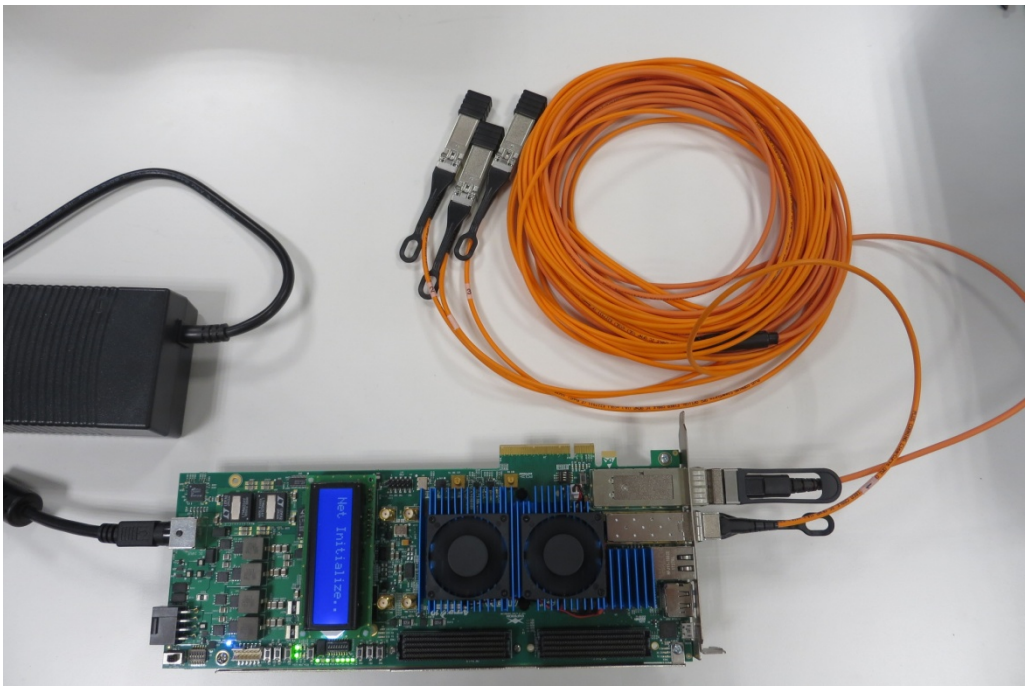


Figura 6 set up sperimentale per le prove sul Development Kit Arria 10 GX

2. PROGETTO DEL SISTEMA

Allo scopo di ottimizzare le prestazioni del sistema, in fase di avanprogetto è stato necessario effettuare alcune considerazioni sulle prestazioni desiderate dal nuovo sistema, per poi valutare l'architettura che meglio rispondesse alle caratteristiche volute.

Il sistema preso in considerazione può suddividersi principalmente in tre macro blocchi funzionali:

- SERVER
- HUB CUSTOM
- BOARD CUSTOM

Per quanto riguarda il server, esso è 'la mente' del sistema. Svolge le funzioni decisionali e di coordinamento delle attività del sistema. Allo scopo di ottimizzarne le prestazioni, si utilizzerà un server ad alte prestazioni.
Il server comunica con gli Hub Custom FPGA-based, creando un'interfaccia di comunicazione. L'hub rappresenta uno dei due terminali della comunicazione di cui si occupa questo progetto. L'altro terminale è rappresentato dalle Board Custom, che compongono l'ultimo livello del sistema fungendo da driver per applicazioni general purpose.

Nei vari livelli potranno essere utilizzati protocolli di comunicazione differenti in base all'architettura che verrà scelta, in modo da ottimizzare le varie comunicazioni.

2.1 SPECIFICHE

La comunicazione di nostro interesse è quella tra Hub e Board, e richiede le seguenti caratteristiche:

- Banda totale del Sistema: 19 Gbyte/s (152 Gbit/s)
- Bassa latenza:< 1us
- Fpga Hub Custom: ARRIA 10 GX
- Fpga Board Custom: CYCLONE 10 GX

Gli obiettivi richiesti invece sono i seguenti:

- Studio e scelta della banda ottimale per entrambi i livelli di comunicazione (server/hub e hub/board).
- Nella connessione server/hub considerare una connessione di tipo PCIe sostenibile dal dispositivo scelto in termini di banda e numero di transceiver.
- Nella connessione hub/board considerare la compatibilità con i modelli di fpga Arria 10 GX e Cyclone 10 GX.

2.2 ARCHITETTURA

Sapendo che la banda del sistema deve essere 19 Gbyte/s (152 Gbit/s), esso può essere sviluppato in tre principali architetture differenti tra loro. Di seguito verranno esposte nel dettaglio le varie architetture proposte, per poi passare ad un successivo confronto. Occorre specificare che per tutte le architetture proposte, visto l'elevato flusso dati ed i possibili collegamenti lunghi dei cavi, non è consentito inserire l'alimentazione nel cablaggio, occorrerà dunque alimentare le board in modo indipendente.

2.2.1 SOLUZIONI PROPOSTE

La prima architettura proposta è single-server. Il server alimenta quattro hub, dove a sua volta ciascuno pilota 24 board. Il flusso dati richiesto dal sistema è di 152 Gbit/s. Da questo è possibile calcolare le data rate in tutti i vari livelli del sistema. I risultati sono riportati nella tabella seguente:

PRIMA ARCHITETTURA	
Nr. SERVER	1
Nr. HUB	4
PROTOCOLLO SERVER-HUB RICHIESTO	PCIe 3.0ad x8
Nr. BOARD PILOTATE (xHUB)	24 (24 channel)
FLUSSO DATI SERVER	152 Gbit/s
FLUSSO DATI HUB	38 Gbit/s
FLUSSO DATI PER CANALE	1.6 Gbit/s
COMPATIBILITA' HUB-ARRIA 10 GX	Si
COMPATIBILITA' BOARD-CYCLONE 10 GX	Si

Tabella 1 specifiche del sistema ottenute nell'ipotesi di utilizzo della prima architettura

Osservando i datasheet delle varie FPGA, si può notare che la l'ARRIA 10 GX precedentemente descritta soddisfa pienamente le specifiche sia in termini di numero di transceiver, sia per il flusso dati che è in grado di garantire.

Symbol/Description	Condition	Transceiver Speed Grade 1	Transceiver Speed Grade 2	Transceiver Speed Grade 3	Transceiver Speed Grade 4	Unit
Chip-to-Chip (41)	Maximum data rate $V_{CCR\_GXB} = V_{CCT\_GXB} = 1.03$ V	17.4	15	14.2	12.5	Gbps
	Maximum data rate $V_{CCR\_GXB} = V_{CCT\_GXB} = 0.95$ V	11.3	11.3	11.3	11.3	Gbps
	Minimum Data Rate	1.0 (42)				Gbps
Backplane (41)	Maximum data rate $V_{CCR\_GXB} = V_{CCT\_GXB} = 1.03$ V	12.5	12.5	12.5	10.3125	Gbps
	Minimum Data Rate	1.0 (42)				Gbps

Figura 7 caratteristiche dei transceiver della scheda Arria 10 GX. Come si può osservare, essi raggiungono un flusso dati molto superiore a quello richiesto da specifiche

Nell'architettura proposta i cavi di interconnessione tra hub e board possono essere in rame o in fibra ottica. Dei vantaggi e degli svantaggi di ciascuna delle due opzioni si discuterà in seguito. Nell'immagine sottostante viene proposto lo schema rappresentante la prima architettura.

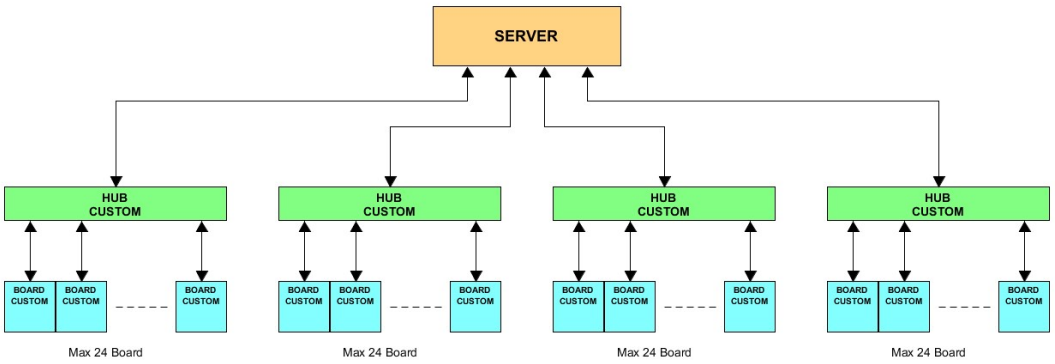


Figura 8 prima architettura proposta

Come protocollo PCIe richiesto tra server e hub, è necessario scegliere il PCIeGen3 ad x8. Infatti dovendo raggiungere un data rate di 19Gbyte/s in 4 schede, è necessario un protocollo che supporti un data rate di 4.75.

Quello che risponde alla richiesta, anche con un margine di tolleranza, è appunto il PCIeGen3 ad x8.

	PCI-e 1.0 [MByte]	***PCI-e 2.x*** [MByte]	***PCI-e 3.0*** [MByte]	***PCI-e 4.x*** [MByte]
x1	250	500	985	1969
x4	1000	2000	3940	7876
x8	2000	4000	7880	15752
x16	4000	8000	15760	31504

Tabella 2 caratteristiche dei differenti protocolli PCIe

Procedendo con le architetture, viene proposta una seconda soluzione, sempre single-server. Per diminuire il numero di canali in uscita dall'hub, questa architettura suggerisce come possibile soluzione la gestione di più board in cascata. In questo caso però, gestendo le card in cascata, il numero di canali in uscita dall'hub sarà inferiore a scapito della banda dati richiesta che aumenterà di conseguenza, inoltre saranno necessari due canali di trasmissione/ricezione per ciascuna board. Di seguito viene riportata la tabella con le specifiche dell'architettura:

SECONDA ARCHITETTURA	
Nr. SERVER	1
Nr. HUB	4
PROTOCOLLO SERVER-HUB RICHIESTO	PCIe 3.0ad x8
Nr. BOARD PILOTATE (xHUB)	24 (4 transceiver channel x6 board)
FLUSSO DATI SERVER	152 Gbit/s
FLUSSO DATI HUB	38 Gbit/s
FLUSSO DATI PER CANALE	9.5 Gbit/s
COMPATIBILITA' HUB-ARRIA 10 GX	Si
COMPATIBILITA' BOARD-CYCLONE 10 GX	Si

Tabella 3 specifiche del sistema ottenute nell'ipotesi di utilizzo della seconda architettura

Lo schema sottostante è rappresentativo del sistema impostato in questo modo.

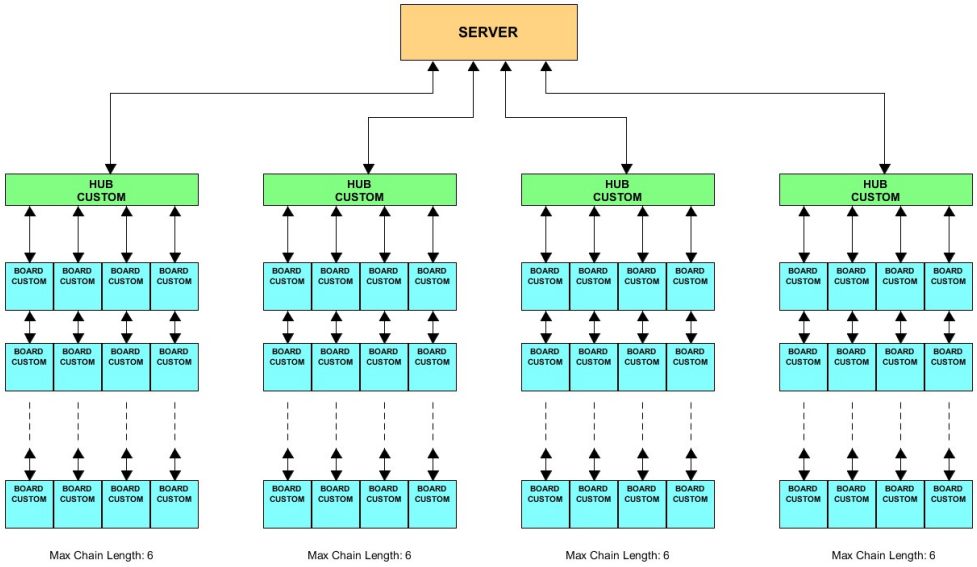


Figura 9 seconda architettura proposta

Anche in questo caso con lo stesso ragionamento del caso precedente, è necessaria la scelta del protocollo PCIeGen3 ad x8.

L'ultima soluzione proposta è multi-server. Ogni server alimenta 4 hub, che a sua volta pilota un massimo di 12 board ciascuno. Ancora una volta, si riporta nella tabella seguente il calcolo delle specifiche che ne consegue, e successivamente lo schema relativo.

TERZA ARCHITETTURA	
Nr. SERVER	2
Nr. HUB	8
PROTOCOLLO SERVER-HUB RICHIESTO	PCIe 3.0 (x4)
Nr. BOARD PILOTATE (xHUB)	12(12 transceiver channel)
FLUSSO DATI SERVER	76Gbit/s
FLUSSO DATI HUB	16Gbit/s
FLUSSO DATI PER CANALE	1.6 Gbit/s
COMPATIBILITA' HUB-ARRIA 10 GX	Si
COMPATIBILITA' BOARD-CYCLONE 10 GX	Si

Tabella 4specifiche del sistema ottenute nell'ipotesi di utilizzo della terza architettura

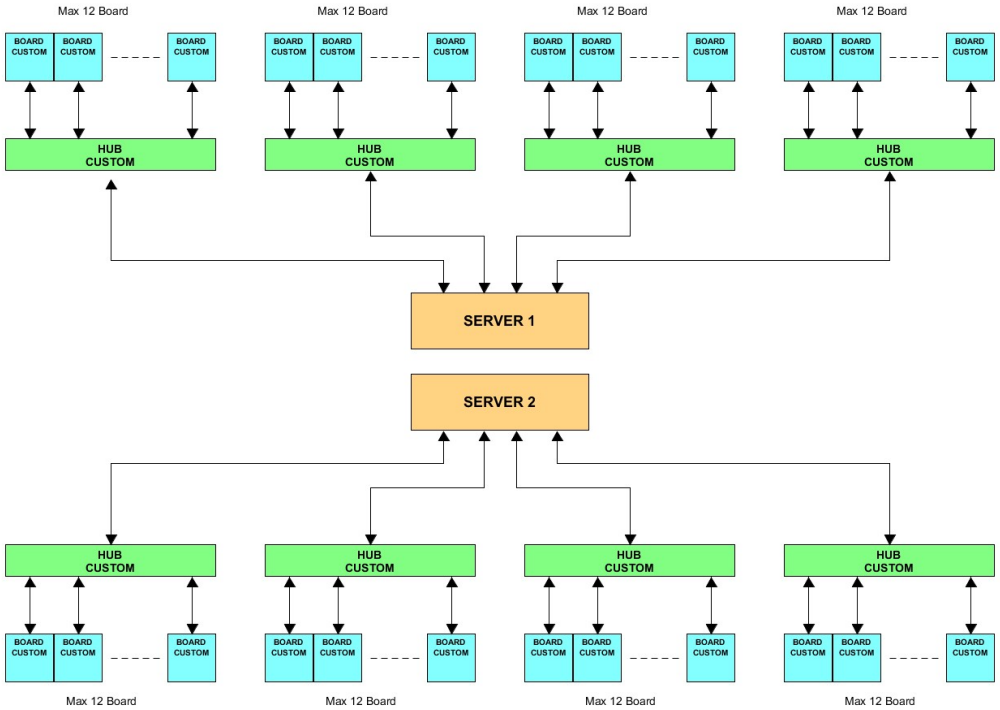


Figura 10 terza architettura proposta

Con questa architettura è possibile utilizzare un protocollo di comunicazione PCIeGen 3.0 x4 tra Server ed Hub.

2.2.2 CONFRONTO

Al fine di confrontare le varie soluzioni proposte in precedenza, riportiamo in una tabella tutte le caratteristiche per tutti i casi di studio:

	ARCHITETTURA 1	ARCHITETTURA 2	ARCHITETTURA 3
NR. SERVER	1	1	2
NR. HUB	4	4	8
PROTOCOLLO SERVER-HUB	PCIe 3.0ad x8	PCIe 3.0ad x8	PCIe 3.0 (x4)
NR.BOARD(xHUB)	24 (channel)	4 (channel)	12 (channel)
FLUSSO DATI SERVER	152 Gbit/s	152 Gbit/s	76 Gbit/s
FLUSSO DATI HUB	38 Gbit/s	38 Gbit/s	16 Gbit/s
FLUSSO DATI PER CANALE	1.6 Gbit/s	9.5 Gbit/s	1.6 Gbit/s
NR. TRANSCEIVER PER HUB	32	12	16
COMPATIBILITA' Hub-Arria10 GX	si	si	Si
COMPATIBILITA' Board-Cyclone 10 GX	si	si	si

Tabella 5. Confronto tra le architetture.

L’architettura 3 proposta risulta essere la soluzione ottimale per la realizzazione del nuovo sistema, perché suddividendo il flusso dati sui due server, essi risultano meno sovraccaricati rispetto al caso 1 e 2, anche perché è necessario l’utilizzo di un minor numero di canali per scheda. Occorre però considerare che la scelta dell’architettura multi-server implica un ulteriore layer di comunicazione, quello tra i server. Un discorso analogo può essere fatto per gli hub, che in questo modo dovranno gestire un numero di canali non troppo oneroso, poiché in questa configurazione in ogni hub sono necessari un minimo di 16 trasceiver (12 channel board + 4 channel PCIe). Questo ci permette di poter progettare il sistema con delle specifiche non particolarmente stringenti per la tecnologia utilizzata e soprattutto lo rende migliorabile in futuro (in termini di banda e numero di canali di comunicazione verso le board).

2.3 CANALE DI COMUNICAZIONE FPGA-FPGA

Un aspetto molto importante da considerare è la scelta della tecnologia per il canale fisico di comunicazione. I cavi dovranno essere idonei alla specifiche di progetto, in modo da non creare un collo di bottiglia nel sistema.

2.3.1 CONNETTORI QSFP+ E SFP+

Per quanto riguarda la tipologia di trasmissione scelta tra quelle possibili nel DevKit della FPGA, è stato scelto il QSFP+ e l’SFP+.

L’SFP (Small Form-factor Pluggable) è un modulo di transceiver ottico compatto e hot-pluggable utilizzato sia nelle applicazioni di telecomunicazioni sia per la comunicazione di dati come in questo caso. I transceiver SFP supportano una serie di standard quali SONET, Gigabit Ethernet e Fibre Channel. Il QSFP(Quad Small Form-factor Pluggable) si basa sugli stessi principi di funzionamento dell’SFP, in particolare permette di trasmettere i dati tramite un'unica interfaccia ad alta densità ed arrivare a 4 SFP separati. In questo modo le informazioni si diramano nei 4 canali di trasmissione solamente a ridosso della ricezione, evitando in buona parte problemi di disturbo e delay tra le linee che si sarebbero invece presentati nel caso di altri standard offerti dalla scheda.

Le versioni aggiornate ed equipaggiate sulla demo-board sono SFP+ e QSFP+. Esse supportano delle data rate di 10/40Gbps. I

moduli SFP+ e QSFP+ hanno le stesse dimensioni di quelli standard, e consentono dunque di riutilizzare i design già impostati per i moduli base. Per queste ragioni si è deciso di utilizzarli nel progetto. Esiste un'altra versione dell'SFP, ovvero l'SFP28, che sta prendendo piede nel settore. I connettori SFP28 sono disegnati per raggiungere 25 Gbps in trasmissione, pur avendo lo stesso design degli SFP+. Essi sono inoltre meno soggetti a crosstalk, e hanno un miglior controllo dell'impedenza. I connettori SFP28 possono essere inseriti anche nelle porte designate per gli SFP+, e non è dunque necessario nel caso si decida di passare da uno standard all'altro modificare anche le porte utilizzate. Essi sono dunque da tenere in considerazione per degli sviluppi futuri del progetto in cui potrebbero essere richiesti dei data rate più elevati.

Una volta scelto lo standard da utilizzare, la scelta da fare è stata quella tra due tipologie di cavi differenti: un cavo in fibra ottica e uno in rame.

2.3.1.1 FIBRA OTTICA

Le fibre ottiche sono filamenti vetrosi o polimerici realizzati in modo da poter condurre al loro interno la luce. Esse sono classificate come guide d'onda dielettriche basate sulla disomogeneità del mezzo il cui nucleo è sede del campo elettromagnetico. Possono dunque convogliare e guidare al loro interno un campo elettromagnetico di frequenza alta con perdite limitate. Nel caso dei cavi, essi sono immuni ai disturbi elettrici ed alle condizioni atmosferiche più estreme. Ogni singola fibra ottica è composta da due strati concentrici di materiale trasparente estremamente puro: un nucleo cilindrico centrale, detto core, e un mantello o cladding che lo ricopre. Il core ha dimensioni molto minori del cladding, ed entrambe le parti hanno dei diametri dell'ordine di micrometri. Al suo interno, la fibra funziona similmente a uno specchio tubolare. La luce che entra nel core ad un certo angolo, detto angolo limite, si propaga mediante una serie di riflessioni alla superficie di separazione fra i due materiali del core e del cladding. Infine, all'esterno della fibre vi è una guaina protettiva o jacket, che fornisce resistenza a stress fisici, corrosione nonché protezione dall'ambiente esterno. Nei cavi possono essere presenti una o più fibre ottiche (fino a 7) all'interno di un'unica protezione.

Il cavo in fibra ottica presenta inoltre un interfaccia opto-elettronica che permette la trasduzione immediata dell'informazione elettrica a quella ottica già all'interno del connettore. A questo proposito infatti l'interfaccia contiene anche l'alimentazione che rende possibile la trasduzione del segnale, rendendo di fatto l'interfaccia esterna del cavo uguale a quella di un normale cavo in rame. A una certa distanza dal cavo (lungo 30 metri), vi è poi uno splitter che dirama il segnale in 4 cavi secondari terminanti ciascuno su una testina diversa SFP, non più ad alta densità. I dati che vengono trasmessi all'interno delle fibre ottiche contenute dal cavo, non sono bounded tra loro, ovvero si possono inviare dati differenti per ciascun singolo canale.

