Università degli Studi di Padova
Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Implementazione su FPGA di un Sistema di Ricetrasmissione per Radiomicrofoni Professionali

Relatore: Ch.mo Prof. Lorenzo Vangelista
Correlatore: Ch.mo dr. Enzo Frigo

Laureando: Denis Testolina
Matricola: 607339 - IL

25 Ottobre 2011
Anno Accademico 2010/2011

Sommario

La presente tesi tratta l'implementazione di un sistema di ricetrasmissione per radiomicrofoni professionali che prevede l'utilizzo di tecnologie innovative quali modulazioni e trasmissione numerica. La trasmissione prevede l'impiego di due antenne al lato trasmissione e due al lato ricezione (trasmissione MIMO). L'implementazione é effettuata su Virtex 6 FPGA Embedded Kit e si basa sul lavoro effettuato dal dottor Alberto Vigato [1], in cui si sono analizzate diverse tecniche di trasmissione.

Ringraziamenti

Credo che questa sezione sia la più importante di tutta la tesi. Quando ho iniziato a pensare a tutte le persone che in questi anni hanno condizionato la mia vita, ho capito che finalmente ho raggiunto qualcosa di importante. Vi dirò che è stata veramente inattesa la sensazione di nostalgia che è nata in me quando ho iniziato a scrivere queste poche righe. Il pensare a tutte le avventure e i traguardi raggiunti mi ha fatto capire che di tutte le decisioni prese, di tutte le azioni fatte, non ne avrei cambiata nessuna.

Le prime persone che voglio ringraziare, sono le persone che hanno permesso tutto questo, i miei familiari.

Mi sono reso conto di quanto mi hanno voluto bene, compiendo enormi sacrifici solo per vedere esaudito un mio desiderio. Realizzando quanto hanno sofferto vedendomi soffrire e quanto hanno gioito vedendomi felice, ma soprattutto quanto mi hanno sopportato per le mie idee e atteggiamenti folli.

Naturalmente poi ringrazio i miei amici più cari che in questi anni mi hanno accompagnato e che mi sono sempre rimasti accanto, ma anche a tutti quelli, vicini e lontani, vecchi e nuovi, che hanno condiviso con me istanti unici e indimenticabili.

Ripensando poi al percorso fatto voglio ringraziare i miei compagni di università che hanno condiviso questi 5 anni tra fatiche e gioie, che con la loro amicizia mi hanno sempre spinto a dare il massimo. Ringrazio poi tutti i professori che in questi anni mi hanno avuto come studente, in particolare ringrazio il prof. Lorenzo Vangelista per i suoi insegnamenti e per l'aiuto datomi durante lo svolgimento della tesi. Infine ringrazio le persone che mi hanno dato un contributo particolare per lo svolgimento di questo lavoro il dr. Enzo Frigo, Wisycom SRL e per i numerosi ed essenziali consigli il dr. Alberto Vigato.

A tutte queste persone l'unica cosa che mi sento di dire è un semplice ma sentito GRAZIE.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Di seguito verranno spiegate le principali motivazioni che hanno portato alla scelta della modulazione digitale, spiegando non solo i pregi, ma anche le limitazioni imposte dalle normative vigenti. Si illustreranno, inoltre, le principali caratteristiche dell'FPGA utilizzata per l'implementazione. Nei capitoli seguenti, invece, ci si concentrerà sulla parte teorica e progettuale che ha portato all'implementazione finale.

1.1 Specifiche e Parametri

La scelta della trasmissione digitale è stata presa per avere un ulteriore passo verso la qualità non solo nella parte vocale, ma anche per un utilizzo di un più grande spettro musicale. Il problema principale della modulazione FM classica è dovuto alla poca dinamica del sistema trasmissivo che ne porta all'utilizzo di compressori analogici che sono difficilmente adattabili per tutte le sorgenti. Infatti si dovrebbero cambiare volta per volta i settaggi per non avere problemi e/o effetti fastidiosi non voluti.

Si è pensato dunque all'utilizzo della modulazione digitale, in cui la compressione del campione avviene in modo meno invasivo, trasmettendo gli effettivi bit campionati.

Si passa dunque da un sistema analogico con le seguenti caratteristiche:

- Modulazione FM;
- Ritardo massimo: $d_{\max} = 2\text{ms}$;
- Dinamica del segnale audio: $[0, -117]$ dBm;
- Banda del segnale audio: 20 Hz $\div$ 20 kHz;
- Frequenza della portante: 470 $\div$ 800 MHz;
- Massima deviazione del segnale modulato: 56 kHz;
- Distorsione (effetto delle non linearità residue): $\leq 0.1\%$ (60 dB)
- Rapporto segnale su rumore: $\text{SNR}_{\max}$ 80 dB.
- Numero antenne in trasmissione: 1;

- Numero antenne in ricezione: 2.