Figura 11 Cavo di 30 metri in fibra ottica da QSFP a 4x SFP tipo FinisarFCBN510QE2C30

Il cavo in fibra come menzionato precedentemente, pur essendo indubbiamente più delicato rispetto ad altri cavi, porta con se diversi vantaggi. Primo fra tutti, l'immunità al rumore che affligge invece quasi tutti i tipi di cavi differenti nonostante la schermatura. Inoltre, la fibra non è sensibile ad eventuali variazioni di temperatura, vantaggio non indifferente nel caso di utilizzo in ambienti che potrebbero surriscaldarsi. Infine, la fibra potrebbe teoricamente accoppiarsi con campi magnetici esterni tuttavia, essendo la fibra scelta multimodale e quindi meno soggetta al backscattering di Rayleight, essa non dovrebbe presentare problemi di questo genere nell'ambiente nella quale si andrà a posizionare. Dalle diverse prove effettuate coi due cavi, che si illustreranno in seguito, alla fine è risultato più conveniente l'utilizzo di questa tipologia di cavo.

2.3.1.2 CAVI IN RAME

Il cavo in rame è sicuramente più utilizzato. Esso senza dubbio è più economico del cavo in fibra. Inoltre, è meno delicato e sensibile ai piegamenti o agli urti e non necessita dell'alimentazione all'interno dell'interfaccia che invece il cavo in fibra richiede per la trasduzione, di conseguenza porterebbe a un dispendio minore di potenza.

Figura 12 Cavo di 2 metri in rame da QSFP a 4x SFP tipo TE Connectivity 2821033-2

Tuttavia, il cavo in rame è sensibile alla temperatura, e di conseguenza potrebbe essere oggetto in maggior misura ad errori o malfunzionamenti durante la trasmissione. Inoltre, anche se schermato, il rame non presenta l'insensibilità quasi totale che la fibra ha al rumore. È dunque un altro aspetto da considerare nella sua valutazione.
Infine, la tecnologia basata sul rame risulta compatibile con data rate minori piuttosto che la fibra. Di conseguenza nonostante le prove in fase di progetto siano state effettuate con entrambi i cavi , il rame non è stato scelto.

3. STUDIO E SVILUPPO DEL CANALE DI COMUNICAZIONE FPGA-FPGA

Una volta ripristinata la condizione di default della scheda ed effettuate le considerazioni già descritte nei capitoli precedenti sui cavi di connessione e sulle specifiche richieste, si è potuto procedere con la fase di test e progettazione vera e propria. Di seguito quindi, verranno illustrati i blocchi base necessari per lo sviluppo del sistema ed i vari test effettuati per verificarne le prestazioni in termini di qualità del canale di comunicazione.
Infine si tratterà il tema della sincronizzazione del canale, descrivendo il sistema implementato per raggiungere la configurazione adottata nel progetto finale e la descrizione delle macchine a stati che gestiscono automaticamente il flusso di dati vero e proprio, anche tramite protocollo.

3.1 BLOCCHI BASE DEL SISTEMA

Intel mette a disposizione all'interno di Quartus dei blocchi fisici nativi "Native Phy Transceiver" che permettono di spingersi a basso livello sia sulla configurazione che sul funzionamento dei transceiver. Questi blocchi necessitano di una particolare logica dedicata rappresentata da un "Reset Controller" (RC) e da un "Phase Lock Loop" (PLL).

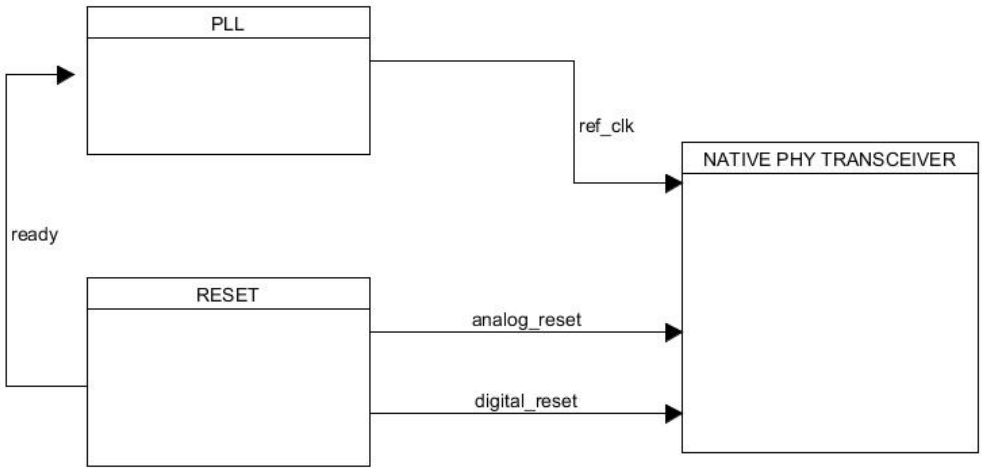


Figura 13 schema generale di come interagiscono i blocchi di RC,PLL,nativo

Il PLL è un blocco fisico dedicato per i transceiver, che permette di rielaborare i clock di riferimento in input e fornire in uscita il clock desiderato da inviare ai blocchi di trasmissione e ricezione, insieme a una serie di segnali di chiamata per il RC, al fine di sincronizzare il blocco con il sistema.

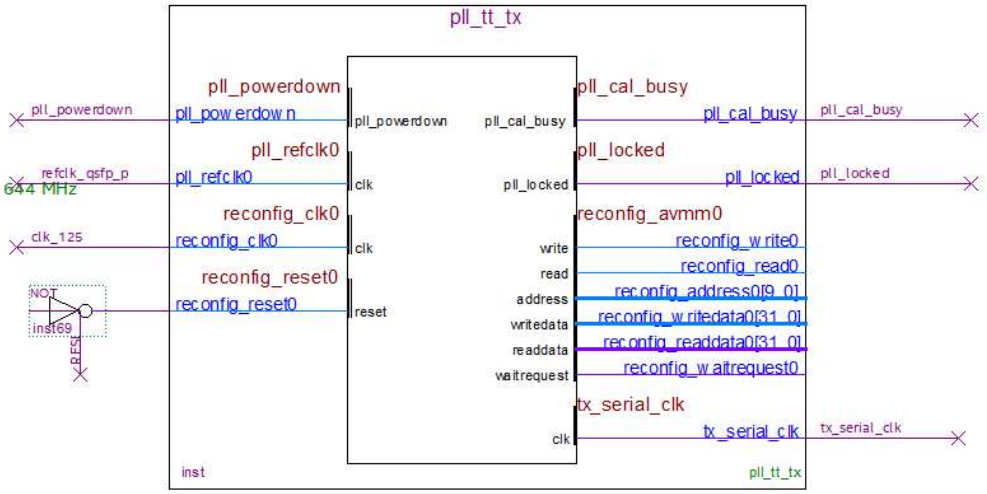


Figura 14 blocco di PLL

Per il PLL sono disponibili diversi parametri configurabili, tra cui ad esempio la frequenza del clock in uscita diretto ai transceiver.

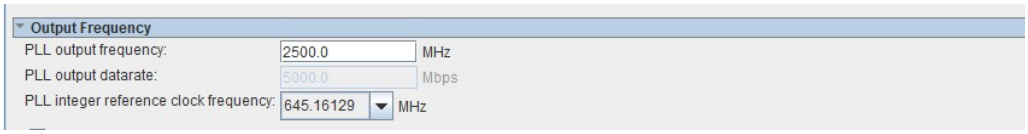


Figura 15 impostazione della frequenza di clock in output

Come si può notare in figura, viene richiesta in concomitanza della frequenza del clock in uscita anche la frequenza del clock di riferimento. Nel kit di sviluppo è possibile modificare questa frequenza tramite il Board Test System (BTS), un tool fornito da Intel che consente diverse operazioni dirette sul Development Kit, in modo da rispettare i vincoli imposti dalla frequenza di output che si vuole ottenere dal PLL.

Per quanto riguarda il Reset Controller (RC), esso si occupa della gestione dei segnali di sincronizzazione dei vari blocchi legati ai transceiver, utilizzando segnali di "*ready*" e di "*reset*" che

raggiungono tutti i blocchi del sistema. Anche in questo caso viene fornito nella libreria di Quartus e presenta diversi parametri che possono essere configurati in base alle esigenze progettuali.

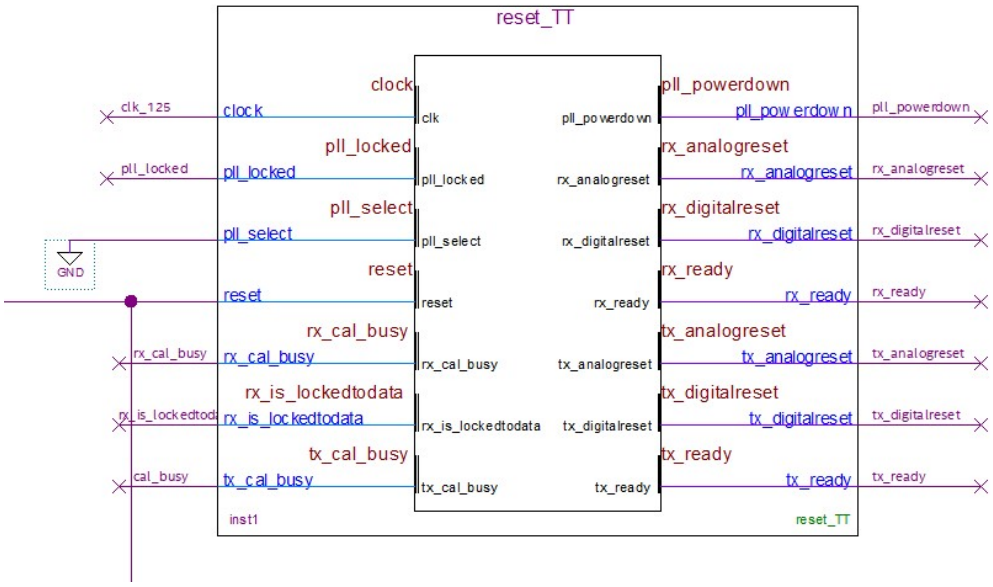


Figura 16 blocco di Reset

L'utilizzo di questo blocco è estremamente vantaggioso perché permette di resettare e sincronizzare i vari blocchi del sistema in contemporanea, evitando così i mismatch dovuti ai differenti domini di clock che ciascun blocco utilizza, riducendo la possibilità di errori legati ai timing.
L'unico parametro che è stato necessario adattare al sistema, è stato il clock di riferimento.

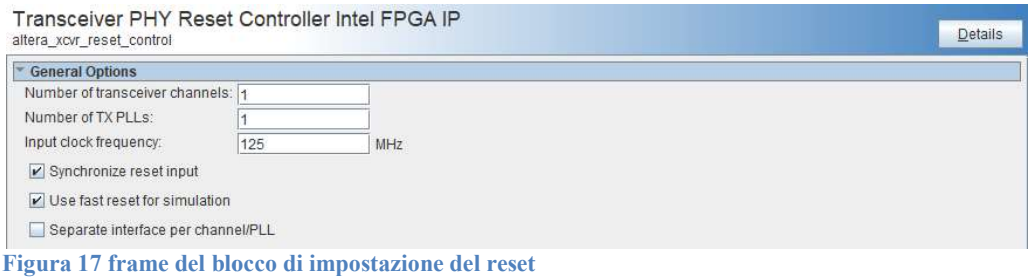


Figura 17 frame del blocco di impostazione del reset

Il clock di riferimento infatti, è un clock dedicato e deve essere compatibile con l'input del blocco di Reset. In questo caso, è stato scelto un clock a 125 MHz disponibile sull'Arria Development Kit.

Una volta impostati i blocchi di RC e PLL, si può procedere alla configurazione dei blocchi nativi per la ricezione e la trasmissione

per mezzo di transceiver. E' possibile utilizzare i blocchi nativi in due modalità differenti:

- Gestione della trasmissione o della ricezione del canale di comunicazione (saranno necessari 2 blocchi per sviluppare una trasmissione TX/RX)
- Gestione della trasmissione e della ricezione del canale di comunicazione.

Per i primi test ad un solo canale, si è preferito utilizzare i due blocchi separati che simulassero quelli nelle due schede finali.

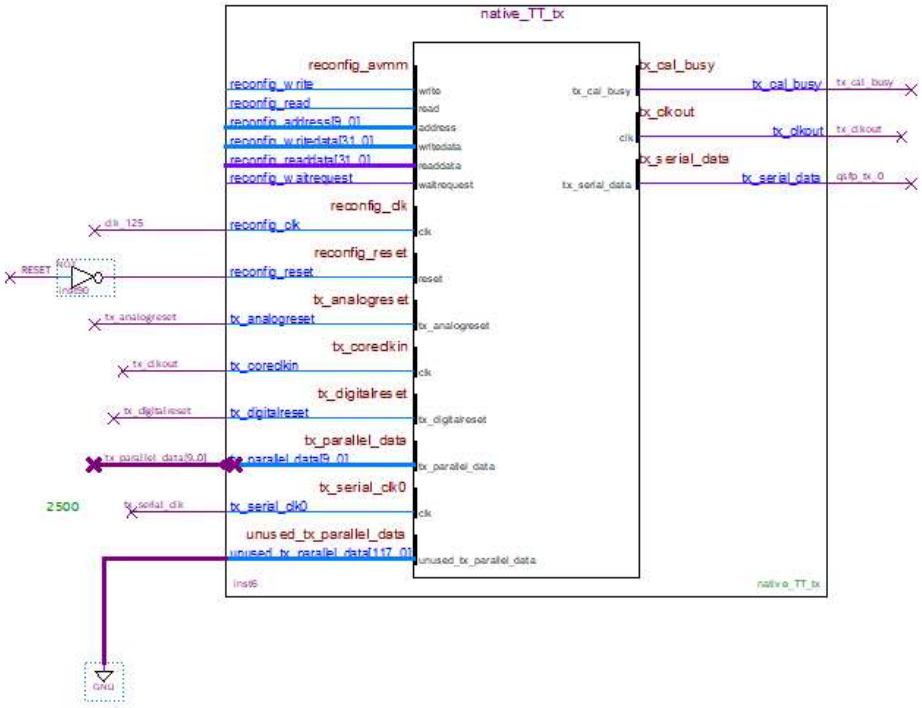


Figura 18 blocco di trasmissione

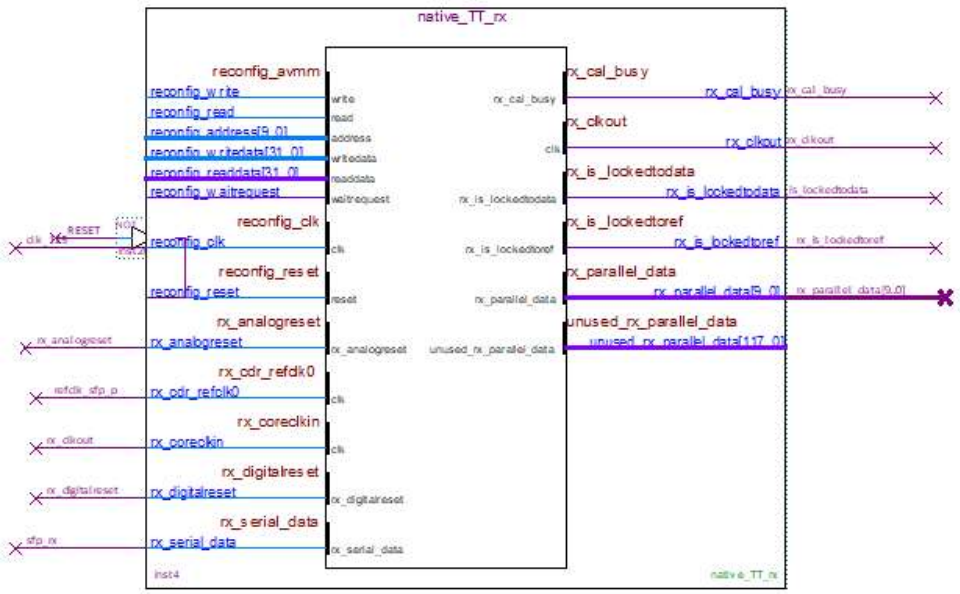


Figura 19 blocco nativo di ricezione

Questi blocchi si occupano della trasformazione parallelo-serie dei dati e viceversa, con lo scopo di inviarli alla trasmissione vera e propria sul canale fisico.

Alcune impostazioni dei blocchi risultano le medesime sia per il blocco di trasmissione, sia per il blocco di ricezione, in quanto legate allo stesso canale dati in comune. Al fine di rendere la trattazione più lineare possibile, si procederà ad illustrare in primis le impostazioni che i due blocchi condividono per poi procedere con le peculiarità di ciascuno dei due. In entrambi i blocchi è possibile selezionare il tipo di configurazione dei transceiver, che spazia dal PCS, al 10GBEthernet, al PCIe. In questo caso è stato scelto il PCS, una configurazione di base, sia nella sua versione *Standard*, sia nella sua versione *Enhanced*.

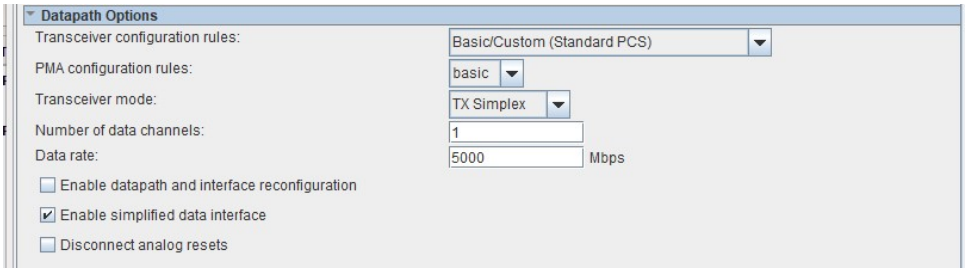


Figura 20 impostazioni data rate blocchi di trasmissione

Come si può notare in figura, è inoltre possibile selezionare il data rate desiderato e la tipologia di nativo desiderata (TX/RX oppure singoli). Il "*simplified data interface*" invece, se abilitato gestisce automaticamente una serie di segnali avanzati non necessari allo scopo di questo progetto.

Sia il blocco di trasmissione sia quello di ricezione, hanno in ingresso una serie di segnali di "*reset*" gestiti dal RC, oltre ad un clock interno che gestisce il campionamento dei dati sull'interfaccia parallela affinché il blocco raggiunga il data rate desiderato. Questo clock (*tx_clkout* o *rx_clkout*) viene generato automaticamente e chiuso in loopback sul blocco stesso. La frequenza del clock è data dalla seguente formula:

$$Tx_{clkout} = Rx_{clkout} = \frac{datarate}{numero\ di\ bit\ interfaccia\ parallela - seriale}$$

Il solo blocco di trasmissione necessita di un clock (*tx_serial_clk_0)* dedicato alla serializzazione dei dati lungo il canale che gli viene fornito dal PLL (*tx_serial_clk*).

Una caratteristica particolare del blocco di ricezione invece, è la necessità di settare la frequenza del clock di riferimento. E' stata scelta quella di default sul Development Kit per il banco dei transceiver del canale SFP+.

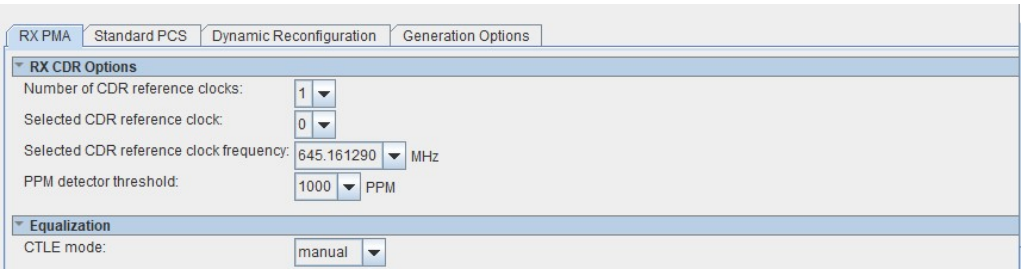


Figura 21 parametri specifici del blocco RX

Il sistema così ottenuto, composto dai due blocchi di trasmissione e ricezione, il PLL e il RC, rappresenta il progetto di base per eseguire le prime prove di trasmissione dati.

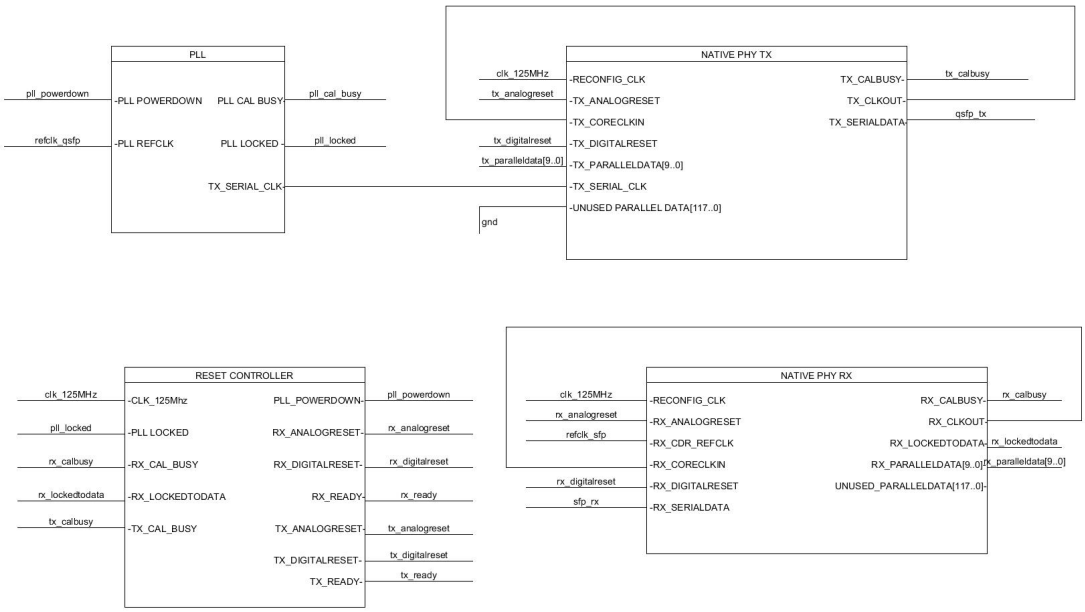


Figura 22 visione complessiva del progetto utilizzato nelle prove con il Transceiver Toolkit. Sono stati omessi i segnali di riconfigurazione per maggiore leggibilità

3.1.1 ANALISI DI QUALITA'DEL CANALE CON CAVI IN RAME E FIBRA OTTICA

Il primo step nella progettazione, è stato quello di effettuare un'analisi di qualità dei cavi in rame ed in fibra ottica per verificarne le prestazioni a differenti bit rate. A questo scopo è stato utilizzato il sistema base di trasmissione/ricezione dati utilizzando il Transceiver Toolkit(TT), un tool dedicato ai transceiver.

Il Transceiver Toolkit permette di eseguire una riconfigurazione dinamica che consiste nella modifica automatica di determinati parametri di equalizzazione e bilanciamento dei canali da parte del sistema, in modo tale da individuare la combinazione che garantisce risultati migliori in termini di prestazioni. Per fare questo, viene instaurato un meccanismo di trasmissione/ricezione dati interno,che

progressivamente modifica i parametri utili (quali, per esempio, l'equalizzazione del canale) e verifica la qualità della trasmissione, fornendo in uscita le diverse combinazioni ed i relativi indici di qualità. Nella versione attuale di Quartus, all'interno del TT la quasi totalità dei parametri è stata automatizzata, è possibile però decidere a che tipo di test sottoporre la scheda ed alcuni settaggi legati al voltaggio della scheda. Il TT è in grado infatti di istanziare un sistema che permette di inviare in "loopback"diverse sequenze pseudo-random (PRBS7, PRBS9, PRBS15, PRBS23, PRBS31). Queste sequenze si basano ciascuna su un differente algoritmo, ognuno ottimizzato per un particolare tipo di test:

PRBS Pattern	10 bit PCS-PMA width	64 bit PCS-PMA width
PRBS7: $x^7 + x^6 + 1$		Yes
PRBS9: $x^9 + x^5 + 1$	Yes	Yes
PRBS15: $x^{15} + x^{14} + 1$		Yes
PRBS23: $x^{23} + x^{18} + 1$		Yes
PRBS31: $x^{31} + x^{28} + 1$		Yes

Figura 23 tabella che riassume le differenti tipologie di test PRBS e la rispettiva sequenza

Il test PRBS7 e il test PRBS9 sono utilizzati per testare i link tra i transceiver con problemi di tipo lineare e con la codifica 8B/10B. Il PRBS15 invece viene utilizzato per la valutazione del jitter. Infine, i test PRBS23 e PRBS31 sono impiegati sempre per la valutazione del jitter (jitter dipendente dai dati) ma anche dei link senza codifica 8B/10B come ad esempio SDH/SONET/OTN. Ciascuno di questi test è stato effettuato sul sistema implementato in questo progetto. Occorre menzionare la necessità di utilizzare l'*enhanced* PCS,nel caso delle prove con il TT, per le prove che superavano i 5 Gbps di data rate, in quanto lo *standard* non risulta abbastanza prestante per i test automatici effettuati da questo tool per data rate superiori a quella menzionata. In concomitanza all'utilizzo dell' *enhanced* occorreva modificare anche l'interfaccia a 64 bit e non più a 16. Una volta presi in considerazioni i sistemi successivi, che sono svincolati dal TT, è stato poi possibile implementare il sistema con lo *standard* PCS con tutti i data rate scelti.

Il TT esegue uno "swap" dei parametri. Successivamente sarà possibile usufruire dei risultati per integrarli manualmente al progetto sviluppato. Naturalmente, nei blocchi all'interno di questo progetto è stato necessario abilitare la riconfigurazione dinamica in modo che fossero abilitate le configurazioni interne necessarie per

l'utilizzo del TT. Inoltre, per ciascun blocco è stato abilitato anche un secondo parametro, l'ADME o Altera Debug Master Endpoint, che fornisce accesso alla riconfigurazione nel register space all'interno dei canali dei transceiver e dei PLL. Esso permette alla System Console di identificare automaticamente e controllare i canali dei transceivers in maniera diretta. Tutti gli altri parametri disponibili per la riconfigurazione dinamica nei blocchi invece non erano necessari allo scopo di testare il canale di comunicazione tramite il Transceiver Toolkit.

Il parametro di riferimento utilizzato per lo studio della qualità dei materiali dei cavi in esame all'interno del TT è la Bit Error Rate (BER).
La BER evidenzia quanto della originaria trasmissione viene perso o giunge distorto all'apparecchio ricevente. Essa è qualitativamente legata al rapporto segnale/rumore (SNR) in maniera inversa: tanto maggiore è la BER rilevata, tanto minore è il rapporto SNR e viceversa. La sua definizione matematica è data dalla formula seguente:

$$BER = \frac{numero\ BIT\ errati}{numero\ BIT\ emessi}$$

Per eseguire uno studio completo sulla qualità della linea di trasmissione dati, si è deciso di rilevare la BER fino ad arrivare alla massima banda possibile per i transceiver: 12.5 Gbps.

Le prove con il cavo in rame ed in fibra ottica hanno mostrato evidenti differenze in termini prestazionali.
La fibra ha mantenuto una BER a 0 fino a 12.5 Gbps, ovvero la banda massima dell'Arria 10 GX, mentre il rame ha mostrato un calo di prestazioni a 7,4 Gbps in alcuni test fino ad arrivare a una BER>0 per tutti i test intorno ai 9,2 Gbps. Inoltre è da considerare che il cavo in fibra utilizzato per le prove era lungo 30 metri, mentre quello in rame solamente 2 metri. Di conseguenza su una più lunga distanza, paragonabile a quella utilizzata per la fibra, è possibile che il rame abbia prestazioni ancora peggiori rispetto a quelle rilevate con i test effettuati in questo progetto.

Di seguito si riassumono i risultati dei test in una tabella:

Data Rate	BER Rame	BER Fibra
5 Gbps	0	0
6.4 Gbps	0	0
7.4 Gbps	BER PRBS31>0	0
9,2Gbps	BER all_PRBS>0	0
9.5Gbps	BER all_PRBS>0	0
10 Gbps	BER all_PRBS>0	0
12.5 Gbps	BER all_PRBS>0	0

Tabella 6 risultati dei test sui cavi in fibra e rame. Inizialmente solo alcuni test falliscono per quanto riguarda il rame, fino ad arrivare a una BER >0 per tutti i test ai quali è stato sottoposto

Sulla base di queste prove si è deciso di utilizzare i cavi con tecnologia in fibra ottica. Questo permette di avere un margine maggiore nel caso di future modifiche in velocità del sistema e di conseguenza poter sfruttare appieno le potenzialità della scheda. È possibile visualizzare i risultati dei test con Transceiver Toolkit per mezzo di un report che può essere anche esportato.

3.2 IMPLEMENTAZIONE DELLA TRASMISSIONE DATI SUL CANALE

Una volta chiara la struttura del sistema si è proceduto all'implementazione di una trasmissione dati monodirezionale e successivamente ad una trasmissione dati bidirezionale.

Sono stati quindi inseriti i blocchi di trasmissione/ricezione, il blocco del PLL e del Reset Controller. In questo caso non è stata abilitata la riconfigurazione dinamica perché non necessaria in quanto non utilizzeremo il TT ma invieremo dei dati custom. A tal proposito è stato necessario creare due nuovi blocchi, uno che si occupasse di alimentare i dati in trasmissione ed uno legato ai dati ricevuti con funzione "*checker*", che ha la funzione di sincronizzazione e validazione dei dati.

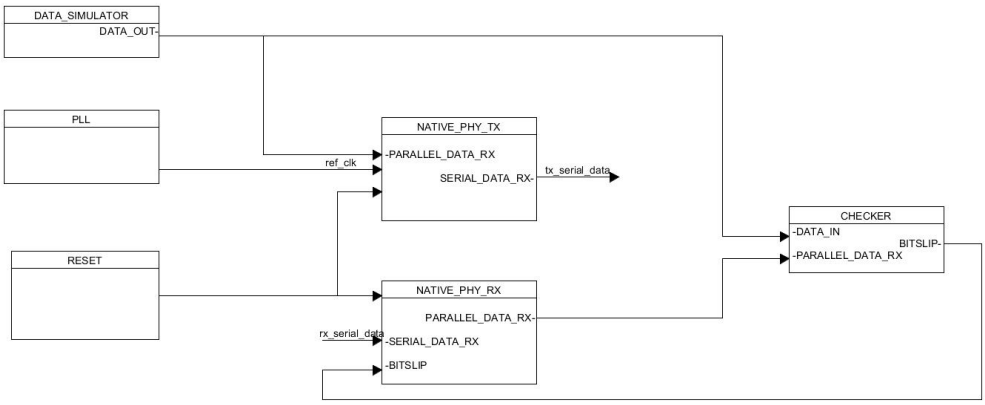


Figura 24 schema generale del sistema implementato con PLL,RC,nativo RX e TX, il simulatore di dati e il checker.

E' necessario l'utilizzo di un *checker* nel sistema in quanto il ricevitore ricostruisce i dati tramite il clock di riferimento e l'arrivo dei dati stessi. Di conseguenza generalmente i dati non sono ricostruiti in fase e la sequenza viene ricostruita in maniera errata, solitamente "shiftata" rispetto a quella inviata. Esiste però un segnale, definito come *bitslip*, che permette di shiftare di uno o più bit alla volta la sequenza, e quindi sincronizzare i dati ricevuti. In questo caso si è deciso di shiftare la sequenza di un bit alla volta per maggior precisione. Il segnale di bitslip (attivo alto), viene abilitato dal *checker* ogni qualvolta i dati da esso ricevuti non corrispondano a quelli di riferimento forniti al blocco, ovvero i dati inviati dal blocco di trasmissione. E' d'obbligo che il segnale di *bitslip* resti attivo due cicli di clock e successivamente sia disabilitato per 20 cicli di clock (clock di riferimento *rx_clkout*) per permettere che i dati shiftati abbiano un tempo sufficiente per stabilizzarsi. È stato necessario abilitare il *bitslip* all'interno del blocco nativo di ricezione nella propria configurazione.

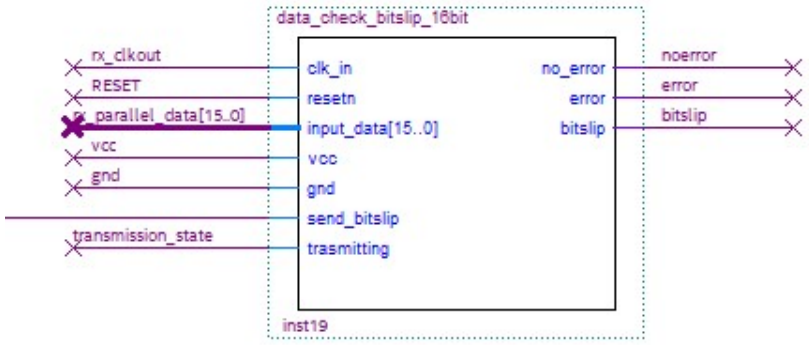


Figura 25 blocco di checker implementato per la verifica dei dati ricevuti

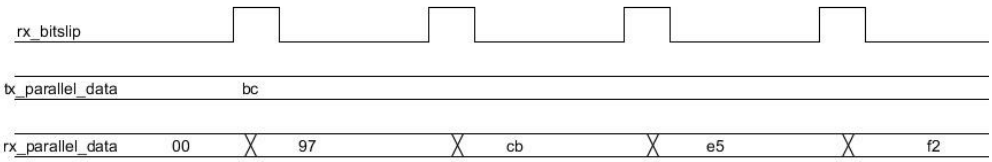


Figura 26 schema di funzionamento del segnale rx_bitslip

Come accennato,una possibile alternativa al *bitslip* sul ricevitore è quella di shiftare i dati in ingresso al trasmettitore, ove è possibile effettuarlo a più di un bit alla volta. Per fare ciò, è necessario abilitare un'ulteriore porta del blocco di ricezione detta *tx_std_bitslipboundarysel*. In quel caso, viene indicato il numero di bit dei quali si vogliono shiftare i dati in ingresso in concomitanza di questa porta.

3.2.1 COMUNICAZIONE MONODIREZIONALE

Il primo sistema impostato implementa la trasmissione dati tramite un canale monodirezionale. È stato deciso di direzionare la trasmissione dal primo pin del QSFP a quello SFP. Quindi il blocco di trasmissione verrà collegato al primo dei due pin, quello di ricezione all'altro. L'interfaccia scelta per il canale è a 20 bit, e il clock di riferimento è stato impostato a 400 MHz in modo da rispettare i vincoli posti dalla data rate scelta pari a 10 Gbps.

Sono stati impostati sia il blocco di trasmissione sia quello di ricezione con protocollo base PCS *standard*.

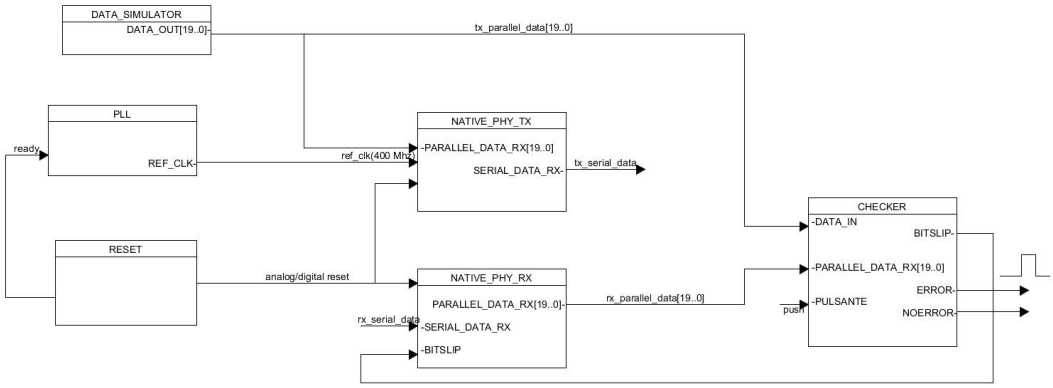


Figura 27 schema generale del sistema implementato

Il primo step che è stato implementato per mezzo di questo sistema, è la trasmissione di una sequenza fissa di bit generata dal simulatore di dati e trasmessa ai nativi . Il checker forniva in uscita sia il segnale di bitslip, sia il segnale di errore o non errore per avere un feedback sulla coerenza dei dati ricevuti.
In queste prime implementazioni, il segnale del *bitslip* è stato generato manualmente grazie all'utilizzo di un pulsante disponibile sulla board di sviluppo, in modo da osservare più agevolmente ogni shift imposto fino all'allineamento corretto.
Dopo aver caricato il progetto sulla scheda e premendo il pulsante associato alla procedura di *bitslip*, si è potuto verificare dunque che effettivamente il sistema così progettato era correttamente funzionante in quanto dopo alcuni shift si raggiungeva il match tra i dati ricevuti e quelli inviati.

E' interessante avere un riscontro visivo dei segnali real time che viaggiano all'interno della scheda. Per ottenerli è possibile utilizzare un tool fornito da Quartus denominato Signal Tap Analyzer. Impostando un clock di campionamento ed inserendo il nome del pin o del wire di cui si vuole sapere il valore, si possono acquisire i segnali durante il normale funzionamento della scheda, monitorandone il valore nel tempo. Nel caso di questo progetto in particolare, è possibile osservare il valore dei dati prima e dopo il raggiungimento della configurazione corretta mediante l'utilizzo del segnale di *bitslip*. Di seguito vengono riportate le schermate del Signaltap prima e dopo che il segnale fosse stato sincronizzato.

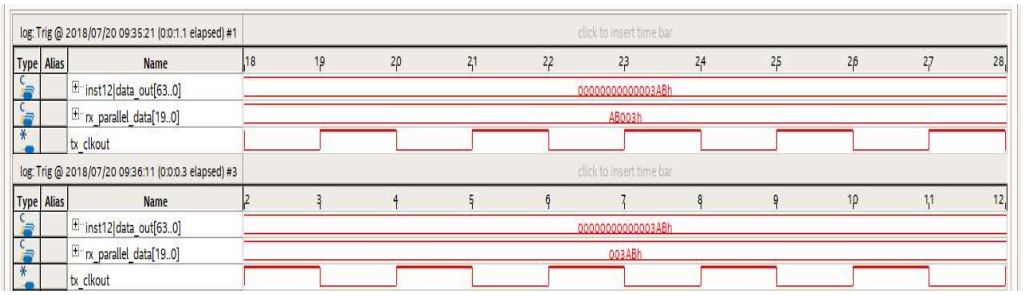


Figura 28 segnali prima e dopo il raggiungimento del valore corretto mediante bitslip

Si è quindi dimostrato il corretto funzionamento del primo sistema di prova trasmissione dati utilizzando i blocchi nativi standard PCS.

Queste prove sono state effettuate anche utilizzando dei pacchetti di dati variabili. Il blocco di trasmissione dati in questo caso generava pacchetti di dati a pacchetti con tutti i bit a 0. In assenza di una codifica sul canale, si è dimostrato che dopo solo circa 100 pacchetti di zeri consecutivi il sistema tendeva a perdere il sincronismo e necessitava di una nuova procedura di sincronizzazione tramite *bitslip*.
Le sequenze di soli zeri sono state scelte in quanto per la loro struttura intrinseca portano sbilanciamento all'interno del canale. Seppur equalizzato infatti, il canale si trova in condizioni critiche dal momento in cui si comincia ad inviare un gran numero di zeri o di uni poiché i transceiver utilizzano anche la sequenza dati trasmessa per ricostruire il clock di trasmissione.

Figura 29perdita del lock dopo una sequenza di zeri

A questo scopo, è stato ritenuto necessario l'utilizzo della codifica 8B/10B. Con la codifica 8B/10B vengono mappate parole di 8 bit in 10 simboli per raggiungere il bilanciamento del canale e la bounded disparity. Con questa codifica la differenza tra il conteggio di uni e zeri in una stringa di almeno 20 bit non è più di due e non ci possono più essere configurazioni di svariati uni o zeri consecutivi. In questo modo si rende il segnale trasmesso molto più stabile. È stato integrata questa codifica al sistema con lo scopo di mantenere il *lock* dei dati con qualsiasi sequenza possibile.

Ovviamente come suggerisce il nome, i bit trasmessi, se si selezionano 20 bit per canale come nel progetto precedente,saranno multipli di 8 (in questo caso 16) più dei bit di controllo (in questo caso 4) che servono a segnalare il corretto funzionamento della trasmissione dei dati.
Naturalmente il checker dovrà analizzare non più 20 bit ma 16, in quanto i bit di controllo aggiuntivi sono inseriti e gestiti all'interno del blocco nativo, e non sono quindi più disponibili per la trasmissione di dati vera e propria. Il protocollo 8B/10B porta dunque con sè un overhead del 20%, e quindi per ogni 10 bit mandati, 8 saranno effettivamente disponibili per la trasmissione dati, due invece saranno di controllo.

I blocchi che variano in maniera più consistente sono quello di trasmissione e di ricezione, che come detto a causa della codifica 8B/10B spezzano i 20 bit che arrivano in 16 bit di dati più quelli di controllo.
Di seguito vengono riportati i due blocchi:

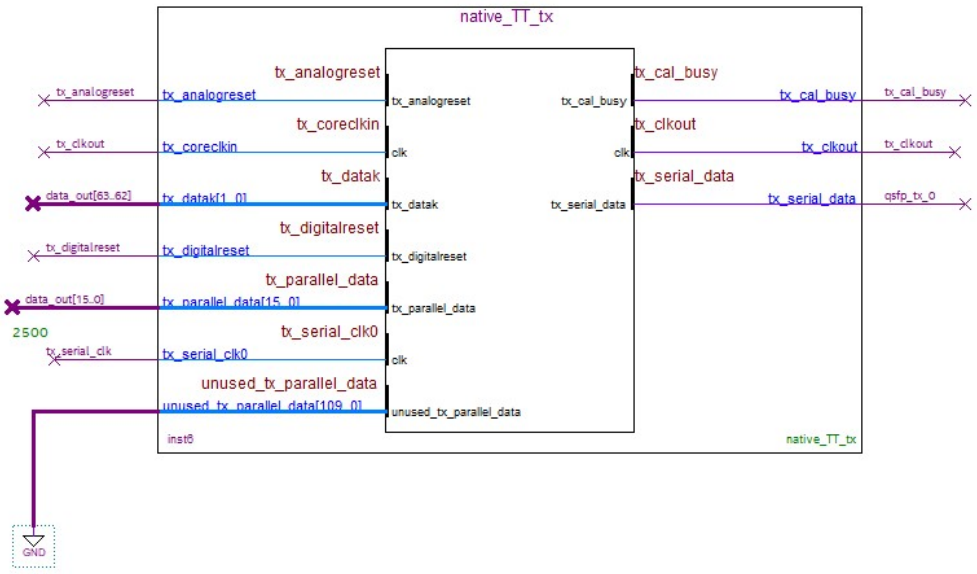


Figura 30 blocco di trasmissione nativo

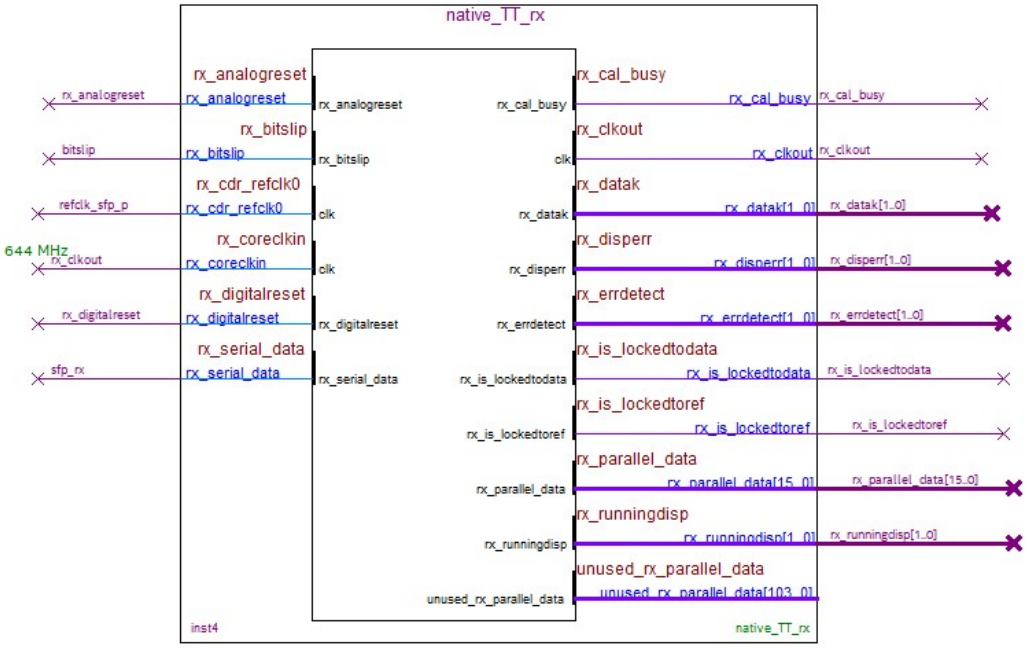


Figura 31 blocco di ricezione nativo

Come si può osservare, nel blocco di ricezione appaiono, oltre al *bitslip* che rimane presente finché si utilizza lo standard PCS, anche dei bit aggiuntivi in uscita:

- *rx_datak* è attivo alto se arrivano i comandi dal blocco di trasmissione. È invece a zero nel caso arrivino dei dati.
- *rx_disperr* segnala se c'è stato un errore di disparità in trasmissione.
- *rx_errordetect* invece segnala se ci sono stati errori in generale e di che tipo di errore si tratta.
- *rx_running_disp* indica se i dati sono stati ricevuti con disparità negativa o positiva, in modo da sapere in caso di mismatch come modificarli per ottenere il pacchetto corretto.