Ad un sistema digitale che deve rispettare le normative ETSI vigenti come il seguente:

- Modulazione Numerica;
- Ritardo massimo: $d_{\max} = 2$ms;
- Dinamica del segnale audio: $[0, -117]$ dBm, $\geq$117 dB;
- Banda del segnale audio: 20 Hz $\div$ 20 kHz;
- Frequenza della portante: 470 $\div$ 800 MHz;
- Banda del segnale modulato: $B = 200$ kHz;
- Numero antenne in trasmissione: 2;
- Numero antenne in ricezione: 2.

La maschera per lo spettro del segnale trasmesso dovrà rispettare i vincoli imposti dalla maschera in Fig. 1.1 imposto dalla normativa europea ETSI [3].

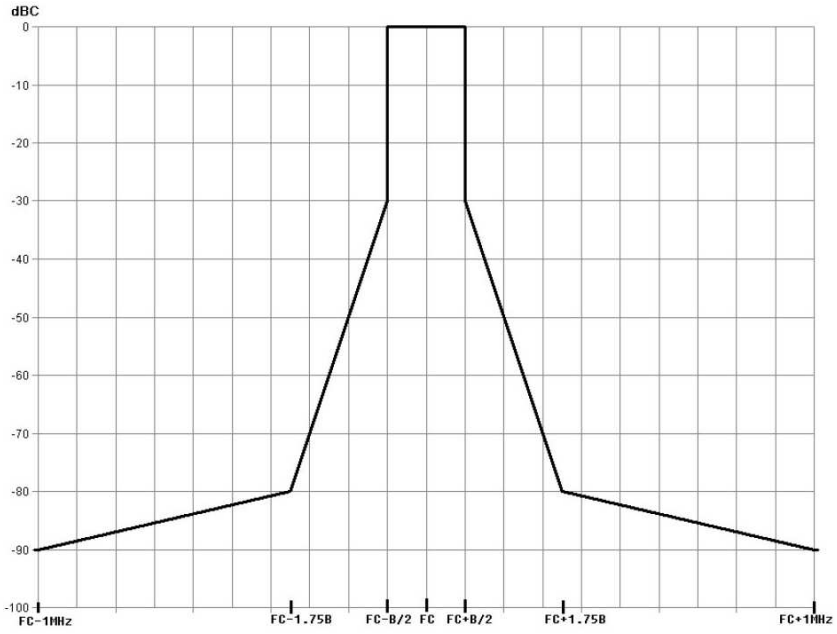


Figura 1.1: Maschera dello spettro per sistemi radiomicrofono digitali.

1.2 L'FPGA: Virtex 6

Un Field Programmable Gate Array (FPGA) è un circuito integrato progettato per essere configurato dall'utilizzatore oppure subire una progettazione successiva alla produzione. La configurazione dell'FPGA è generalmente specificata

utilizzando un linguaggio di descrizione hardware(HDL), simile a quello utilizzato per un circuito integrato per applicazioni specifiche (ASIC)(diagrammi circuitali sono stati utilizzati per specificare la configurazione, come fossero degli ASIC, ma questo tipo di progettazione è sempre più raro). L'FPGA puó essere utilizzato per implementare qualsiasi funzione logica che potrebbe eseguire un ASIC. Ma la possibilità di aggiornare la funzionalità dopo la configurazione, o di una parziale riconfigurazione di una porzione del design e il basso costo relativo alle ingegnerizzazioni non ricorrenti rispetto a un design basato su ASIC (nonostante il costo unitario generalmente superiore), offrono innumerevoli vantaggi per molte applicazioni.

L'FPGA contiene componenti logici programmabili, chiamati ″Logic Blocks″, e una gerarchia di interconnessioni riconfigurabili che permettono i blocchi essere ″collegati assieme″. Questo metodo puó essere paragonato a raggruppare molte porte logiche (modificabili) che possono essere interconnesse in diverse configurazioni. I Logic Blocks possono essere configurati per eseguire complesse funzioni combinatorie, o porte logiche semplici come AND o XOR. Nella maggior parte degli FPGA, i Logic Blocks includono anche elementi di memoria, che possono essere semplici Flip-Flops o più completi blocchi di memoria.

Oltre alle funzioni logiche digitali, alcuni FPGA hanno caratteristiche analogiche. La funzionalità analogica più comune è lo slew rate programmabile e la regolazione della potenza d'uscita su ogni pin, permettendo all'utente di impostare variazioni lente sui pin poco caricati che altrimenti andrebbero fuori controllo, oppure una variazione più veloce e con più potenza su un pin con carico elevato o su canali ad alta velocità che altrimenti sarebbero condizionati e d avrebbero variazioni troppo lente per essere identificate come tali.

Un'altra caratteristica analogica relativamente comune è l'utilizzo degli ingressi differenziali. Alcuni FPGA a ″segnali misti″ hanno inoltre integrato periferiche di conversione analogico-digitale (ADC) e/o digitale-analogico (DAC) con blocchi di condizionamento del segnale analogico che permettono loro di operare come dei system-on-a-chip. Tali dispositivi sono un misto tra gli FPGA, che trasportano segnali digitali sulle interconnessioni interne programmabili e quelli che trasportano solo segnali analogici Field Programmable Analog Array(FPAA).