Questi flag sono estremamente utili per monitorare il corretto funzionamento dei blocchi di trasmissione e ricezione, e di conseguenza intervenire dopo la segnalazione di uno degli errori o incongruenze sopra elencati.

Anche in questo caso, si può verificare il corretto funzionamento del sistema mediante l'utilizzo del SignalTap, che ci permette di visualizzare i segnali real time.
Come si può vedere dalle schermate riportate, i segnali raggiungono grazie al bitslip il lock dei dati.

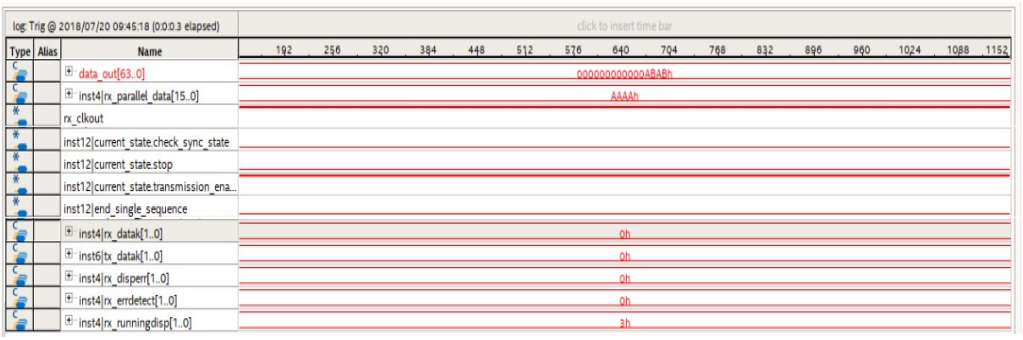


Figura 32 segnali prima del conseguimento del lock dei dati tramite bitslip

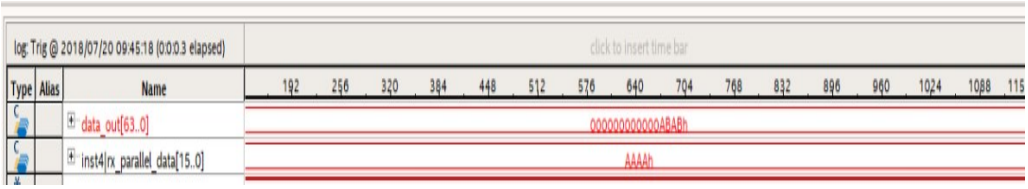


Figura 33 ingrandimento dei dati trasmessi e ricevuti prima del raggiungimento del lock tramite bitslip

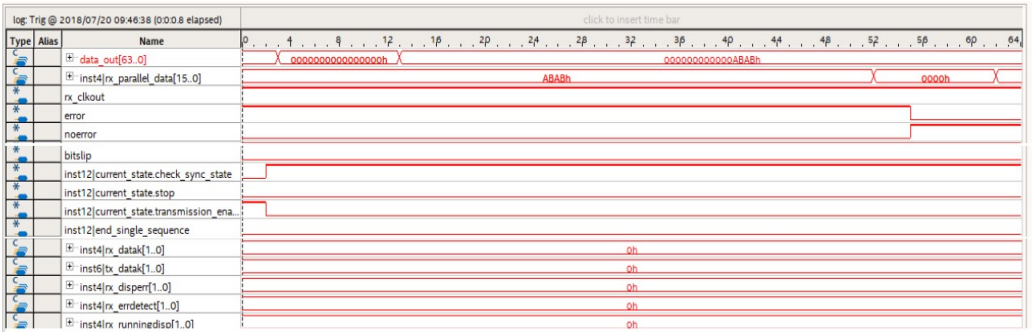


Figura 34signaltap dopo il raggiungimento del lock dei dati inviando pacchetti alternati con pattern costanti o con serie di zeri

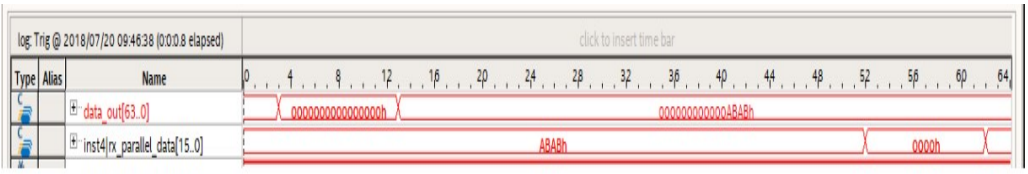


Figura 35 invio dei pacchetti alternati anche con zeri ripetuti per sequenze lunghe

Una prova che si è rivelato utile effettuare, è stata quella di visualizzare se effettivamente dopo l'invio di pacchetti di zeri la comunicazione rimanesse stabile. Nel caso del progetto precedente infatti, una volta inserita la sequenza di zeri il sistema perdeva il lock dei dati ed era necessario ripristinare una condizione di stabilità. In questo caso invece come è possibile osservare nella figura sottostante, i dati vengono trasmessi e ricevuti correttamente anche al termine della sequenza di zeri inserita, confermando così il miglioramento nella comunicazione introdotto per mezzo della codifica 8B/10B.

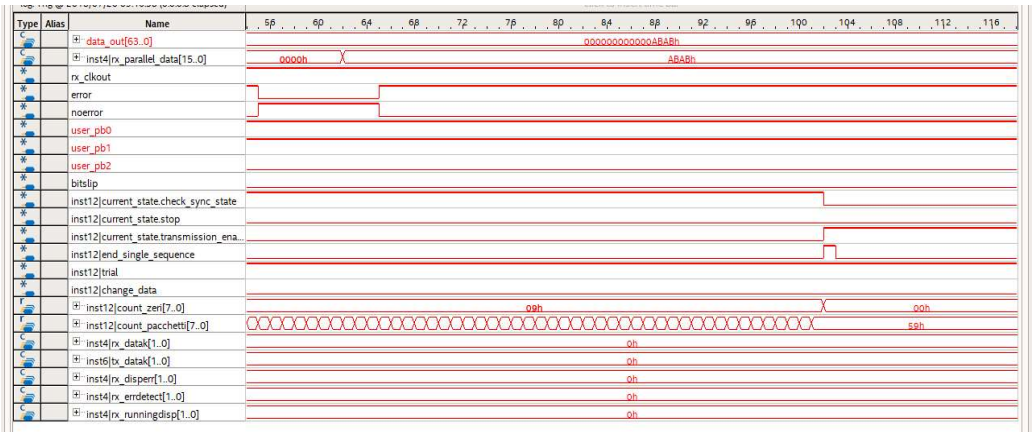


Figura 36 come si può notare, i dati tornano al lock anche dopo l'invio di pacchetti di zeri

Allo stesso modo è stato verificato su questo progetto il comportamento del sistema nel caso del cavo di ricezione disconnesso. Una eventuale disconnessione porta in uscita al sistema dei dati nulli (pacchetti di zeri). Questa informazione è di fondamentale importanza, in quanto una volta progettato il sistema finale, si potrebbe captare un eventuale malfunzionamento dovuto a un evento di questo genere semplicemente controllando che non arrivi un determinato numero di sequenze di zeri consecutivi in ricezione, operazione che risulterebbe decisamente meno agevole qualora il sistema ricevesse in questo caso dei dati spuri privi di logica.

In questo modo si sono verificati i comportamenti del sistema in corrispondenza sia a una ripetizione di zeri in trasmissione, sia nel caso di cavo disconnesso. Dal momento che il sistema di trasmissione risulta funzionante nella trasmissione da un nativo all'altro (che nel progetto finale saranno su due schede differenti),

si può passare al progetto di prova finale che prevede anche la trasmissione/ricezione inversa.

3.2.2 COMUNICAZIONE BIDIREZIONALE

Nel progetto per testare la trasmissione e ricezione bidirezionale sono stati utilizzati due blocchi nativi TX/RX, sempre disponibili nella libreria di Quartus, in modo che ciascuno di essi potesse sia trasmettere sia ricevere. Uno è stato considerato come master del sistema (quello che sarà poi interno al Hub) e uno come slave (quello a controllo della Board).

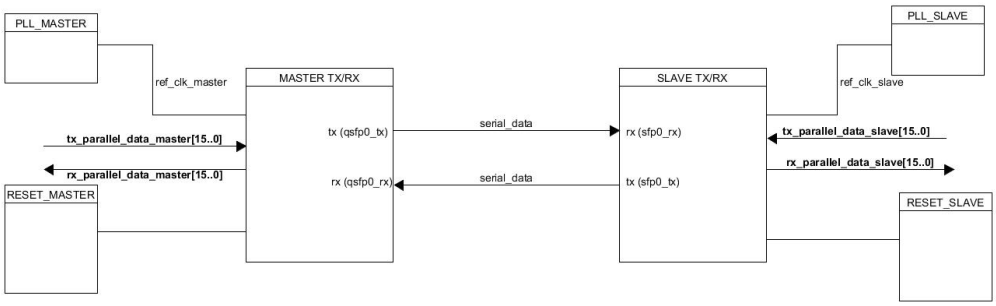


Figura 37 schema del progetto per la trasmissione bidirezionale

I blocchi inizialmente sono stati configurati entrambi con la codifica 8B/10B a 5 GBps, con un clock di riferimento a 400 MHz. Successivamente il sistema è stato portato fino ai 10 GBps.
Essendo sia il blocco del PLL sia il RC blocchi dedicati a un particolare nativo, è stato necessario per il nuovo sistema inserirne due di ciascuno, ognuno dedicato alla sincronizzazione e al timing o del master o dello slave. È importante sottolineare questo aspetto in primo luogo poiché questi due blocchi come detto saranno dislocati in schede differenti, in secondo luogo in quanto durante la progettazione successiva del meccanismo di sincronizzazione occorrerà considerare che i blocchi si potranno sincronizzare solo separatamente, rendendo necessario un protocollo di comunicazione descritto in seguito.
Naturalmente, è stato necessario raddoppiare anche i blocchi di trasmissione dei dati e quelli di *checker*. Ciascuno di essi viene gestito dai clock dedicati da ciascun blocco nativo (trasmissione/ricezione del master,trasmissione/ricezione dello slave).

Per completezza, vengono qui di seguito riportati i blocchi nativi master e slave, entrambi con la configurazione TX/RX, che rappresentano la parte del sistema che più viene modificata in sede di comunicazione bidirezionale.
Per la comunicazione bidirezionale naturalmente sono stati mappati anche i pin di trasmissione del SFP+ e i pin di ricezione del QSFP+, come si può notare nei blocchi. È stata implementata la trasmissione dal canale QSFP a quello SFP e viceversa. Una volta testato il corretto funzionamento del sistema per il canale 0 si mapperanno anche i canali 1/2/3 del QSFP per la trasmissione verso il canale SFP di ciascuna delle 4 schede nelle rispettive board.

Per quanto riguarda i segnali e i flag presenti nell'interfaccia dei blocchi nativi invece, essi sono esattamente gli stessi descritti in precedenza per la comunicazione monodirezionale. È importante ricordare che incrociando due sistemi di trasmissione/ricezione, sia nel master sia nello slave sarà presente il pin di *bitslip* per il controllo dei dati ricevuti sia da uno, sia dall'altro blocco.

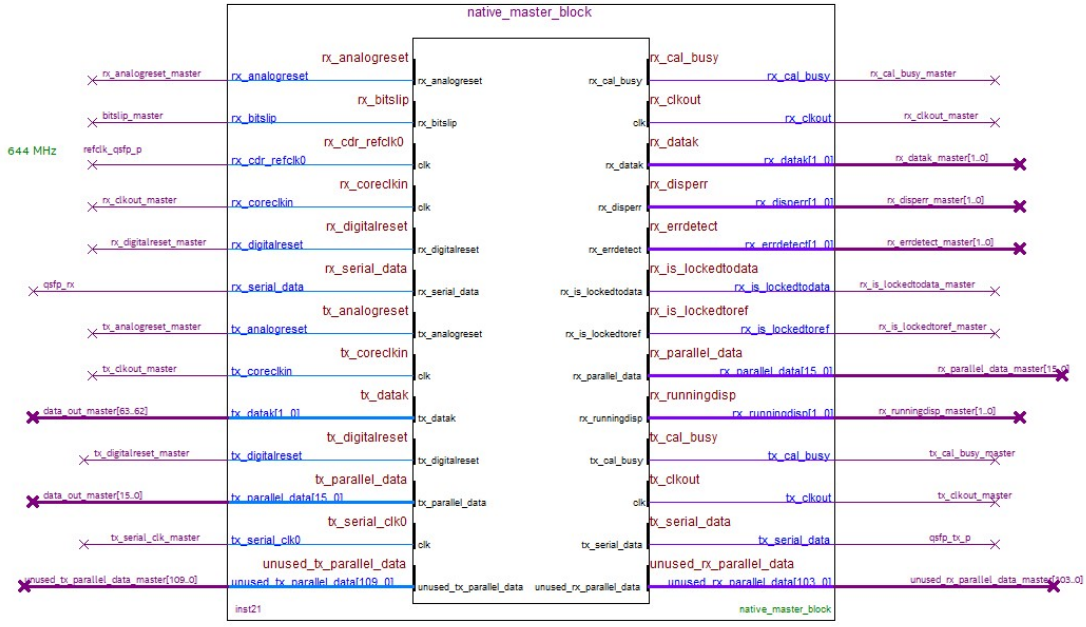


Figura 38 blocco master di trasmissione e ricezione MASTER

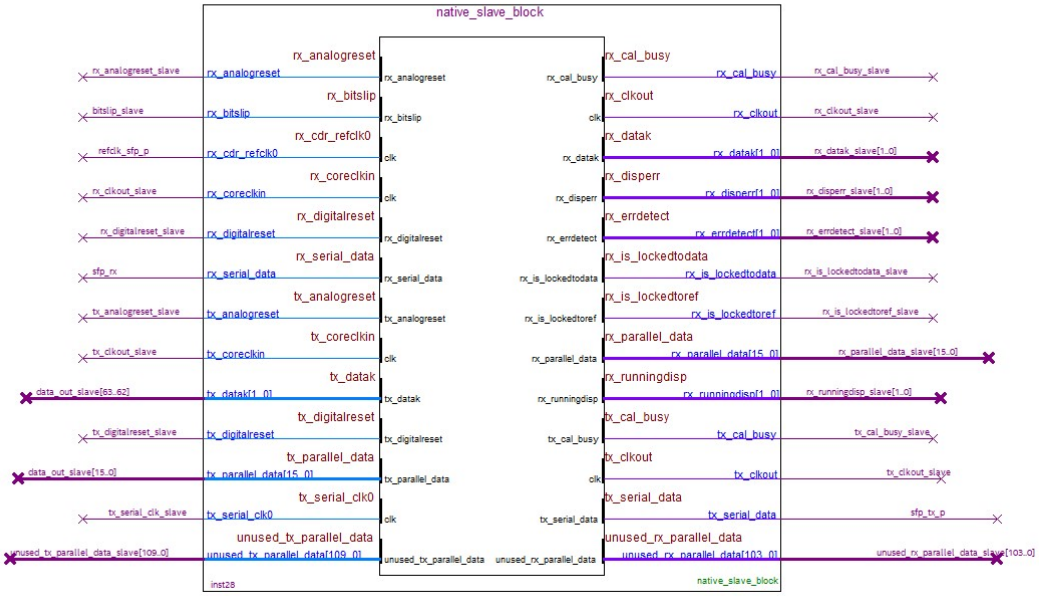


Figura 39 blocco slave di trasmissione e ricezione SLAVE

Un accorgimento che però è necessario adottare affinché la trasmissione bidirezionale sia possibile, è l'attivazione del pin *sfp_tx_disable*. Questo pin legato al connettore SFP+ permette di abilitare o meno la trasmissione dati sull'SFP+. È dunque necessario durante l'impostazione del progetto abilitare il pin mettendolo a massa.

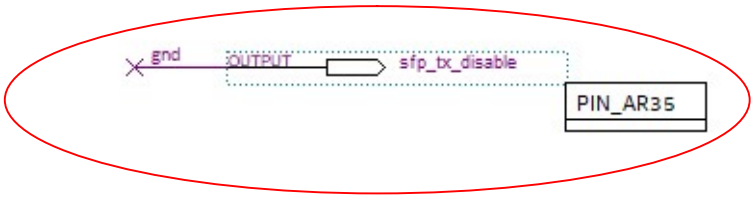


Figura 40 pin da fissare a gnd per abilitare la trasmissione bidirezionale

Una volta preso questo accorgimento, se si carica il sistema descritto all'interno della scheda FPGA tramite programmer, si può verificare che la trasmissione avvenga in entrambe le direzioni. Inoltre tramite i *bitslip* si riesce a raggiungere il *lock* dei dati per ciascuno dei blocchi di ricezione (master e slave). Di seguito si riportano le schermate del SignalTap. L'analisi è stata effettuata per entrambe le direzioni di comunicazione.

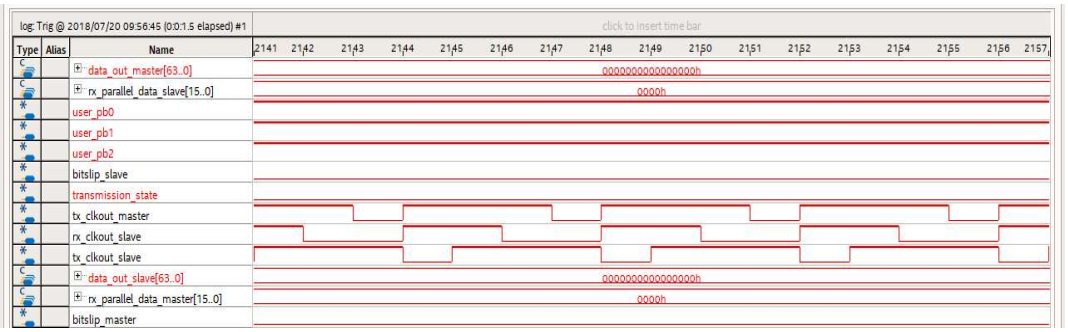


Figura 41signaltap senza trasmissione

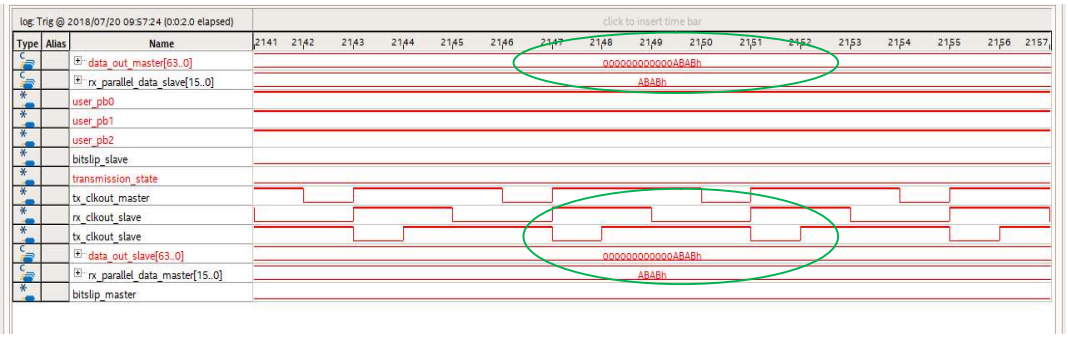


Figura 42signaltap durante la trasmissione bidirezionale. si può vedere che in entrambe le direzioni i dati vengono trasmessi correttamente

Una volta verificato il corretto funzionamento del sistema con comunicazione bidirezionale sarà necessario procedere con la creazione di un sistema per la gestione delle sincronizzazioni dei vari blocchi (master e slave) considerando che i due sistemi saranno su due board differenti e quindi richiederanno un meccanismo dedicato.

3.2.3 ANALISI DELLE PRESTAZIONI

Il progetto di simulazione così ottenuto soddisfa pienamente le specifiche iniziali del sistema in termini di banda utile, in quanto risulta di 8 Gbps. L'utilizzo del protocollo 8B/10B necessario per la stabilità della trasmissione, introduce però come già accennato anche il relativo overhead di 20% che influisce negativamente sulla banda, portandola a 8Gbps dai 10 Gbps configurati.

Tipologia di trasmissione	Utilizzo protocollo	Data rate	Bit di interfaccia	Encoding 8B/10B	Mantenimento lock del canale in casi critici	Meccanismo di sincronizzazione automatica
Monodirezionale	No	10 GBps	20	No	Non garantito	No
Monodirezionale	No	10 Gbps (8 effettivi)	16+4	Si	Garantito	No
Bidirezionale	No	10 Gbps (8 effettivi)	16+4	Si	Garantito	No

Figura 43 tabella riassuntiva dei sistemi implementati

Occorre considerare che nel progetto implementato non è stato ancora inserito un protocollo di comunicazione, ed essa avviene quindi con dati grezzi. Per il progetto finale questo tipo di comunicazione non è possibile, in quanto non c'è modo di individuare dopo l'avvenuta sincronizzazione eventuali problemi nella trasmissione dei dati. Si renderà dunque necessario nell'impostazione completa del progetto inserire un protocollo che gestisca anche questa eventualità.

3.3 GESTIONE DELLA SINCRONIZZAZIONE DEL CANALE TRA DISPOSITIVI FPGA-BASED

Nella versione del progetto sopra descritta, i blocchi vengono comandati manualmente per raggiungere la configurazione desiderata. Infatti non è stato ancora implementato un meccanismo di sincronizzazione automatico che tenga conto della completa sincronizzazione del sistema e quindi delle condizioni di stabilità nelle quali è necessario che si trovi per il suo corretto funzionamento. Prendendo a riferimento l'architettura descritta precedentemente, il canale di comunicazione che si sta prendendo in considerazione è quello tra l'hub e le board. L'hub ha la funzione di smistare i dati sulle board e di inviare ad essa comandi, di conseguenza sarà il master del sistema. Le board invece ricevono i dati con un flusso quasi solo monodirezionale dall'hub, ad eccezione di qualche comando che segue il flusso inverso. Di conseguenza, la board sarà lo slave nel sistema implementato.

3.3.1 LE MACCHINE A STATI

Per rendere automatico il sistema mediante un meccanismo di sincronizzazione tra i blocchi sono state create 4 macchine a stati, una per ciascun blocco di trasmissione e ricezione, sia del master sia dello slave, che si occupano di gestire e verificare il flusso dati in entrambe le direzioni allo scopo di ottenere un sistema con sincronizzazione automatica.

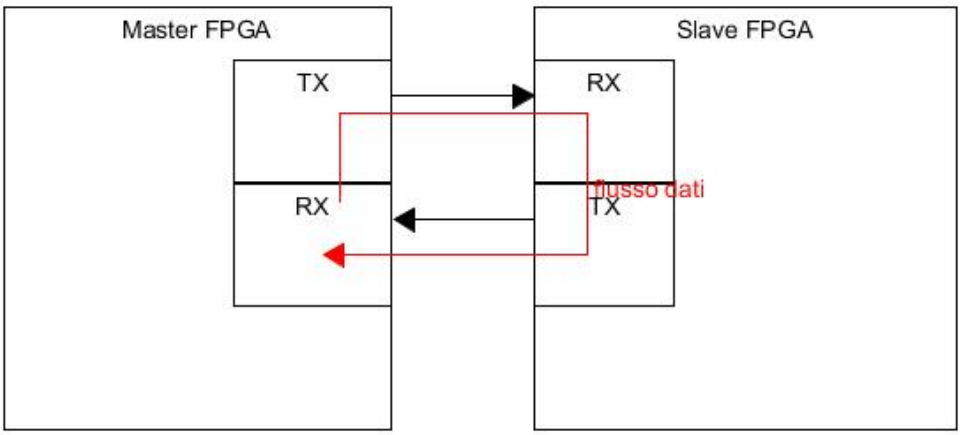


Figura 44 schema del flusso dati desiderato. come si può osservare, è necessario richiudere il loopback

Lo scopo del sistema così implementato, è quello di creare un *loopback* di verifica dei dati che parta dal blocco di trasmissione master e si richiuda su se stesso, passando per tutti gli altri blocchi come evidenziato in figura.
Per svolgere questa funzione, ciascuna macchina a stati gestisce e analizza i dati in trasmissione/ricezione in modo che il flusso sia completamente controllato, gestendo dei flag per monitorare e informare il sistema sul processo in corso fino a giungere alla completa sincronizzazione del sistema.

Nelle prossime sezioni verranno descritti i blocchi di verifica e di gestione dei dati uno per volta, descrivendo lo scopo ed il funzionamento in dettaglio di tutte le macchine a stati. Di seguito si riportano lo schema generale che si vuole implementare e gli step di sincronizzazione dei blocchi.

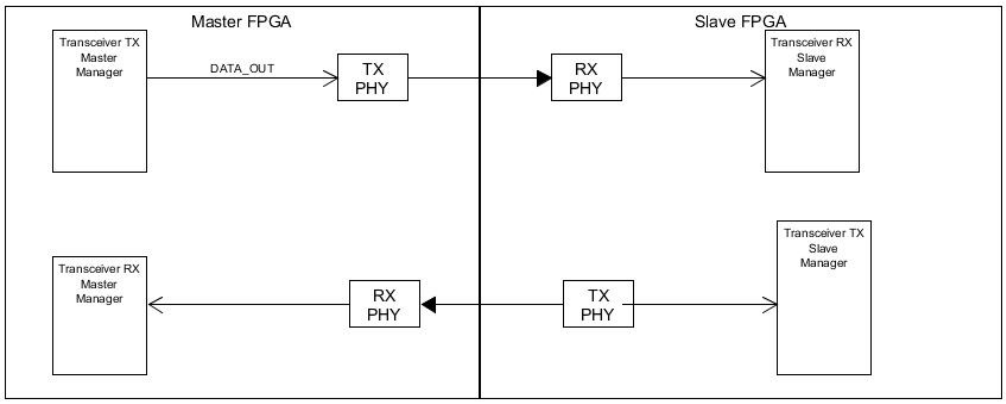


Figura 45 sistema finale complessivo dei blocchi

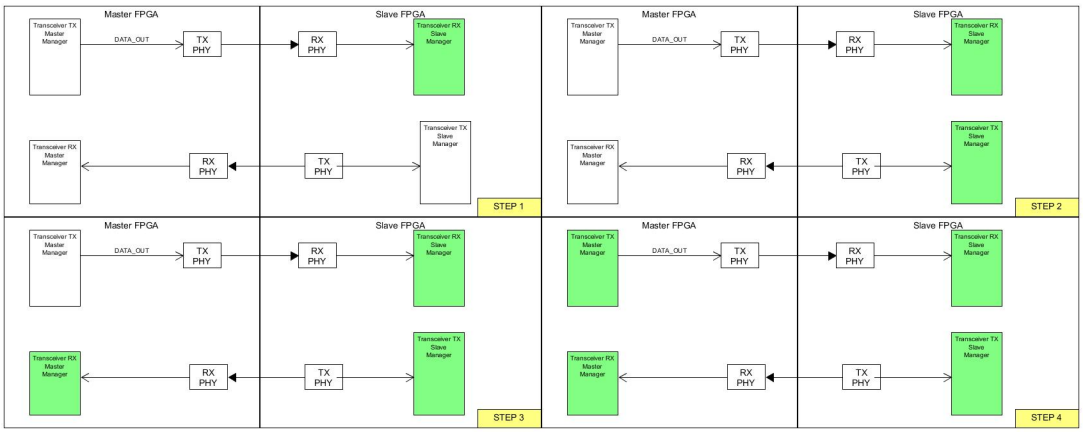
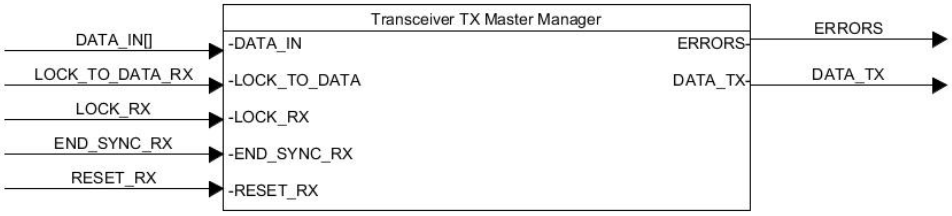


Figura 46 step di sincronizzazione del sistema implementato

Il primo blocco che si deve sincronizzare è il gestore ricevitore slave, per mezzo di sequenze di dati che arrivano dal gestore trasmettitore master, di cui si parlerà più dettagliatamente nel capitolo dedicato. Dopodiché si deve sincronizzare il gestore trasmettitore slave, che a sua volta invia sequenze specifiche per la sincronizzazione del gestore ricevitore master, terzo negli step di sincronizzazione. Infine, il loop si richiude con la sincronizzazione del gestore trasmettitore master che deve ricevere una particolare sequenza dal ricevitore master. Se tutti gli step vengono eseguiti dal sistema correttamente, esso risulta sincronizzato in ogni sua parte. Si procede ora nella descrizione dettagliata di ciascun blocco.

3.3.1.1 GESTORE TRASMETTITORE MASTER

Il gestore trasmettitore master è il blocco che si occupa della gestione dei dati in trasmissione da parte del master. Esso viene inserito tra il simulatore dati e il blocco nativo fisico di trasmissione.



47. Blocco Transceiver TX Master Manager

Come si può osservare in figura, i segnali di input e output del blocco sono i seguenti:

- *DATA_IN* : dati che vengono trasmessi attraverso il transceiver tx master manager. Nel progetto implementato sono 16 bit.
- *LOCK_TO_DATA*: verifica il lock dei dati. È un segnale che viene prodotto dal blocco fisico del transceiver stesso quando è in grado di ricevere dati. Questo segnale è utile, ad esempio, nel caso il cavo si disconnettesse.
- *LOCK*: verifica la permanenza del lock dei dati nei canali. Qualora i dati si disallineassero per qualche motivo dopo aver raggiunto il lock, il segnale lo rileverebbe.
- *END_SYNC*: da conto della terminata sincronizzazione del blocco che lo invia.
- *RESET*: segnale che ripristina la condizione di default del blocco al quale viene inviato.

Questi segnali, presenti nel gestore trasmettitore master, regolano l'intero meccanismo di sincronizzazione e sono quindi presenti anche negli altri blocchi inseriti per lo scopo.

I dati che si desiderano inviare vengono dunque prima filtrati e gestiti da questo blocco. Come si può notare dallo schema in figura, la macchina a stati che gestisce questo step della trasmissione consta di 5 stati.

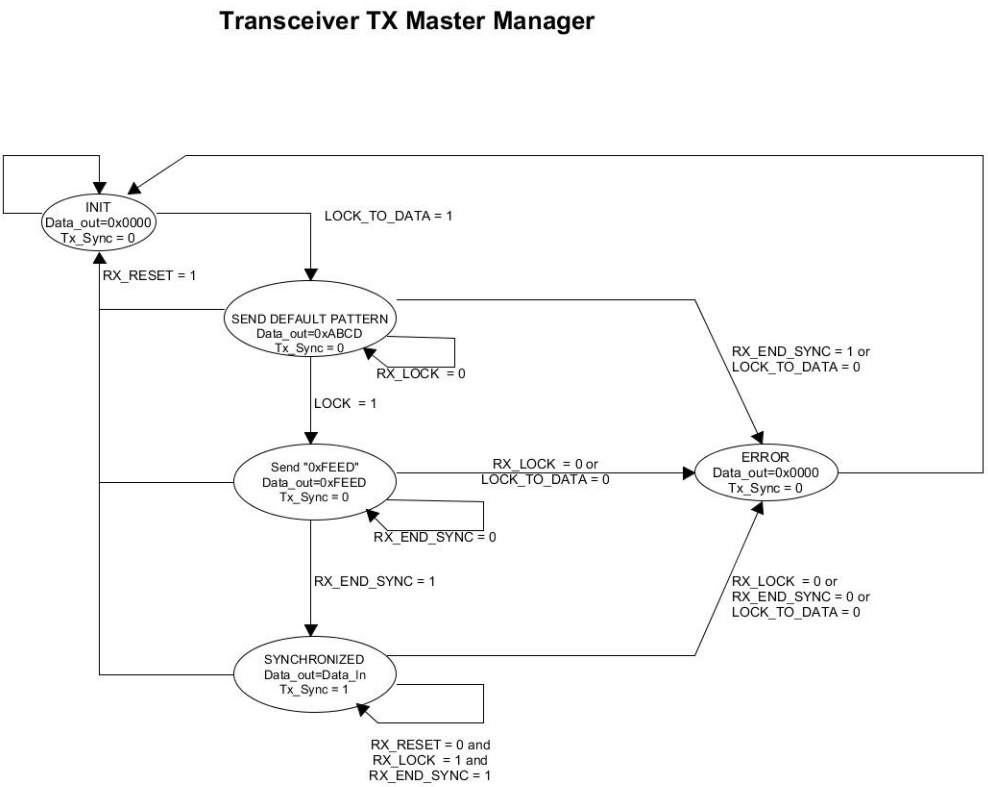


Figura 48 macchina a stati per il blocco di trasmissione del Master

Il primo stato è quello di INIT che si vedrà essere comune a ciascuno dei blocchi: è uno stato di inizializzazione dove il sistema aspetta che tutti i segnali ed il blocco nativo dei transceiver si portino a regime.

Il sistema rimane in attesa che il segnale di *LockToData* venga attivato, indicando che il ricevitore master sta ricevendo dati (anche random) quindi il ricevitore è abilitato alla ricezione dei dati.

Successivamente si passa nello stato di SEND DEFAULT PATTERN dove il blocco invia un pacchetto dati di default (0xABCD). Questa operazione continua fino a quando non giunge da parte del ricevitore master il segnale di *lock* precedentemente descritto, che segnala la corretta ricezione del pattern di default da parte del blocco di gestione della ricezione. Ciò implica che i blocchi dello slave risultino già sincronizzati.

A questo punto è possibile passare allo stadio successivo. Il blocco invia una particolare sequenza "0xFEED" per informare tutto il sistema che la procedura di sincronizzazione è completata e quindi

è da terminare. Quando la sequenza giunge al ricevitore del master, esso attiva il segnale *end_sync_rx*, ed una volta campionato dal blocco di trasmissione master tutto il sistema risulterà sincronizzato ed i dati che saranno inviati all'utilizzatore verranno campionati dalla porta d'ingresso *data_in[0..15]*. Da ciascuno degli stati descritti, in caso di funzionamento anomalo o di segnali non coerenti è stato previsto il passaggio da uno stato di ERROR. Per come è stato concepito, in concomitanza di questo stato viene segnalato sia dove è avvenuto l'errore, sia che genere di errore sia. Da questo stato il sistema viene poi resettato e re inizializzato, ricominciando la procedura di sincronizzazione dal principio.

stato	**Codice errore stato**	**Tipologia di errore possibile**
INIT	00	Nessun errore possibile.
SEND_DEFAULT_PATTERN	01	.perdita*LockToData* *.RxEndsync=1*
SEND_ENDSYNC_PATTERN	10	.perdita*LockToData* .perdita *Lock* dei dati
SYNCHRONIZED	11	.perdita*LockToData* .perdita*Lock*dei dati .perdita*Ensync*

Figura 49 tabella degli errori possibili relativi a ciascuno stato

3.3.1.2 GESTORE RICEVITORE SLAVE

Il blocco del gestore di ricezione relativo allo slave si occupa della gestione dei dati ricevuti dallo slave. Viene infatti inserito dopo il blocco nativo di ricezione dello slave e quindi vede in ingresso i dati in parallelo ricevuti dal master.

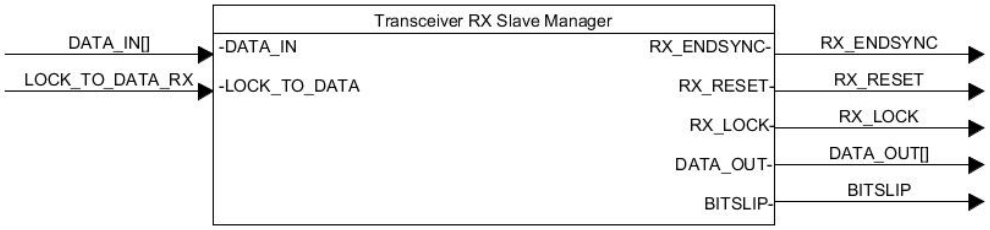


Figura 50Blocco Transceiver RX Slave Manager

Come si può osservare, i segnali di input e output che vengono gestiti dal blocco sono gli stessi già descritti per il gestore trasmettitore master, ad eccezione del bitslip necessario per l'allineamento dei dati ricevuti.

Il blocco consta di diversi stati. Primo tra tutti lo stato di inizializzazione nella quale vengono settati tutti i parametri a una condizione di default.

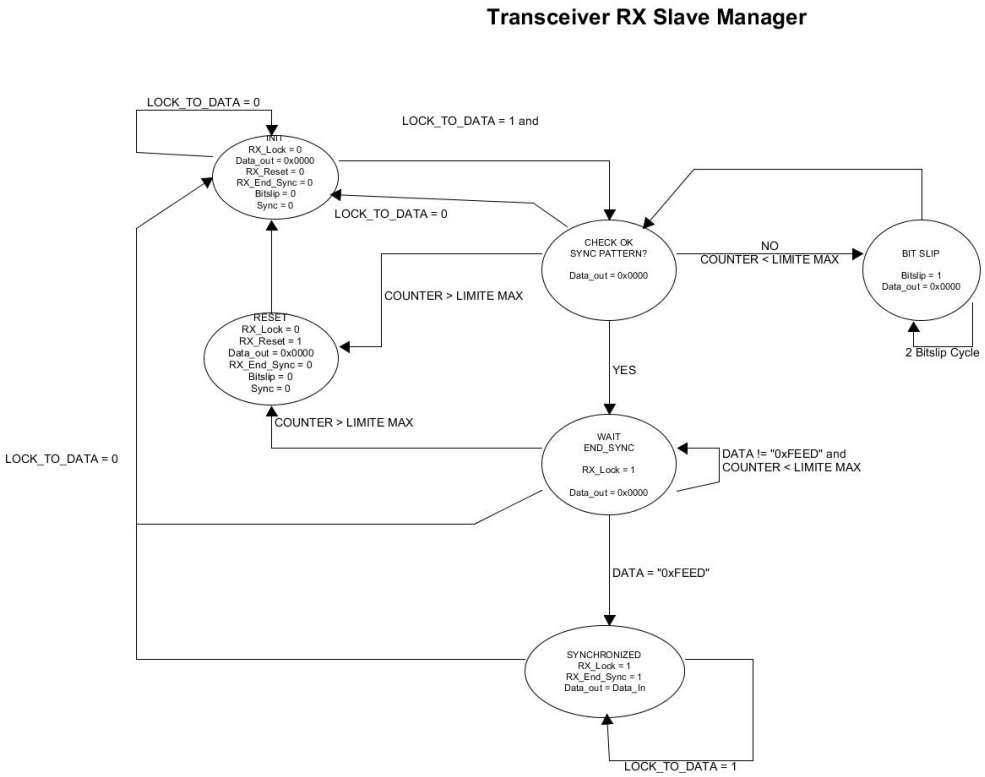


Figura 51macchina a stati per il blocco di ricezione dello Slave

Anche in questo caso, affinché la procedura di gestione dati sia inizializzata è necessario il segnale di lock dei dati. Dal momento che questo segnale viene attivato, il blocco analizza i dati che giungono dal blocco nativo di ricezione confrontandoli con il pattern di default (0xABCD), che deve coincidere con quello inviato dal gestore del master in trasmissione. Qui viene inserito il meccanismo di bitslip che nei progetti precedenti veniva implementato dal checker. È di fondamentale importanza notare che questa procedura non continua per un tempo indeterminato, ma ha un limite massimo di implementazioni successive dopo le quali se la corrispondenza non viene trovata, il sistema si resetta tornando a una condizione iniziale. Questo meccanismo è significativo perché nel caso di errori in trasmissione, o ad esempio se il cavo viene disconnesso, il sistema riesce a individuare l'errore e tentare di ripristinare il corretto funzionamento, non bloccandosi così in un loop infinito del quale l'utente non riesce ad essere informato. Se invece si raggiunge lo stato di sincronizzazione con il pattern di

default, si passa allo stato successivo nel quale si attende la sequenza di terminata sincronizzazione, inviata dal master tx una volta che anche il master rx è sincronizzato. Anche in questo caso è stato previsto un tempo massimo di attesa, terminato il quale il sistema viene resettato per segnalare un malfunzionamento. Infine, così come per il gestore trasmettitore master, se giunge conferma dell'avvenuta sincronizzazione il sistema passa allo stato di sincronizzazione. Per ciascuno stato inoltre, è stato previsto che nel caso alcuni segnali presentassero un comportamento anomalo, si ritornasse allo stato iniziale di inizializzazione.

3.3.1.3 GESTORE TRASMETTITORE SLAVE

Il blocco gestore trasmettitore slave viene inserito all'interno del loop che si sta descrivendo tra il blocco gestore ricevitore slave e il blocco nativo di trasmissione dello slave. Questo blocco viene utilizzato principalmente per inviare feedback al master sul corretto funzionamento della trasmissione dei dati.

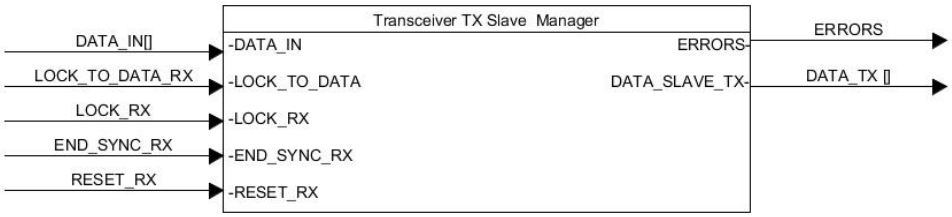


Figura 52 Blocco Transceiver TX Slave Manager

Oltre allo stato di inizializzazione comune a tutte le macchine a stati descritte, questo blocco è composto da un solo macrostato che gestisce interamente la trasmissione dati, più quello di segnalazione errore.

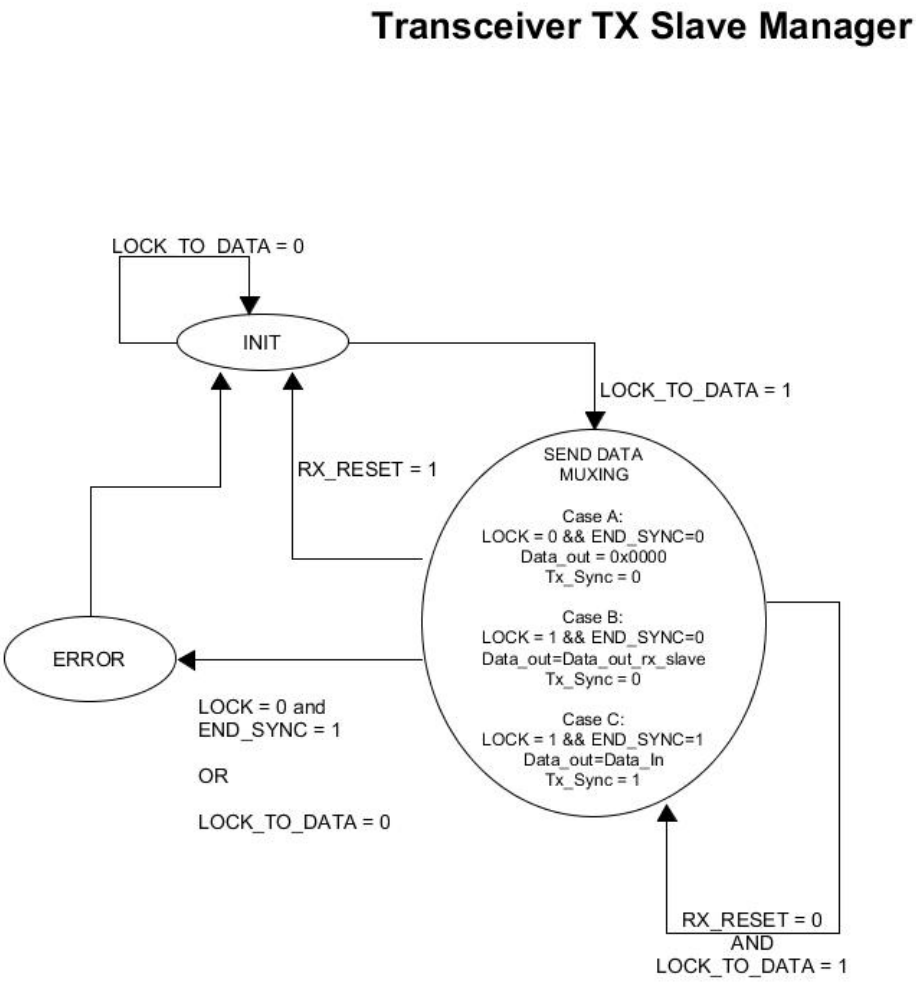


Figura 53macchina a stati per il blocco di trasmissione dello Slave

Questo macrostato invia una sequenza differente di dati in base ai segnali che risultano attivi in ingresso al blocco. Di conseguenza si avrà in uscita nel caso di lock non raggiunto e di non avvenuta sincronizzazione una sequenza di zeri, nel caso invece di presenza di lock i dati in loopback inviati dal blocco di ricezione dello slave. Infine, nel caso della presenza di entrambi i segnali attivi, sarà possibile inviare i dati scelti dall'utilizzatore.
Se si presentano anomalie nella configurazione di questi segnali, si passa a uno stato di errore con il riavvio della procedura di inizializzazione della macchina a stati. Generalmente i dati slave inseriti in ingresso ad avvenuta sincronizzazione vengono utilizzati per inviare comandi al blocco master. Sulle modalità con cui questo avvenga, se ne discuterà nella sezione apposita.