1.2.1 Architettura

L'architettura FPGA più comune è costituita da una matrice di Logic Blocks (chiamati Configurable Logic Block, CLB, o Logic Array Block, LAB, a seconda del fornitore), pad di input e output e canali di routing (ref fig. 1.2). In generale, tutti i canali di routing hanno la stessa larghezza (numero di connessioni). I pad di I/O multipli, invece, possono adattarsi all'altezza di una riga o la larghezza di una colonna nella matrice.

Un circuito applicativo deve essere mappato in un FPGA con risorse adeguate. Mentre il numero di blocchi CLB/LAB e pad I/O sono facilmente determinati dalla progettazione, il numero di tracce di routing necessarie, invece, puó variare notevolmente anche tra disegni con la stessa quantità di logica. Poichè le tracce di routing inutilizzate provocano un aumento del costo (e la

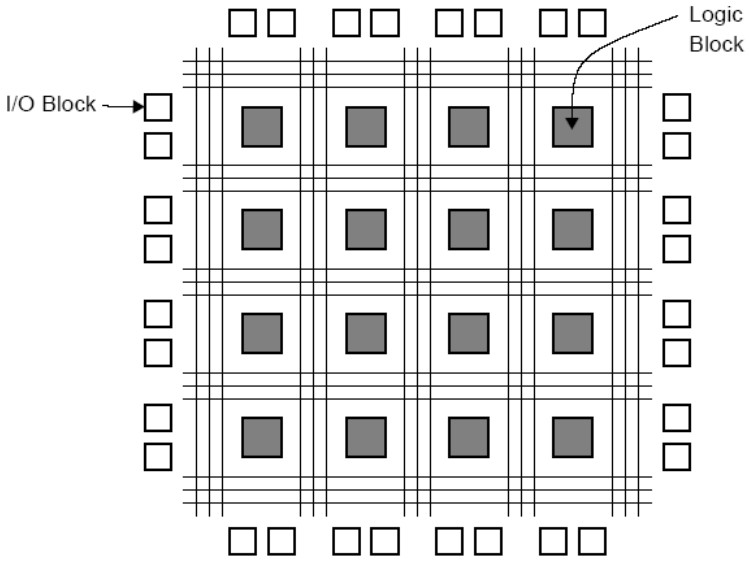


Figura 1.2: Esempio della struttura interna di un FPGA.

diminuzione delle prestazioni) dell'implementazione senza fornire alcun beneficio. I produttori di FPGA tentano di fornire abbastanza tracce di routing, in modo che la maggior parte dei disegni, che si adattano in termini di LUTs e pin I/O possano essere indirizzati. Questo è determinato dalle stime effettuate o dagli esperimenti con i disegni esistenti.

In generale, un blocco di logica (CLB o LAB) è costituito da poche unità logiche (chiamate ALM, LE, Slice ecc.). Una cella tipica è costituita da una tabella di ricerca o Lookup table 4 input (LUT), un Full adder (FA) e un flip-flop di tipo D, come illustrato di seguito (ref. fig 1.3). Le LUT sono, in questa figura, divise in due LUT a 3 ingressi. In modalità *normale* quelle vengono combinate in una LUT a 4 ingressi attraverso il multiplexer a sinistra. In modalità *aritmetica*, le loro uscite sono connesse al FA. La selezione della modalità è programmata dal multiplexer centrale. L'uscita puó essere sincrona o asincrona, nell'esempio di figura è determinato dalla programmazione del multiplexer a destra. In pratica, per risparmiare spazio tutte o parti delle operazioni del FA sono integrate come funzioni della LUT.

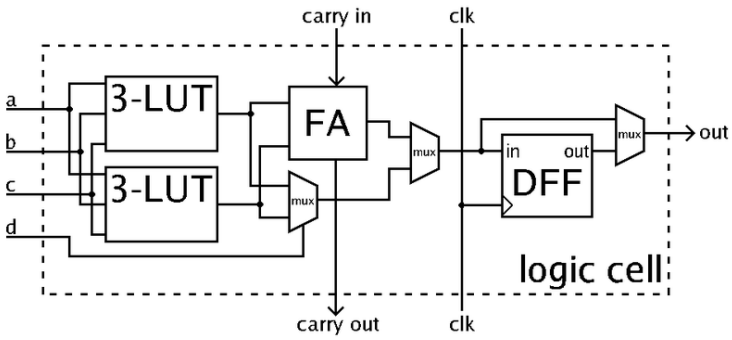


Figura 1.3: Esempio semplice di Logic Block.

ALM e Slice di solito contengono 2 o 4 strutture simili a quelle riportate

in figura, con alcuni segnali condivisi. I Blocchi logici CLB/LAB in genere contengono qualche ALM/LE/Slice. Spesso i segnali di clock (o altri segnali con fronti ripidi) vengono instradati tramite speciali reti di routing dedicate all'interno dell' FPGA commerciale, ed inoltre vengono gestiti separatamente.

Per questa architettura di esempio, le posizioni dei pin di un Logic Block sono indicate qui sotto.