Di seguito si riporta la tabella degli errori possibili relativi all'unico stato esistente, quello di *SEND_DATA_MUXING*.

Stato	Tipologie di errori possibili
SEND_DATA_MUXING	. *Lock*=0 &*Ensync*=1 . *LockToData*=0

Figura 54 tabella errori possibili nel gestore trasmettitore slave

3.3.1.4 GESTORE RICEVITORE MASTER

L'ultimo blocco da affrontare in questa trattazione, è quello di gestore ricevitore del master. Esso riceve i dati in uscita dal blocco nativo di ricezione del master e rappresenta la chiusura del loop di sincronizzazione tra i 4 blocchi di trasmissione/ricezione dei dati.

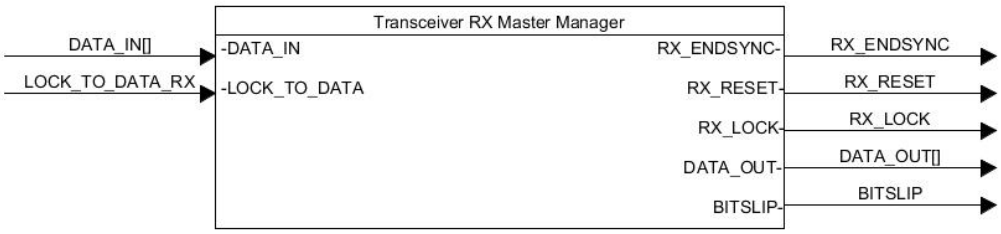


Figura 55 Blocco Transceiver RX Master Manager

Come per le altre macchine a stati, il sistema parte da una fase di inizializzazione in cui i segnali vengono posti a un valore di default.

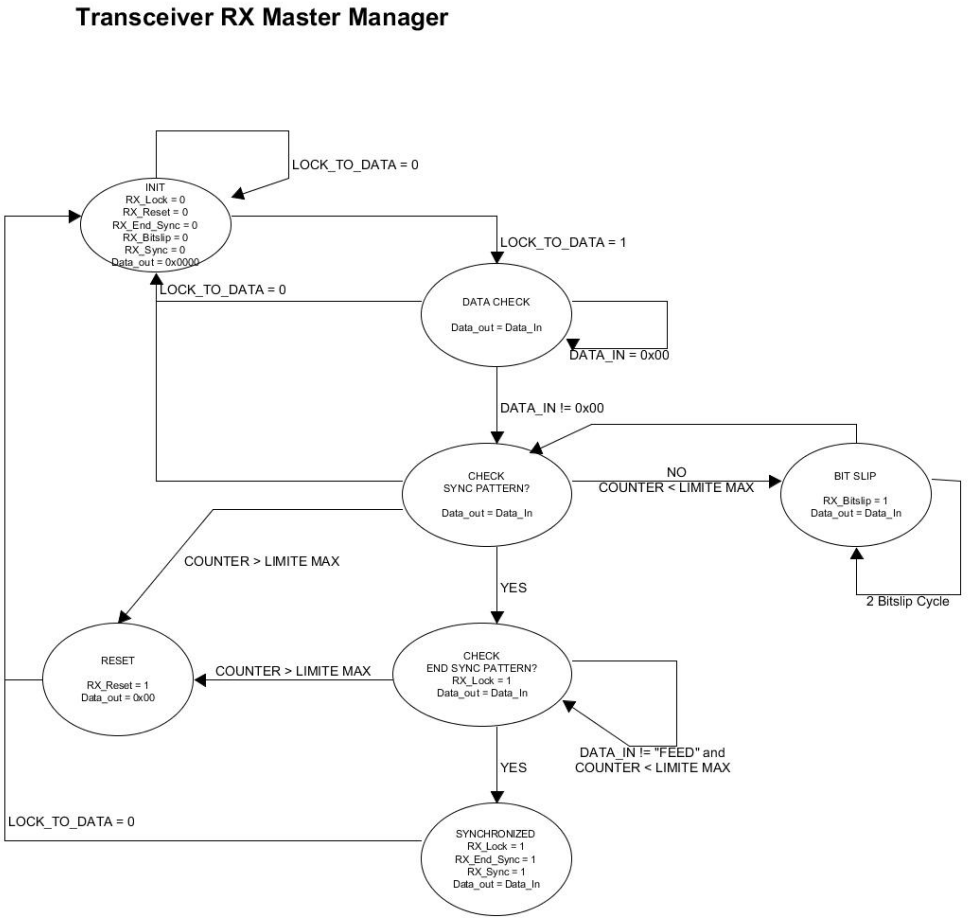


Figura 56macchina a stati per il blocco di ricezionedel Master

Una volta raggiunto il lock dei dati, si passa a una fase di check durante la quale il sistema permane fino a quando la sequenza di dati ricevuti è una sequenza di zeri. Successivamente se i dati inviati variano, la macchina a stati passa a uno stato di verifica della sincronizzazione con il blocco di trasmissione dello slave per mezzo del bitslip, allo stesso modo del blocco di ricezione del master già precedentemente descritto. Anche in questo caso, se il sistema permane troppo nello stato, è sintomo di un malfunzionamento che può essere sia un'anomalia nella trasmissione, sia un caso di cavo disconnesso. Superato il numero massimo di iterazioni consentite, il sistema passa a una fase di reset per poi tornare all'inizializzazione della macchina. Se invece viene raggiunta la sincronizzazione con i dati inviati dal blocco di trasmissione dello slave,si passa allo stadio successivo, ovvero la sincronizzazione con il blocco di trasmissione del master. Questa avviene per mezzo del riconoscimento della sequenza di dati di fine trasmissione inviata dal blocco di trasmissione del master. Se dopo un determinato numero di iterazioni questa sequenza non viene

riconosciuta nei dati che vengono ricevuti, anche in questo caso si passa alla condizione di reset. In caso contrario, anche l'ultimo blocco giunge allo stato di sincronizzazione. In questo modo il sistema dovrebbe risultare totalmente sincronizzato e pronto per trasmettere i dati dell'utente.

3.3.2 ANALISI PRESTAZIONI E ROBUSTEZZA

L'integrazione di questo sistema di sincronismo nel progetto, porta con sé sia vantaggi sia svantaggi. Per quanto riguarda i vantaggi, al termine del procedimento l'intero sistema risulta sincronizzato, con la conseguente validazione dei dati inviati. Inoltre, essendo stati previsti nei blocchi sopra descritti sistemi di segnalazione errore e ripristino della sincronizzazione, anche considerando un eventuale malfunzionamento il sistema riesce a far fronte ad esso in breve tempo, o a segnalarlo immediatamente, arginando così la perdita di dati. Il sistema risulta quindi più robusto di quello precedente e più affidabile per quanto riguarda la correttezza dei dati inviati. Come svantaggio, l'integrazione di questo sistema porta con sé il ritardo inevitabile dovuto al tempo di sincronizzazione dei vari blocchi ogni qualvolta venga perso il lock. Questo aspetto dovrà essere gestito nei progetti futuri. Nel complesso però, lavorando ad altissime data rate, questo tipo di inconveniente non penalizza eccessivamente il sistema sviluppato.

L'unico aspetto che è necessario considerare anche in questo progetto, è che essendo i blocchi sincronizzati a due a due, non è possibile per come sono state gestite le macchine a stati accorgersi di un eventuale errore di trasmissione dopo l'avvenuta sincronizzazione. Per far fronte a questo problema, è stato inserito un protocollo per la trasmissione di dati e comandi.

3.4 PROGETTAZIONE DI UN PROTOCOLLO CUSTOM PER LA TRASMISSIONE DI DATI E COMANDI

Nei capitoli precedenti è stata affrontata la progettazione del sistema fino al raggiungimento della sincronizzazione di ciascun blocco con gli altri. Un aspetto però che non è stato ancora valutato è la verifica del corretto funzionamento della trasmissione dati

anche dopo l'avvenuta sincronizzazione. Questa verifica viene inserita mediante la creazione di un protocollo custom sviluppato appositamente per le esigenze del progetto, che si basa sulla trasmissione di pacchetti dati e pacchetti comandi.
Per effettuare ciò, sono stati assegnati due specifici frame ai dati e ai comandi che si vogliono inviare.

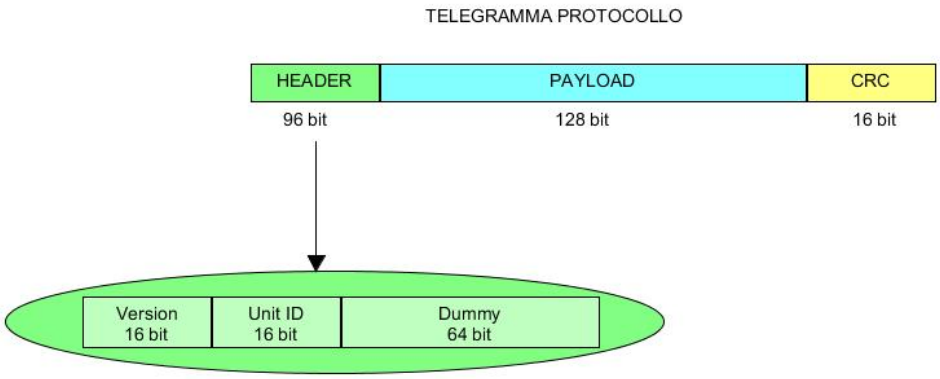


Figura 57 telegramma utilizzato per la trasmissione di comandi

In figura è stata riportata la struttura interna del telegramma utilizzato per la trasmissione dei comandi. Come si può osservare, essa si compone principalmente da un *header*, che contiene una sequenza di bit di riconoscimento comando (è stata predisposta una zona *dummy* per permettere eventuali sviluppi futuri senza dover modificare la lunghezza del frame), una parte centrale che costituisce il cuore del comando inviato, detto *payload*, e il CRC calcolato allo scopo di garantire la robustezza della trasmissione.
Nel caso dei dati la struttura è simile. Non è però necessario il CRC, e solitamente la lunghezza del payload può variare.

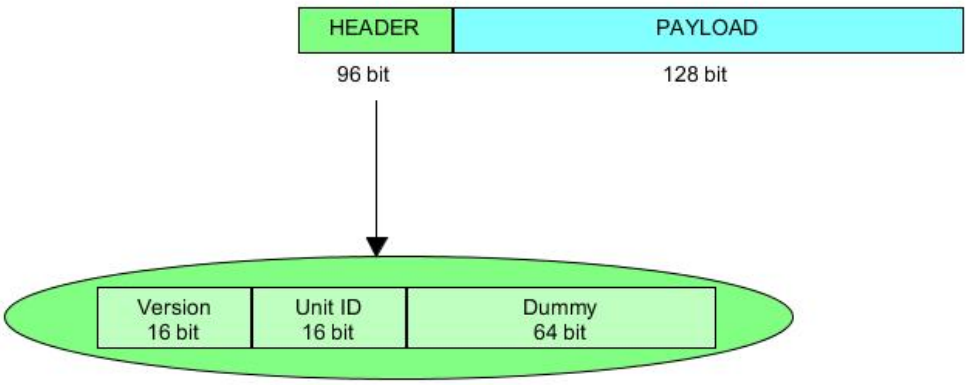


Figura 58 telegramma utilizzato per la trasmissione dei dati

Considerando la struttura del sistema preso in considerazione, è necessario implementare il flusso di dati col protocollo in due modalità differenti per dati e comandi: monodirezionale nel caso dei dati, bidirezionale per i comandi. La trasmissione di dati avviene infatti dal master allo slave, e non è necessario implementarla, dopo l'avvenuta sincronizzazione, anche dallo slave al master. I comandi invece è necessario che vengano trasmessi in entrambe le direzioni dal momento che regolano le azioni dell'intero sistema. Di conseguenza è necessario impostare un blocco di controllo bidirezionale. Sono stati dunque ideati quattro blocchi principali: un trasmettitore di dati e comandi per la trasmissione dal master allo slave e i corrispondenti ricevitore dati e ricevitore comandi in ricezione nello slave; un trasmettitore solo per i comandi e il relativo ricevitore per la trasmissione dallo slave al master.
Per il ciclo di trasmissione da slave a master infatti era sufficiente testare l'invio corretto dei comandi. Per testare il sistema simulandone il reale funzionamento è stato affiancato ai blocchi anche un simulatore di dati e comandi che svolgeva il compito di creare le sequenze da inviare ai blocchi di gestione.

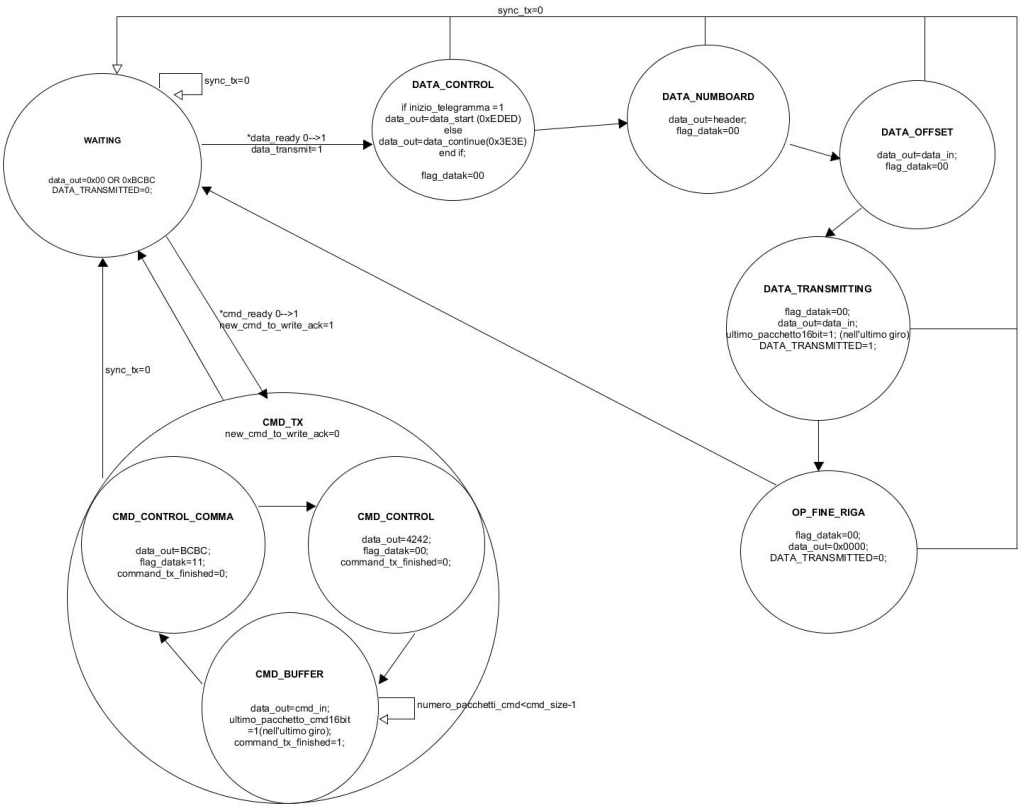


Figura 59 blocco di trasmissione dati e comandi

Il primo blocco progettato è quello di trasmissione dati e comandi. Una volta che il sistema si è sincronizzato, questo blocco ha la funzione di inviare le sequenze complete di dati e comandi,che vengono trasmessi dal simulatore, costruendo il telegramma corrispondente mostrato in figura precedentemente. Fino a quando il blocco non riceve il ready o di dati o di comandi, esso permane nello stato di init alternando uno stato di idle e uno stato di comma (0xBCBC). Ad queste due alternative viene associato un flag (*flag_datak*), che può assumere il valore 0 in corrispondenza dei dati, o 1 in corrispondenza dei comandi. In questo caso, il flag a 0 viene associato allo stato di idle e il flag 1 allo stato di comma. Dal momento che sia il telegramma dati sia quello comandi vengono preceduti dal comma, il flag viene lasciato ad 1 per un ciclo successivo se si riceve un comando da inviare, o viene settato a 0 nel caso dei dati. Il resto del comando ad eccezione del primo ciclo viene considerato al pari di un dato dal flag (quindi settato a 0). Due segnali, *data_ready* e *cmd_ready*, discriminano se il segnale ricevuto da questo blocco sia un dato o un comando, attivando di conseguenza la sequenza di azioni corrispondente. Le sequenze così costruite vengono poi inviate in input alla macchina a stati e rientrano nel sistema precedentemente illustrato.

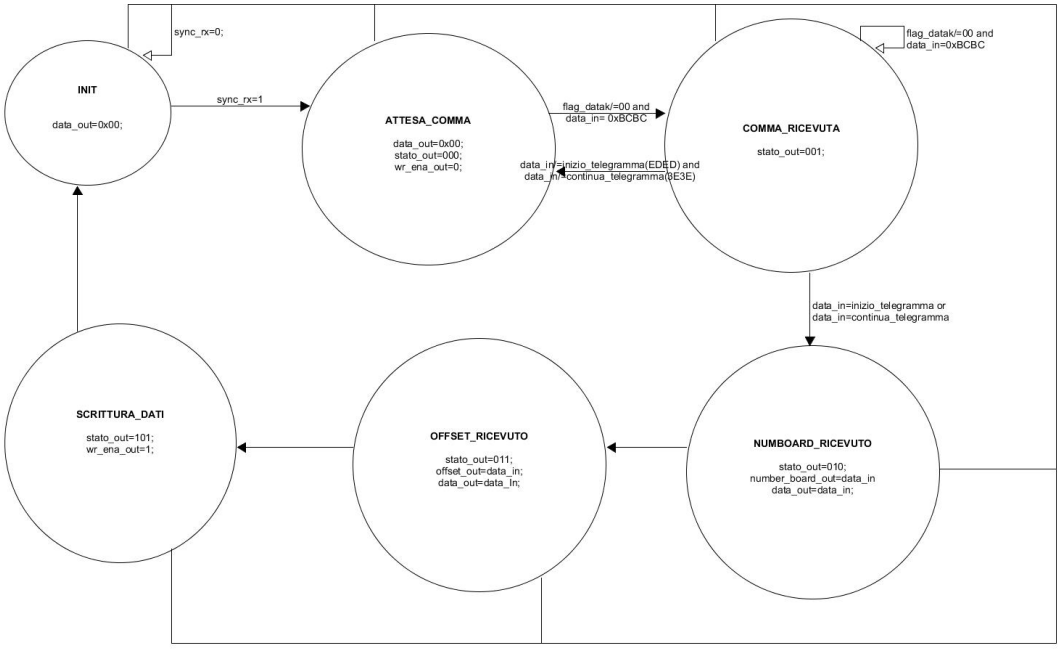


Figura 60 ricezione dati

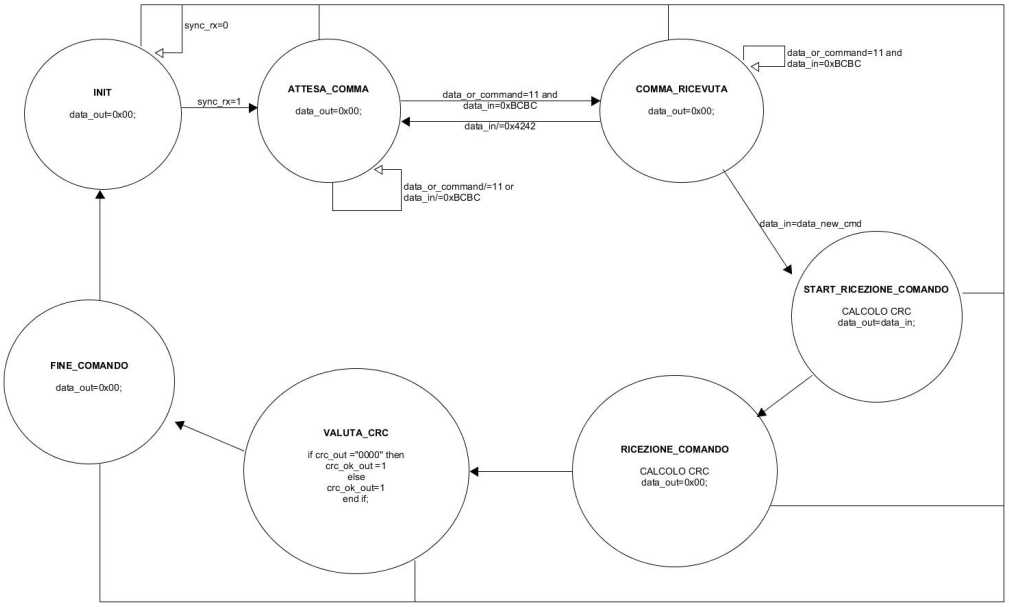


Figura 61 ricezione comandi

I due blocchi di ricezione hanno una struttura simile. Essi attivano la ricezione di dati e comandi quando vengono raggiunti dalla comma(0xBCBC), una sequenza specifica valida per entrambe le combinazioni. In base alla parola successiva di 16 bit, si attiva poi o l'uno o l'altro blocco in quanto essa è specifica della sequenza trasmessa, che sia di dati o comandi. Riconosciuto dunque l'header della parola, il blocco corrispondente trasmette in uscita il dato o il comando ricevuto. È da notare che per i dati in questo sistema l'header contiene il numero della board considerata e un offset, di conseguenza sono stati previsti gli stati corrispondenti nella macchina a stati. Nel caso dei comandi invece, prestabilito il CRC che si vuole utilizzare, il blocco di ricezione verifica anche la corrispondenza tra le due sequenze per confermare o meno la correttezza del comando ricevuto.

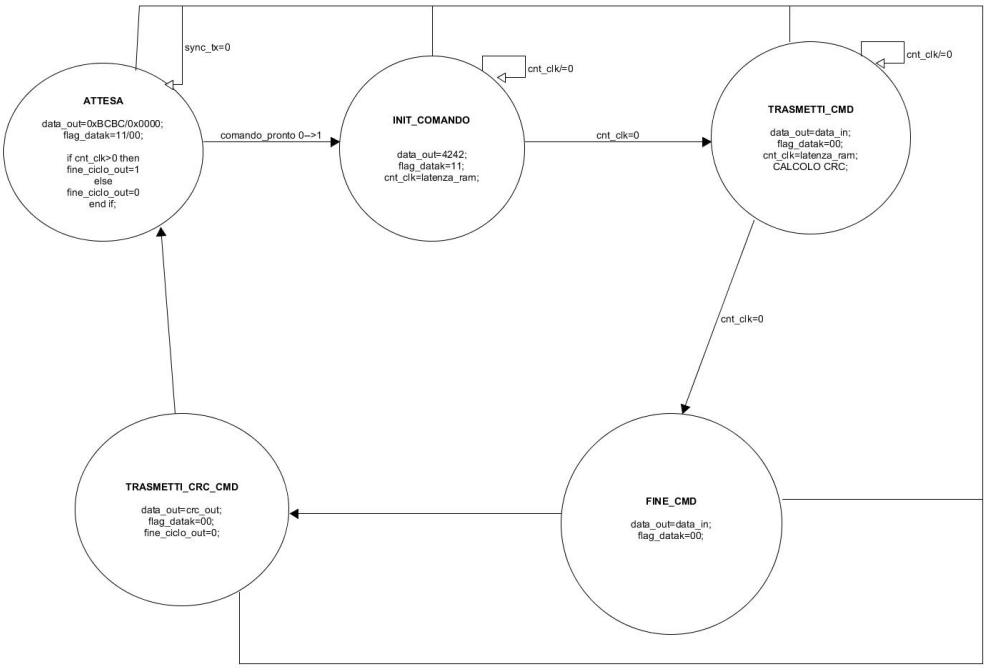


Figura 62 trasmettitore comandi

Infine, per quanto riguarda il trasmettitore di soli comandi, esso ha la stessa funzione del primo trasmettitore seppur con una struttura più semplice dovuta al fatto che non è necessario discriminare in partenza tra dati e comandi.

Sono stati testati questi blocchi visualizzando i segnali con il signaltap analyzer, ottenendo i risultati seguenti nella trasmissione.

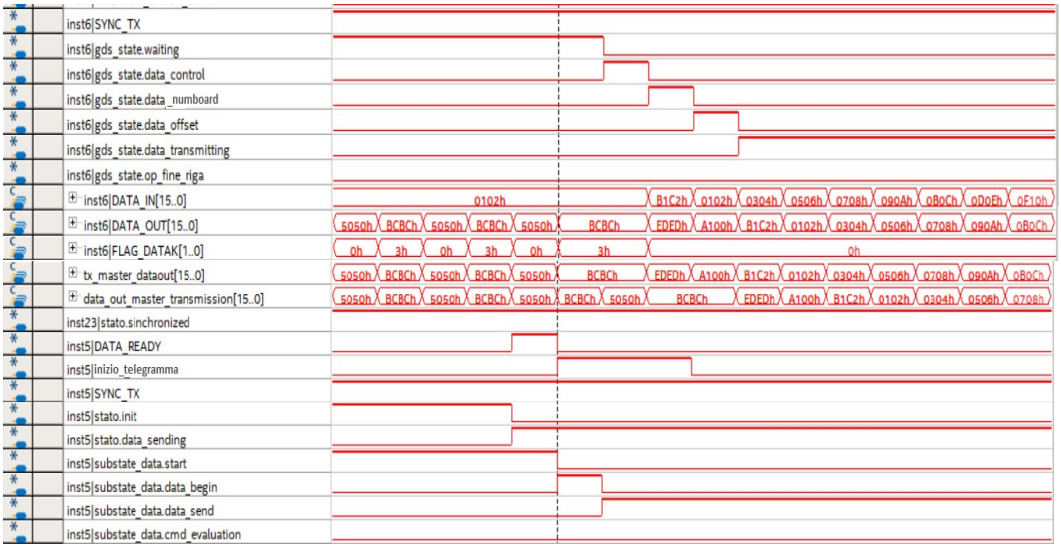


Figura 63 sequenze iniziali della trasmissione dati. Si può osservare la sequenza della comma (BCBC) e della parola che identifica il dato (EDED) prima della trasmissione vera e propria dei dati.

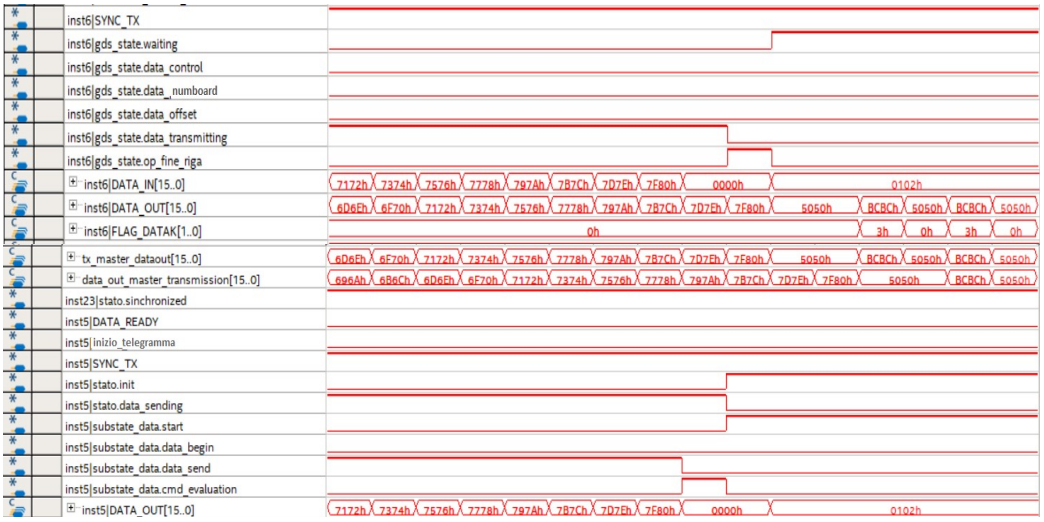


Figura 64 sequenze di fine trasmissione dati. Si può notare il ritorno a uno stato di idle (5050) dopo il termine del dato

Figura 65 sequenza di trasmissione del comando. esso termina col CRC scelto, in questo caso C185.

Nel caso della ricezione di dati e comandi, sono stati acquisiti i segnali allo stesso modo.

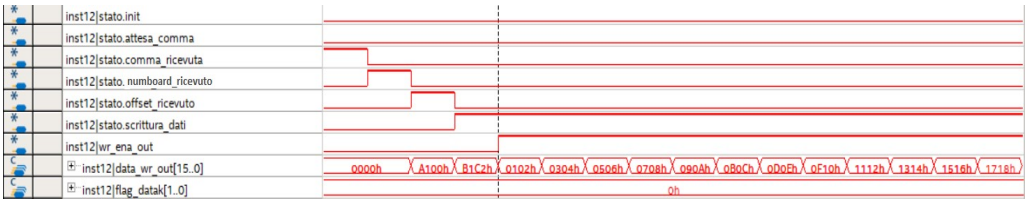


Figura 66 è possibile notare il riconoscimento da parte del blocco del dato, e la conseguente trasmissione in uscita dello stesso (data_wr_out) svincolato dalle parole di header informative iniziali (numboard + offset)

Figura 67 sequenza di inizio e fine ricezione comando. in questo caso viene valutata anche la corrispondenza del CRC(valuta crc).

Tutti i segnali analizzati sono coerenti con il funzionamento desiderato. Il sistema ora è dunque in grado anche di discriminare sequenze di dati e comandi inviate e di essere testato durante il funzionamento anche dopo l'avvenuta sincronizzazione.

4. ANALISI E STUDIO DEL SISTEMA SU PIATTAFORMA XILINX

Per questo progetto è stata valutata in alternativa alla piattaforma Intel, anche la piattaforma Xilinx. È stato dunque implementato un sistema di comunicazione sul tool Vivado, allo scopo di testarne le funzionalità e le prestazioni implementando i primi step già svolti per il progetto basato sull'Arria 10.

4.1 DEMO BOARD DI SVILUPPO: ARTIX AC701 E KINTEX K705

Si è deciso di selezionare la famiglia di fascia media prodotta dalla Xilinx per effettuare i test. Per implementare il progetto dunque è stato scelto il Development Kit della Artix 701, della famiglia delle 7 series.

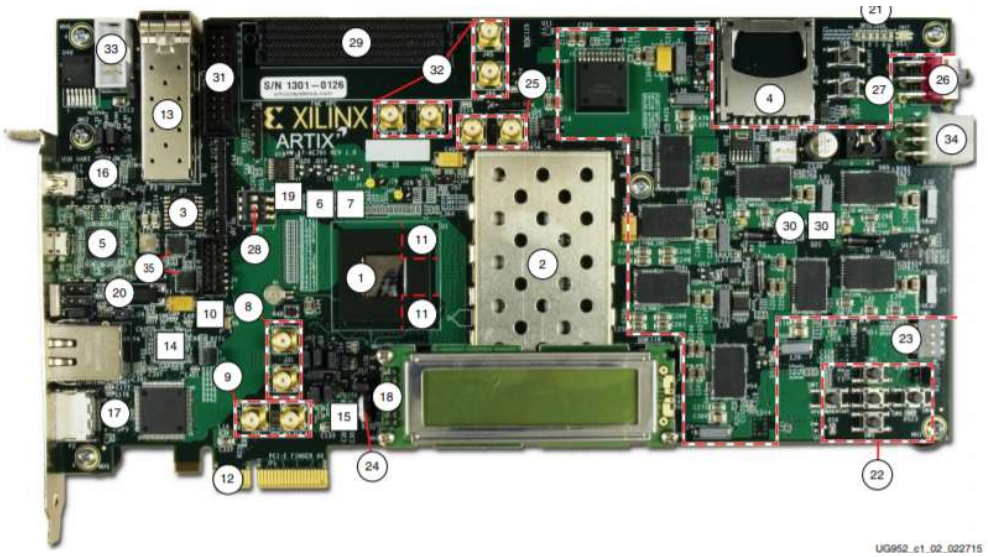


Figura 68Artix 701

Questo Development kit mette a disposizione svariate features. A livello di connessioni, sono resi disponibili:

- un canale SFP+ (nr.13 in fig.)
- un connettore PCIe (nr.12in fig.)
- i canali da tx/rx per i cavi coassiali (nr. 32in fig.)

Come nel caso del Dev. Kit Intel, anche qui il kit dà la possibilità di connessione tramite JTAG ed una serie di pin user che possono essere utilizzati durante la progettazione, ad esempio i led.
Il connettore SFP+ può quindi supportare fino a 28 Gbps in termini di data rate. I transceiver nelle 7 series sono raggruppati a gruppi di 4 canali, chiamati *quad*,ciascuno dei quali ha un clock dedicato. Il canale dedicato alla trasmissione tramite SFP+ si trova nel *Quad213* ed è quello che verrà utilizzato in questo progetto.

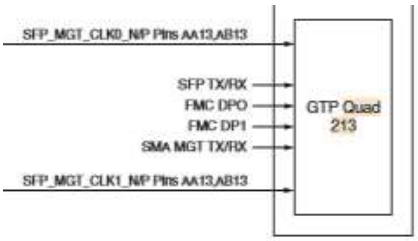


Figura 69 Quad 213

Osservando il datasheet della scheda, si può inoltre verificare che Il data rate massimo a cui i transceiver possono arrivare è 6.6 Gbps. Tale valore è sufficiente per rispettare le minime specifiche definite dal progetto anche se inferiore al data rate raggiunto con l'Arria 10. Un altro aspetto da considerare riguardante l'impostazione del sistema Xilinx, è la disponibilità nel Development kit di un solo canale SFP+. Per testare la trasmissione dati tramite su quel genere di canale, utilizzando il cavo in fibra, occorre dunque utilizzare un'ulteriore scheda di supporto in quanto non è disponibile un ulteriore connettore SFP+ o QSFP+, pertanto sarà necessaria un'altra board di supporto per sviluppare i test.
In questo caso, è stata resa disponibile una Kintex 705. La Kintex 705 appartiene alla stessa famiglia delle 7 series, ma raggiunge prestazioni maggiori in termini di data rate rispetto all'Artix 701, raggiungendo i 12.5 Gbps. Tuttavia, essa appartiene a una fascia più alta dell'Artix, ed il suo utilizzo diretto nel sistema non è stato preso in considerazione in quanto non conveniente in termini di rapporto costi-prestazioni richieste.

	Type	Max Performance[1]	Max Transceivers	Peak Bandwidth[2]
Kintex-7	GTX	12.5	32	800 Gb/s
Artix-7	GTP	6.6	16	211 Gb/s

Figura 70 specifiche delle due schede utilizzate in termini di data rate.

Per implementare il progetto dunque, sono state utilizzate le due schede, un cavo in fibra ottica SFP+/SFP+, i cavi coassiali per le

prove di trasmissione iniziali, e naturalmente il cavo di programmazione.

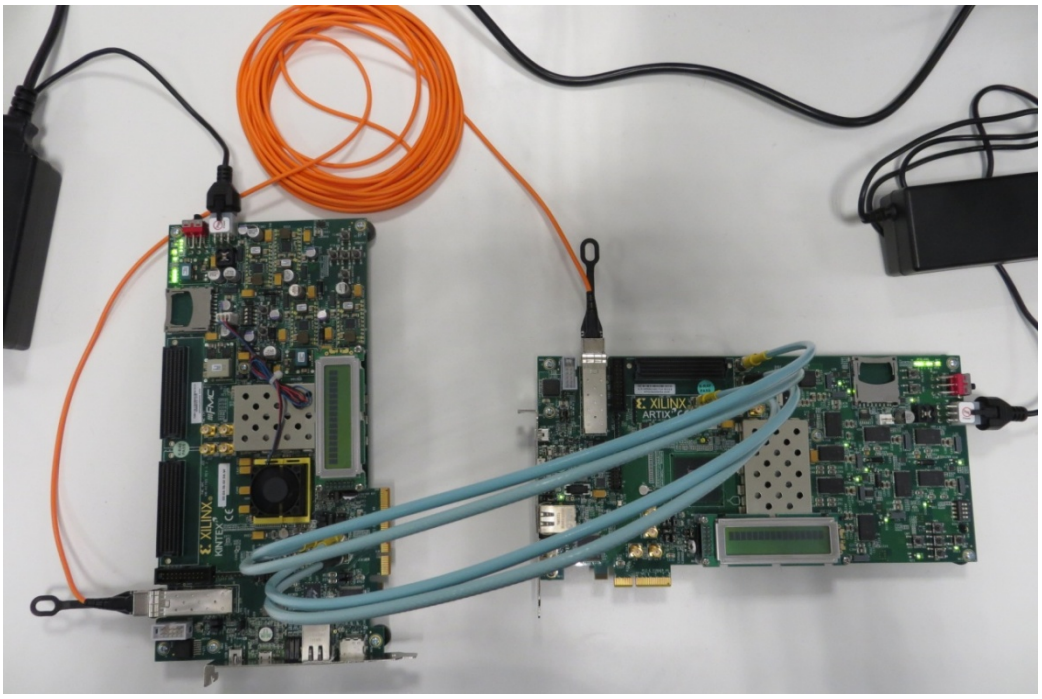


Figura 71 Set up di progetto con piattaforma Xilinx

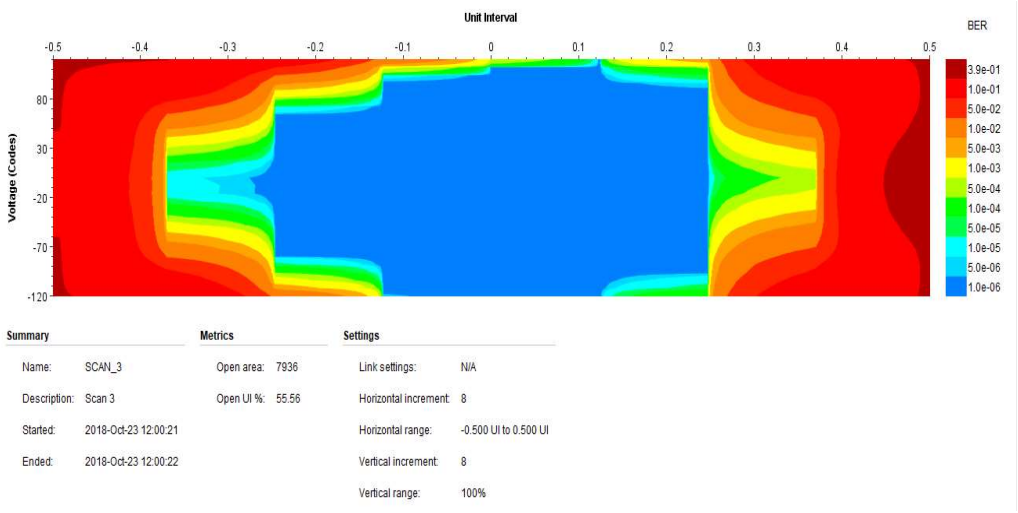
4.2 ANALISI DI QUALITA' DEL CANALE TRAMITE IBERT

Per testare la qualità del canale di trasmissione, Xilinx rende disponibile un tool per la riconfigurazione dinamica dei transceiver denominato IBert. Questo tool agisce su una serie di parametri relativi al canale, come ad esempio l'equalizzazione,per raggiungere la configurazione che più ottimizza le prestazioni del dispositivo. Per fare questo, IBert trasmette una sequenza di test pseudorandom (PRBS test) che deve essere ricevuta uguale all'altro capo del canale di trasmissione. Queste sequenze si basano sugli stessi principi di quelle implementate dal Transceiver Toolkit sulla piattaforma Intel, e permettono di calcolare la BER del canale ad ogni implementazione. IBert in aggiunta presenta ulteriori features che possono essere utilizzate per le analisi di qualità. È possibile infatti scegliere la tipologia di test in loopback che si desidera implementare tra near/far end PCS e near/far end PMA in base alle sezioni del canale che si coinvolgono nella trasmissione. Di questa possibilità si discuterà più dettagliatamente in seguito.

Inoltre, è possibile implementare il diagramma ad occhio. Il diagramma ad occhio è un metodo che permette di valutare la qualità di un segnale digitale attraverso una particolare visualizzazione del segnale ricevuto. Per poter visualizzare il diagramma, occorre considerare il segnale trasmesso attraverso il canale scelto. Simultaneamente viene inviato il clock di ricezione al trigger esterno per sincronizzare l'oscilloscopio sul quale si basa il tool con il flusso di bit in arrivo. Il risultato di queste acquisizioni è il diagramma ad occhio, ovvero una rappresentazione visiva ottenuta dalla sovrapposizione delle forme d'onda associate ai simboli che arrivano al ricevitore. Tramite il diagramma,ne consegue che :

- Un occhio "aperto" è indice di buona qualità del segnale, e quindi di un rapporto segnale/rumore (SNR) elevato;
- Un occhio "chiuso" indica un'insufficiente qualità del segnale ricevuto e quindi un rapporto segnale/rumore (SNR) basso.

Dai test effettuati con IBert tra l'Artix e la Kintex di supporto, risulta un'apertura dell'occhio soddisfacente.



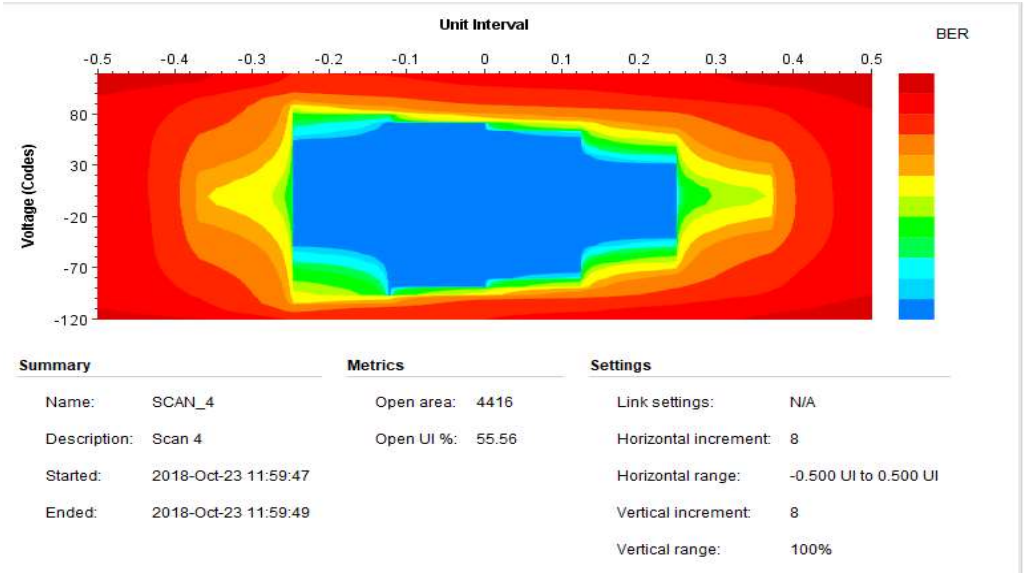


Figura 72 diagramma ad occhio senza loopback con la Kintex(sopra) e con il loopback passante dalla Kintex (sotto). si può osservare la buona apertura dell'occhio e il miglioramento della stessa nel secondo caso

Tramite questi test si è dunque quantificata e verificata la bontà della trasmissione nel canale. Una volta effettuate queste prove,è stato dunque possibile procedere con l'impostazione di un vero e proprio canale di comunicazione per la trasmissione di dati custom.