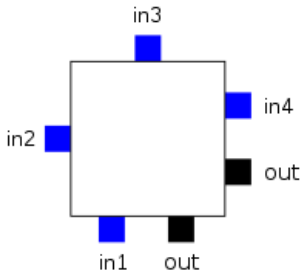


Figura 1.4: Locazione di pin di un Logic Block.

Ogni ingresso è accessibile da un lato del Logic Block, mentre il pin di uscita possono connettersi alle connessioni di routing del canale a destra e sotto il Logic Block. Ogni pin di uscita del Logic Block puó connettersi a uno qualsiasi dei segmenti di cablaggio presente nei canali adiacenti ad esso. In generale, le connessioni dell'FPGA non sono segmentate. Cioè, ogni segmento di cablaggio si estende in un solo Logic Block prima che termini in una switch box. Cambiando alcuni parametri all'interno di una switch box, si possono creare percorsi più lunghi. Per un interconnessione più veloce, alcune architetture di FPGA utilizzano connessioni a più linee che si estendono su più Logic Block.

Ogni volta che un canale verticale e orizzontale si intersecano, c'è una switch box. Nell'architettura presa in esame, quando una connessione entra in una switch box, ci sono tre interruttori programmabili che consentono di connettersi a tre altre connessioni in segmenti del canale adiacente. La tipologia che comanda gli interruttori della switch box è di tipo *planare* o basata sui domini. Ogni interruttore puó connettere sono due connessioni adiacenti, il primo filo con il primo filo, il secondo con il secondo, e cosí via. La figura seguente illustra le connessioni in una switch box.

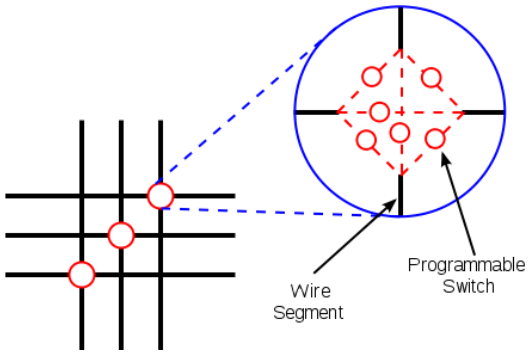


Figura 1.5: Tipologia di Switch Box.

Le moderne famiglie FPGA espandono le funzionalità riportate sopra, per includere funzionalità di livello superiore. Le funzionalità vengono integrate nel chip di silicio donandogli maggiore velocità e minor consumo di area rispetto ad una costruzione tramite Logic Block. Ad esempio si integrano moltiplicatori, blocchi generici DSP, processori embedded, logica I/O ad alta velocità e memorie embedded.

Gli FPGA sono anche ampiamente utilizzati per la convalida dei sistemi tra cui convalida pre-silicon, post-silicon convalida e lo sviluppo di firmware. Questo permette alle aziende di chip di convalidare il loro design prima della produzione, riducendo il time-to-market.

1.2.2 Virtex 6

Per l'implementazione abbiamo utilizzato un Embedded Kit della Xilinx. è composto da una scheda base l'ML605 il cui componente principale è un FPGA Virtex 6 XC6VLX240T-1FFG1156.

Figura 1.6: Virtex 6 FPGA Embedded Kit ML605.

Questo tipo di FPGA fa parte della famiglia delle Virtex 6 che hanno dimostrato un'ottima versatilità. L'XC6VLX240T contiene al suo interno:

- 241152 celle logiche
- 37680 Slices
- 3650Kb di RAM distribuita
- 768 Slices DSP48

Nello specifico modello con package FFG1156 si hanno a disposizione 600 pin di I/O e 20 Pin dedicati a segnali con fronti ripidi e clock (GTXs).

La configurazione dell'FPGA viene memorizzata all'interno di memoria di tipo SRAM. Il numero di bit per una configurazione tipica è indipendente dal design specifico deciso dall'utente, ed è di tipo volatile e deve essere ricaricata ad ogni accensione dell'FPGA.

Le LUTs possono essere configurate come delle LUTs a 6 ingressi, oppure, come due LUTs a 5 ingressi, ma con logica di input comune. Ogni uscita della LUT puó essere memorizzata tramite un flip-flop dedicato. Quattro di queste LUTs, i loro otto Flip-Flop associati come i multiplexer e la logica di riporto formano una slice, due slice formano un Logic Block (CLB).

Normalmente le applicazioni di digital signal processing utilizzano molti moltiplicatori e accumulatori binari, in questa tipologia di FPGA vi sono delle slice dedicate a questo scopo. Le slice DSP48 sono formate fondamentalmente da un moltiplicatore 25x18 complemento a due e da un accumulatore da 48-bit, entambi hanno la possibilità di operare a 600Mhz. Il moltiplicatore puó essere aggirato in modo dinamico, e due blocchi da 48 bit possono generare un'unità aritmetica single-instruction-multiple-data (SIMD) (doppio sommatore/sottrattore/accumulatore da 24-bit o un quadruplo sommatore/sottrattore/accumulatore a 12-bit), o un'unità logica che puó generare una qualsiasi delle 10 funzioni logiche differenti dei due operandi.

Questa FPGA ha reso possibile una ottima versatilità nell'implementazione del progetto portando a risultati voluti.

Capitolo 2

Trasmettitore

In questa sezione si presenterà parte dell'analisi effettuata dal dottor Alberto Vigato per la trasmissione digitale del segnale audio [1] e si illustrerà, inoltre, un'implementazione del trasmettitore. Descriviamo il sistema di trasmissione nei suoi vari aspetti: l'elaborazione numerica del segnale, la modulazione numerica, e le specifiche di trasmissione RF.

Analizziamo ora come realizzare il sistema utilizzano trasmissione numerica. Lo schema di trasmissione (microfono wireless) è riportato in Fig. 2.1.

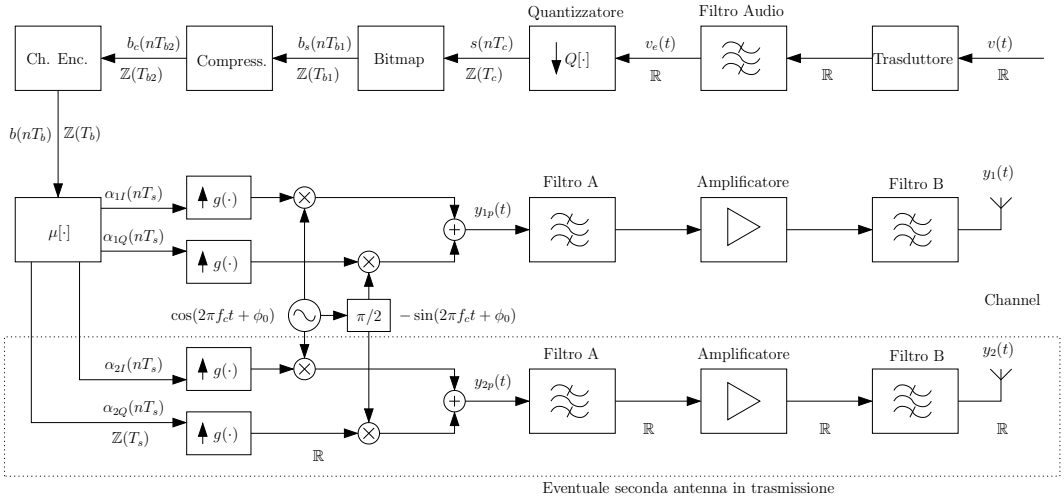


Figura 2.1: Schema a blocchi del trasmettitore.

Come si può osservare dallo schema una parte del sistema lavora nel dominio dei tempi discreti $\mathbb{Z}(T)$, caratteristica tipica dei sistemi numerici. La parte tratteggiata rappresenta il percorso relativo alla seconda antenna in trasmissione. Ciascuna parte del sistema viene analizzata singolarmente nei paragrafi che seguono.

La parte alta della catena realizza la parte di elaborazione numerica del segnale, ovvero tutto quello che occorre effettuare sul segnale fisico perchè esso venga tradotto in qualcosa di trasmissibile e codificabile. Il punto chiave della trasmissione numerica invece è dato dal blocco $\mu[\cdot]$ che trasforma il flusso di bit nell'unità fondamentale di trasmissione, ovvero il frame che analizzeremo

nel dettaglio nel paragrafo 2.2. segue la modulazione vera e propria del segnale che trasforma i dati numerici in un segnale fisico che viene mandato in aria e deve perciò rispettare le specifiche di trasmissione.

2.1 Elaborazione e Modulazione del segnale

2.1.1 Elaborazione del segnale

Per prima cosa si utilizzerà un trasduttore adeguato per rispettare la dinamica richiesta al sistema, successivamente dopo operazioni di filtraggio per eliminare eventuali componenti non desiderate, il segnale verrà campionato.

La qualità della trasmissione è data in primo luogo dalla codifica del segnale audio. Per avere una alta dinamica si devono avere un numero di bit utili adeguato per soddisfare a

$$10 \log_{10} |2^{N_b}|^2 \geq 117 \text{ dB} \tag{2.1}$$

da cui si ricava

$$N_b \geq \frac{117}{20 \log_{10}(2)} \simeq 19.43 \geq 19 \text{ bit.} \tag{2.2}$$

Il rate di campionamento, in conformità alla banda del segnale audio, sarà di 44.1 kHz (frequenza di campionamento del CD audio) o di 48.0 kHz (frequenza di campionamento dei DVD e nel digitale terrestre). I valori di bitrate non compresso sono riportati in Tab. 2.1.

bit/s	Dynamic [dB]	Bitrate [kb/s]	
		44.1 kHz	48.0 kHz
18	108.37	793.8	864.0
19	114.39	837.9	912.0
20	120.41	882.0	960.0
21	126.43	926.1	1,008.0
22	132.45	970.2	1,056.0
23	138.47	1,014.3	1,104.0
24	144.49	1,058.4	1,152.0

Tabella 2.1: Bitrate lordo per audio campionato.