4.3 SVILUPPO DI UN CANALE DI COMUNICAZIONE TX/RX

Dopo aver testato il canale con il tool IBert, si sono eseguite diverse prove di trasmissione tramite canale SFP+ concentrandosi sulle prestazioni dell'Artix. Il sistema implementato per queste prove era essenziale. Per i transceiver,Vivado mette a disposizione una serie di blocchi fisici già impostati all'interno dei quali è possibile scegliere parametri quali ad esempio il data rate, o l'ottimizzazione di area o potenza, che possono poi essere inseriti nel progetto.

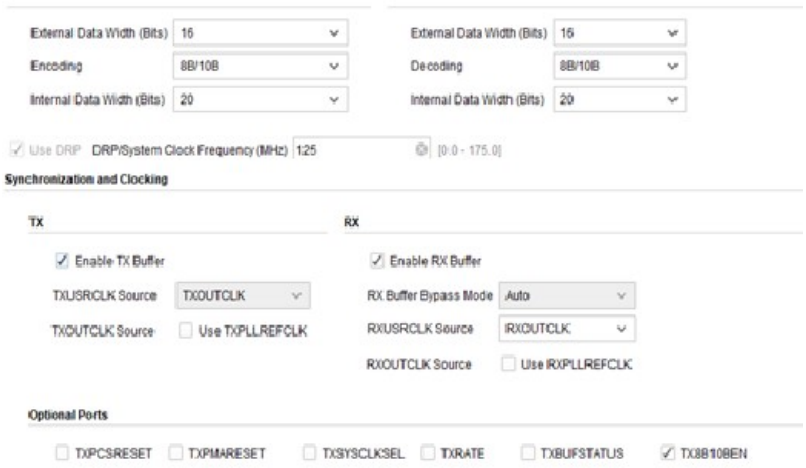


Figura 73 schermate di impostazione dei blocchi nativi per i transceiver

Impostandone uno a un data rate di 5Gbps, a 16 bit d'interfaccia con codifica 8B/10B con gli altri parametri settati alle impostazioni di default, è stato possibile implementare la trasmissione/ricezione attraverso il canale. Di seguito è riportato lo schema del sistema implementato.

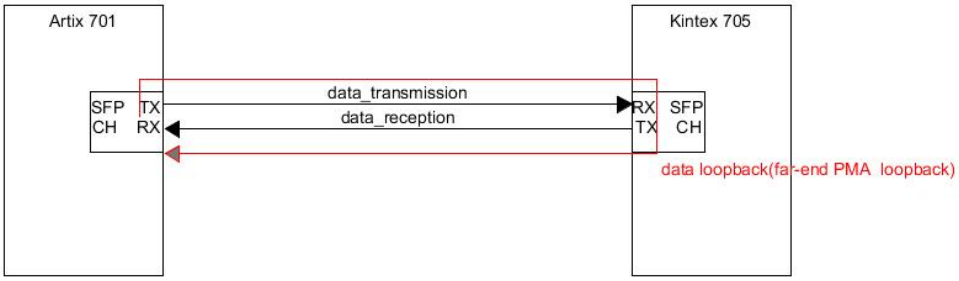


Figura 74 schema del flusso di dati implementato tra le due schede. La decisione di utilizzare il far-end PMA loopback verrà discussa in seguito nella trattazione.

I blocchi principali che componevano il sistema di base erano un blocco di simulazione dati, che creava la sequenza di dati da inviare attraverso il canale, il blocco di transceiver di trasmissione e ricezione che si occupava della trasformazione seriale/parallelo e viceversa per la trasmissione di dati e un checker, che verificava che i dati ricevuti dal transceiver fossero coerenti con quelli inviati.

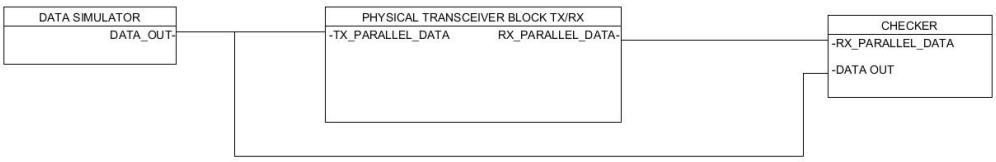


Figura 75schema del progetto implementato

Il blocco fisico del transceiver consta anche di una serie di segnali che fungono da flag o portano informazioni in input o in ouput al blocco. Ad esempio, è possibile segnalare al blocco se la sequenza inviata è un dato (00) o un comando (11). Inoltre, siccome i dati non sempre vengono recepiti dal canale di ricezione già nella configurazione corretta ma è possibile che siano *shiftati*, è disponibile un segnale di *shift* dei bit in ingresso fino a raggiungere la sequenza inviata. A questo proposito, esiste anche un'altra procedura che è possibile implementare in Vivado. Il blocco dei transceiver può essere programmato per riconoscere determinate sequenze di comando (per esempio "0xBC"). Inviando questa parola al blocco insieme al flag di riconoscimento del comando menzionato in precedenza, il blocco si setta con un processo automatico sulla sequenza di ricezione corretta. Da quel momento qualsiasi dato inviato risulta già correttamente ricevuto senza dover aggiungere ulteriori passaggi. Questo processo non è più valido nel caso in cui la trasmissione venga interrotta (sia per causa Hw che per causa Sw).

Impostato il progetto, si può scegliere la tipologia di test di trasmissione che si vuole implementare. In primo luogo, è stato effettuato il test in loopback interno alla scheda. Xilinx mette a disposizione quattro tipologie differenti di loopback:

- Near end PCS
- Near end PMA
- Far end PCS
- Far end PMA

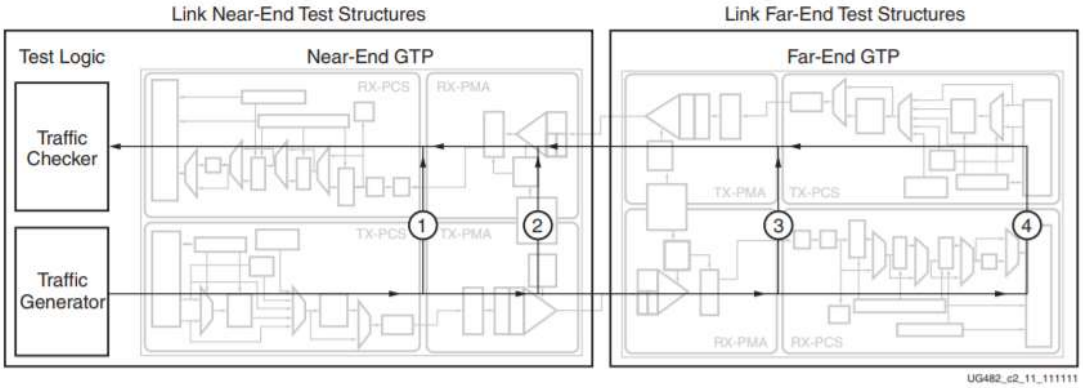


Figura76tipologie di loopback: near end pcs (1), near end pma (2), far end pma(3), far end pcs (4)

Nel caso del near end PCS, i transceiver appartenenti alla stessa scheda vengono posti in comunicazione senza attraversare il canale

fisico. Nel caso invece del near end PMA, esso coinvolge anche la trasformazione da parallelo a seriale e viceversa. Nel caso invece del loopback far end, esso coinvolge un'altra scheda. Infine,nel caso del far end PMA, il loopback viene richiuso tra i transceiver della scheda di supporto senza passare per la logica interna della scheda stessa, mentre nel caso del far end PCS il loopback comprende anche quest'ultima. Nel sistema considerato, sono stati svolti i test di trasmissione nel caso di near end PCS e nel caso di far end PMA utilizzando come scheda di supporto la Kintex. In entrambi i casi si è potuto constatare che la trasmissione avveniva correttamente, quindi sia la scheda sia il canale di trasmissione soddisfavano le esigenze progettuali desiderate.

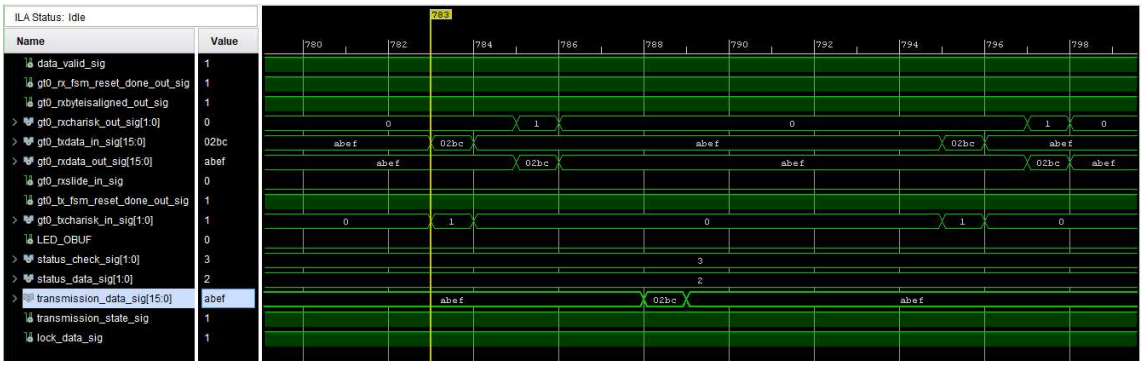


Figura 77 acquisizione dati tramite ILA del funzionamento intervallando dati e comandi

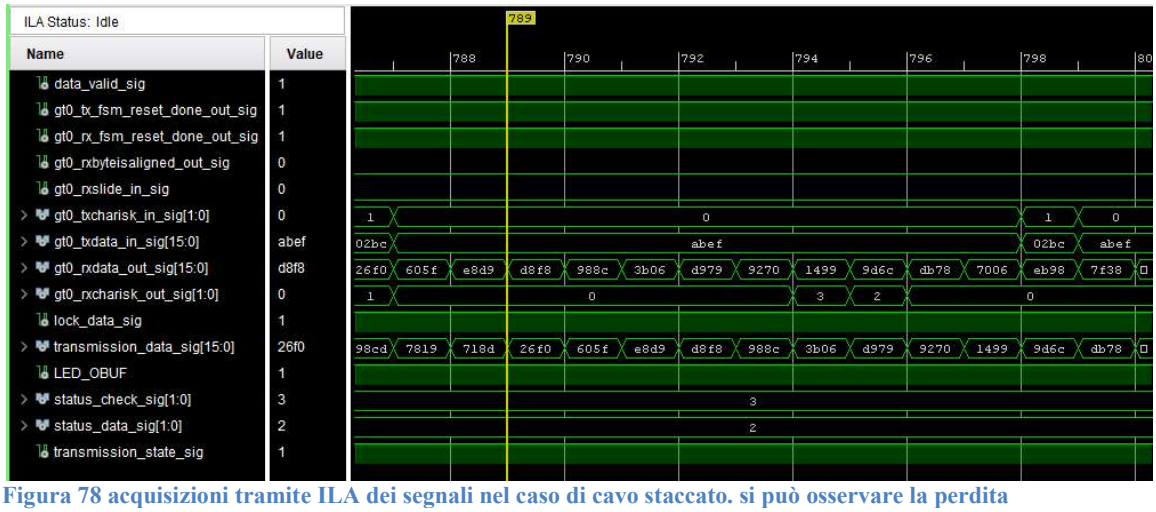


Figura 78 acquisizioni tramite ILA dei segnali nel caso di cavo staccato. si può osservare la perdita del lock dei dati

Una volta verificato che la trasmissione funzioni, i successivi step di progettazione svolti nel caso dell'Arria 10 possono essere implementati allo stesso modo per l'Artix 701.

5. RISULTATI SPERIMENTALI

Una volta terminate tutte le prove sopra descritte, se ne possono trarre diverse conclusioni.

5 .1 VERIFICA E VALIDAZIONE DELLE SPECIFICHE DI PROGETTO

Per quanto riguarda il progetto implementato con piattaforma Intel, si possono riassumere i progetti di prova effettuati e i relativi risultati nella tabella riportata di seguito.

Tipologia di trasmissione	Protocollo	Data rate	Bit di interfaccia	Encoding 8B/10B	*Lock* del canale	Sincronizzazione automatica
Monodirezionale	No	10 GBps	20	No	Non garantito	No
Monodirezionale	No	10 Gbps (8Gbps)	16+4	Si	Garantito	No
Bidirezionale	No	10 Gbps (8Gbps)	16+4	Si	Garantito	No
Bidirezionale	No	10 Gbps (8Gbps)	16+4	Si	Garantito	Si
Bidirezionale	Si, custom	10 Gbps (8Gbps)	16+4	Si	Garantito	Si

Figura 79 tabella riassuntiva progetto con piattaforma Intel

Come si può evincere dai dati,l'ultima configurazione implementata con flusso dati bidirezionale, protocollo di comunicazione, codifica 8B/10B e sistema di sincronizzazione automatizzato risulta la più idonea per il progetto in questione. Inoltre, analizzando i dati ottenuti, è facile verificare che essa soddisfa pienamente le specifiche richieste inizialmente per il sistema. Di seguito si riporta una tabella di confronto tra le prestazioni richieste al sistema, e quelle ottenute dal sistema implementato descritto sopra.

ARRIA 10 GX		
SPECIFICA	**MIN. RICHIESTO**	**RISULTATI**
Data rate	1.6 Gbps	10(8 Gbps)
Signal integrity	Si	Codifica 8B/10B
Latenza	<1μs	<100ns
Tecnologia cavo	-	Fibra
Compatibilità con Arria 10 GX	Max 24 transceiver Banda 1.6 Gbps	12 transceiver Banda 12.5 Gbps

Compatibilità con Cyclone X GX	Max 24 transceiver Banda 1.6 Gbps	12 transceiver Banda 12.5 Gbps
Sincronizzazione	Si. Auto.	Automatica
Gestione flusso dati	-	Protocollo custom

Figura 80 confronto tra le specifiche richieste e quelle ottenute dal sistema sviluppato

Per quanto riguarda la piattaforma Xilinx e la scheda Artix 701, anch'essa rispetta le specifiche richieste dal progetto, nonostante il suo limite massimo per il data rate a 6.6 Gbps. Anche in questo caso, avendo introdotto la codifica 8B/10B, è necessario ricordare l'overhead del 20% che essa comporta, portando la banda utile a 5.3 Gbps.

ARTIX 701		
SPECIFICA	**MIN. RICHIESTO**	**RISULTATI**
Data rate	1.6 Gbps	6.6 Gbps(5.3 Gbps)
Signal integrity	Si	Codifica 8B/10B
Latenza	<1µs	<100ns
Tecnologia cavo	-	Fibra
Compatibilità con Artix 701	Max 24 transceiver Banda 1.6 Gbps	16 transceiver Banda 6.6 Gbps
Sincronizzazione	Si. Auto.	Automatica
Gestione flusso dati	-	-

Figura 81confronto delle prestazioni richieste e quelle ottenute dal sistema sviluppato su piattaforma Xilinx

Occorre però sottolineare che questa piattaforma fornisce diverse features non presenti nel caso dell'Intel quali ad esempio il diagramma ad occhio e il lock automatico dei canali. Si presenta dunque come una valida alternativa da tenere in considerazione per lo sviluppo di questo sistema.

CONCLUSIONI E SVILUPPI FUTURI

Il sistema così ideato risulta soddisfare pienamente le specifiche richieste con ampi margini di miglioramenti futuri in termini di data rate. Essendo inoltre non vincolato da alcun protocollo specifico, questo sistema è facilmente riadattabile sia a una versione futura dello stesso progetto, sia per altri progetti che presentano la stessa struttura. È da considerare inoltre la necessità di sviluppare la comunicazione tra i due server di cui è composta l'architettura scelta, e quella tra server e hub. Nel secondo caso, come accennato inizialmente, è possibile utilizzare il sistema sopra descritto introducendo il protocollo PCIe 3.0 x4 compatibile con le prestazioni richieste per il progetto.

RINGRAZIAMENTI

Innanzitutto, il primo enorme “grazie” lo dedico ai miei genitori, che mi hanno supportato (e sopportato) in questo lungo percorso non sempre in discesa. Grazie perché avete gioito con me delle mie soddisfazioni e mi siete stati vicini nelle cadute. Ma soprattutto, perché mi avete spronato a credere in me stessa quando la mia energia da sola sembrava non bastare più. Una sola parola non basterà mai a racchiudere ciò che mi avete dato.

Ai miei fratelli. A Daniele, che nel suo essere silenzioso, ha comunque saputo dimostrarmi il suo supporto in questi anni. Il tuo saper prendere le cose con più leggerezza di me ha più volte smorzato la mia perenne ansia. A Giovanni, piccolo vulcano di energia. Ora puoi tornare a respirare davanti alla porta di camera mia! Grazie perché il tuo incondizionato volermi bene (anche quando sono fuori di testa) mi sprona a tirare fuori il meglio di me. Come sorella maggiore, spero di averti lasciato in questi anni anche qualcosa di buono oltre la fobia per l’università. Vi voglio bene.

Alla mia famiglia, ai nonni, ai cugini e agli zii. I vostri sguardi pieni di orgoglio sono stati la “benzina” per andare avanti quando pensavo di non farcela. Siete una squadra preziosa.

Al mio relatore, il prof Rovatti, che a suo tempo ha saputo appassionarmi alla sua materia, e che poi mi ha accompagnato durante questo ultimo pezzetto di strada verso la fine di una grande avventura.

Ai miei amici, che volente o nolente hanno condiviso con me questi anni di studio. Alcuni da ormai una vita, altri da meno, ma non meno importanti. Siete la ragione per cui posso ancora dire di avere un briciolo di sanità mentale. E per me ormai siete come una famiglia.

A Gildo, che mi ha accolto nel suo reparto come fosse una famiglia, che ha avuto sempre un sorriso per me, che mi ha dato questa splendida opportunità di toccare con mano cosa significa lavorare in R&D in un’azienda come la Sacmi. Al mio vicino di scrivania Marco, a Matteo, Carlo, Martino, Davide, Giuseppe, Luca,Alessandro,Elisa,Max e tutti gli altri ragazzi dell’ufficio che

mi hanno accolto da subito tra loro, facendomi sentire parte della squadra. Siete delle persone speciali, vi porterò sempre nel cuore. Ma soprattutto, un grazie enorme a Marco, tutor per eccellenza. Grazie per avermi seguito con professionalità, gentilezza e un'infinità di pazienza in questi mesi. Mi reputo fortunata ad aver avuto come tutor una così bella persona come te, e non basta un grazie per compensare tutto ciò che mi hai insegnato.

BIBLIOGRAFIA E SITOGRAFIA

- http://www.diee.unica.it/eolab2/ASE-0607/Lucidi-ASE-0607-A.pdf
- http://amslaurea.unibo.it/5361/1/caligari_christian_tesi.pdf
- https://en.wikipedia.org/wiki/Multi-gigabit_transceiver
- http://www.sopto.com/learningcenter_learningcenter_71/the_difference_of_sfp_plus,_sfp_and_xfp.shtml
- https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/arria-10/a10_datasheet.pdf
- https://www.fiberoptics4sale.com/blogs/archive-posts/95047430-active-optical-cable-aoc-explained-in-details
- https://www.finisar.com/active-optical-cables/fcbn510qe2cxx
- https://it.wikipedia.org/wiki/Fibra_ottica
- https://it.wikipedia.org/wiki/Protocollo_di_rete#Elenco_di_protocolli_di_rete_secondo_ISO/OSI
- https://www.wirelessdesignmag.com/product-release/2007/06/serdes-interfaces-fpga-world-what-do-i-need-know-get-started
- https://en.wikipedia.org/wiki/SerDes
- https://www.altera.com/en_US/pdfs/literature/ug/ug_altera_lvds.pdf
- https://www.altera.com/products/intellectual-property/ip.html
- https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-10gbps-ethernet-mac.html
- https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-pci-express-protocol.html
- https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-seriallite2.html
- https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-10gbase-x-xaui-pcs.html
- https://www.altera.com/solutions/partners/partner-profile/tamba-networks/ip/interlaken-from-1gbps-to-1-000gbps.html
- https://it.wikipedia.org/wiki/Spooler_di_stampa
- https://www.utdallas.edu/~akshay.sridharan/index_files/Page5212.htm

- https://www.ge.infn.it/~musico/CourseStuff/VerilogSlides.pdf
- https://en.wikipedia.org/wiki/QSFP
- https://en.wikipedia.org/wiki/Small_form-factor_pluggable_transceiver
- https://it.wikipedia.org/wiki/Fibra_ottica
- http://www.fiberopticshare.com/sfp28-vs-sfp-vs-qsfp28.html
- https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/c10gx-51002.pdf
- https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/ug_cyclone10_xcvr_phy.pdf
- https://www.intel.it/content/www/it/it/products/programmable/fpga/cyclone-10.html
- https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/arria-10-product-table.pdf
- https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/rn/rn_ip_81.pdf
- https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/signal.pdf
- https://www.xilinx.com/support/documentation/boards_and_kits/ac701/ug952-ac701-a7-eval-bd.pdf
- https://www.xilinx.com/support/documentation/boards_and_kits/kc705/ug810_KC705_Eval_Bd.pdf
- https://www.xilinx.com
- https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf
- https://www.xilinx.com/support/documentation/boards_and_kits/artix-7/ac701-schematic-xtp218-rev1-0.pdf
- https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug904-vivado-implementation.pdf
- https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug908-vivado-programming-debugging.pdf
- https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/chipscope_pro_sw_cores_ug029.pdf
- https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_2/pg172-ila.pdf
- https://www.xilinx.com/video/hardware/dashboards-in-vivado-logic-analyzer.html
- https://www.xilinx.com/support/documentation/sw_manuals/chipscope_pro_siotk_10_1_ug213.pdf

- https://www.xilinx.com/support/documentation/ip_documentation/axi_emc/v3_0/pg100-axi-emc.pdf
- https://www.xilinx.com/support/documentation/application_notes/xapp1176-xip-axi-quad-spi-ipi.pdf
- https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug994-vivado-ip-subsystems.pdf
- https://www.so-logic.net/documents/trainings/04_so_lab_commas.pdf
- https://www.xilinx.com/support/answers/57739.html
- https://www.xilinx.com/products/design-tools/vivado/simulator.html
- https://www.xilinx.com/video/hardware/logic-simulation.html
- http://users.wpi.edu/~rjduck/Vivado%20Simple%20VHDL%20Test%20Bench.pdf
- https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug750.pdf
- https://it.wikipedia.org/wiki/Diagramma_ad_occhio
- https://www.xilinx.com/support/documentation/application_notes/xapp1176-xip-axi-quad-spi-ipi.pdf