Compressore

Il basso ritardo minimo accettato impedisce di usare molti comuni algoritmi di compressione basati sullo spettro (percettivi – mp3). Tuttavia può essere preso in considerazione l'utilizzo di tecniche di compressione a basso ritardo come ADPCM che garantisce un ottima qualità del segnale nel range di compressione $[1/2 \div 1/4]$. I range di bitrate netti necessari è illustrato in Tab. 2.2

bit/s	Dynamic	Bitrate ADPCM [kb/s]	
	[dB]	48.0 kHz (1/2)	44.1 kHz (1/4)
24	144.49	576.0	264.6
23	138.47	552.0	253.6
22	132.45	528.0	242.6
21	126.43	504.0	231.5
20	120.41	480.0	220.5
19	114.39	456.0	209.5
18	108.37	432.0	198.5

Tabella 2.2: Bitrate per audio compresso ADPCM.

Codifica di Canale

Sempre a causa del basso ritardo minimo non possono essere adottate tecniche FEC o interleaving su sequenze di dati troppo lunghe. Tuttavia può essere considerata una strategia di tipo CRC o Parity bits per valutare l'integrità di piccoli pacchetti di dati.

2.1.2 Modulazione

L'operazione di modulazione avviene in due fasi. In primo luogo i bit di informazione vengono trasformati in opportuni valori numerici complessi appartenenti ad un insieme detto costellazione: qui si parla di mappatura numerica. La seconda fase prevede di ristrutturare questi numeri secondo un'altra mappatura detta di space–time (STBC) che adatta il flusso di dati per una trasmissione MIMO. Quindi i dati vengono incapsulati nell'unità di trasmissione che viene definita come Frame. Infine vengono interpolati in una forma d'onda opportuna che deve rispettare le specifiche di banda.

Mappatura

La mappatura numerica prevede la conversione dei dati binari in simboli complessi. I simboli compresivi di dati e overhead, vedi paragrafo 2.2, verranno successivamente trasmessi una volta modulati in banda passante.

La scelta della costellazione è importante perchè definisce sia la densità spettrale del sistema che la potenza di lavoro richiesta in termini di SNR per raggiungere un determinato livello di affidabilità della trasmissione misurata in probabilità d'errore sul bit (BER) [1].

Le costellazioni che prendiamo in considerazione sono QPSK e 8PSK che hanno la proprietà di essere a inviluppo costante[1] oppure la 16APSK che agisce su due livelli di ampiezza ma garantisce un efficienza spettrale maggiore. In Fig. 2.2 vediamo le tre costellazioni con mappatura di tipo Gray per minimizzare il BER.

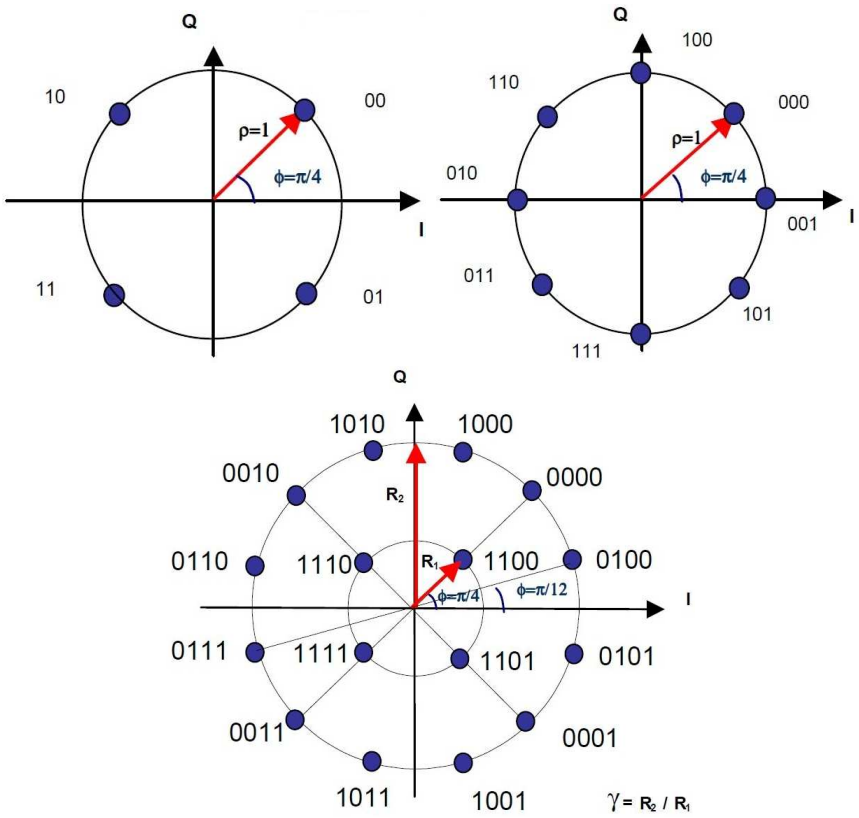


Figura 2.2: Costellazioni Q/8/16APSK con mappatura da DVB–S2.

Queste costellazioni sono prese dallo standard del broadcast televisivo digitale satellitare di seconda generazione DVB–S2 definito in [4].

Per quanto riguarda la scelta del rapporto $\gamma = R_2/R_1$ nella 16APSK occorre tenere presente due vincoli:

1. l'energia media della costellazione deve essere unitaria e ció è garantito se è soddisfatta l'equazione

$$4R_1^2 + 12R_2^2 = 16 \tag{2.3}$$

ovvero imponendo che la dimensione del raggio interno sia vincolata dal valore di quello esterno in base alla relazione

$$R_1 = \sqrt{4 - 3R_2^2}; \tag{2.4}$$

[1]Le costellazioni ad inviluppo costante facilitano la detection limitandola alla sola informazione di fase del simbolo ricevuto

2. la scelta del raggio esterno ottimale viene fatta imponendo che la minima distanza tra punti della circonferenza interna (punti adiacenti) eguagli quella dei punti della circonferenza esterna per garatire ottimalità sulla probabilità d'errore. Questo si ottiene ponendo

$$2R_1^2 = \left(2 + \sqrt{3}\right) R_2^2 \tag{2.5}$$

e sostituendo il primo vincolo sulla potenza media si ricavano i valori normalizzati sull'energia media

$$R_1 = \sqrt{\frac{8 - 4\sqrt{3}}{8 - \sqrt{3}}} \simeq 0.4139 \tag{2.6}$$

$$R_2 = \sqrt{\frac{8}{8 - \sqrt{3}}} \simeq 1.1297 \tag{2.7}$$

Di conseguenza, con tali valori il rapporto ottimo vale $\gamma_{\mathrm{OPT}} = 2.7294$.

Al fine di impedire gli effetti di non linearità attorno allo zero degli amplificatori in trasmissione, potrebbe risultare necessario usare mappature di tipo differenziale o a costellazioni sfasate a tempi alterni.

In Fig 2.3 viene evidenziato come per modulazioni di tipo differenziale di possano ottenere transizioni che mai passano per lo zero a differenza delle modulazioni classiche che dovutamente alla loro simmetria necessariamente presentano transizioni che passano per lo zero.

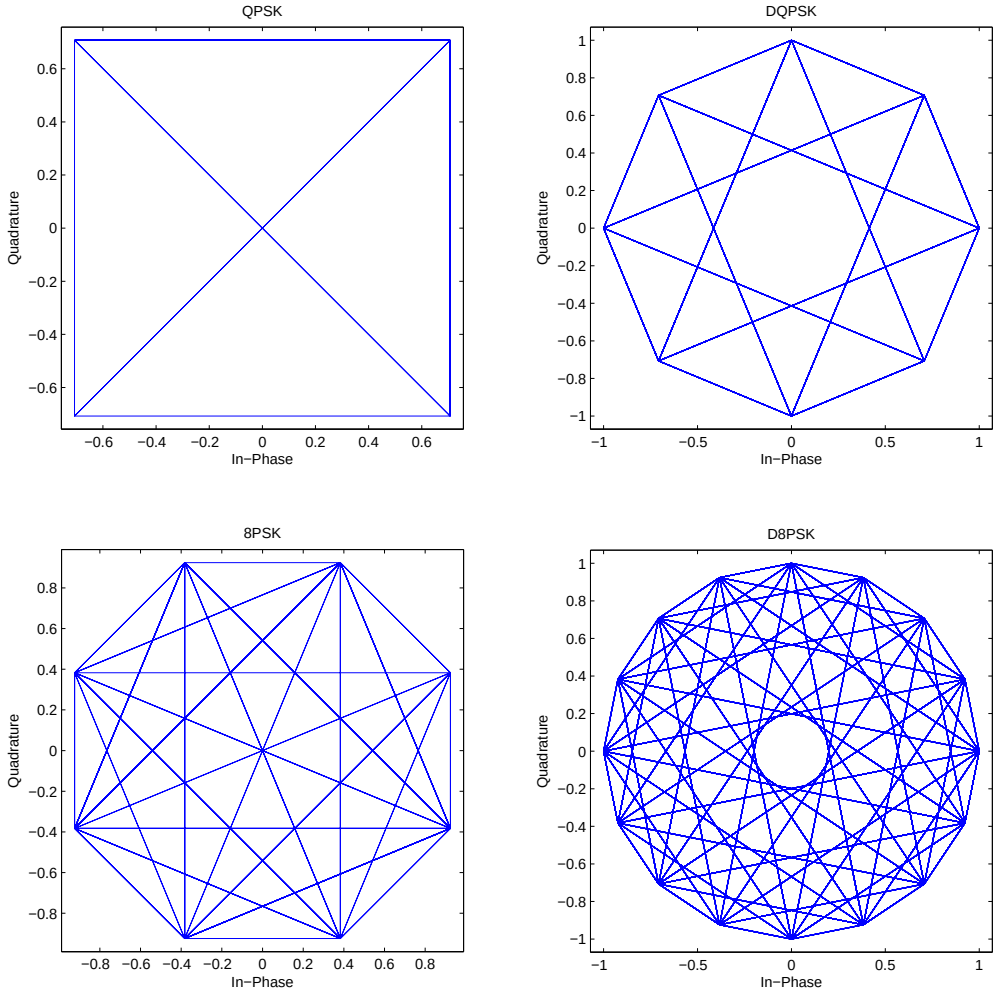


Figura 2.3: Transizioni sulle costellazioni QPSK e 8PSK, a sinistra in versione classica e a destra in versione differenziale.

2.1.3 STBC

Si pensa di sviluppare il progetto con due antenne in trasmissione due in ricezione. La codifica per le due antenne scelta è la codifica Alamouti 2×2 [5].

Questa codifica ha un design molto semplice la cui matrice di codifica space-time ha l'espressione

$$S_A(\boldsymbol{\alpha}) = \frac{1}{\sqrt{2}} \begin{bmatrix} \alpha_1 & \alpha_2 \\ -\alpha_2^* & \alpha_1^* \end{bmatrix} \tag{2.8}$$

che ha il difetto di essere un codice non full–rate ma ha il pregio di non alterare la forma delle costellazioni trasmesse.

2.2 Il Frame

Dopo aver costruito il flusso numerico dei dati di informazione, questi ultimi vanno opportunamente incapsulati in frame. La struttura generale del frame che andremo a utilizzare è caratterizzata da un preambolo, dei dati pilota e si presenta come in Fig. 2.4.

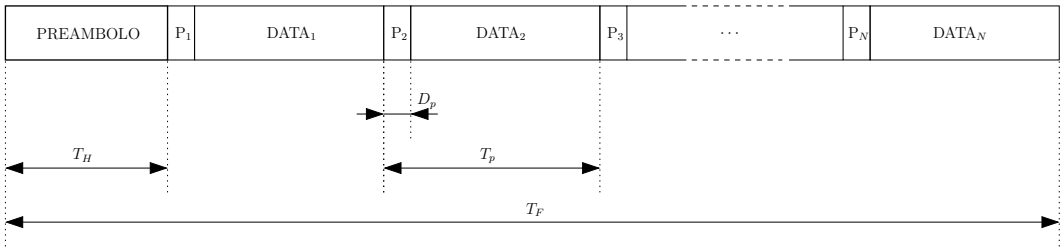


Figura 2.4: Frame di trasmissione audio digitale.

Il frame avrà una durata temporale T_F che sarà scelta in funzione del massimo ritardo che possiamo accettare dall'accensione del dispositivo, ovvero da eventuali perdite di sincronizzazione tra microfono e ricevitore, e l'effettivo avvio della ricetrasmissione real–time.

Trattandosi di trasmissione MIMO con due antenne in trasmissione, il frame sarà composto da due linee parallele di trasmissione ciascuna delle quali costruita come in Fig. 2.4.

La testa del frame è formata da quello che viene chiamato preambolo che è necessario per:

- la temporizzazione[2];
- la sincronizzazione di frame;
- la sincronizzazione di simbolo.

Per talune ragioni, il preambolo dovrà essere una sequenza di una consistente lunghezza temporale $T_H = N_H T_s$ ma non eccessiva da incappare in overheading sovrabbondante che causa eccessivi ritardi e riduzione del throughput. Altri aspetti che possono riguardare il preambolo sono: un'accurata stima iniziale di canale; la correzione di eventuali offset di frequenza che potrebbero presentarsi a causa del non perfetto allineamento in frequenza esistente tra l'oscillatore in trasmissione e quello in ricezione.

Per inseguire la tempovarianza del canale, saranno inserite in mezzo alla sequenza, con cadenza periodica T_p, delle sequenze pilota P, molto semplici e note. Trattandosi di canale MIMO 2×2 a singolo tappo, sono sufficienti sequenze pilot della durata $D_p = 2T_s$. Anche qui per il valore di T_p deve essere un giusto compromesso ossia dovrà essere grande abbastanza per non causare eccessivo overheading ma allo stesso tempo dovrà essere sufficientemente breve da riuscire a inseguire adeguatamente le variazioni del canale.

[2]Scelta dell'istante di campionamento noto anche come *timing*.

Infine le zone che abbiamo indicato con DATA sono le parti del frame effettivamente riservate alla trasmissione del segnale audio digitale. L'efficienza complessiva del frame sarà dato quindi dalla relazione

$$\eta_F = \frac{T_F - T_H - ND_p}{T_F} = 1 - \frac{T_H + ND_p}{T_F} \tag{2.9}$$

dove $N = (T_F - T_H)/T_p$ è il numero di pilot P presenti in un singolo frame.

Avendo già discusso del contenuto nella parte dati del frame, analizzeremo ora la struttura del preambolo e delle pilot che adotteremo nell'implementazione del sistema.

Preambolo

Il preambolo si basa sulle sequenze di Golay [7] che possono essere costruite in maniera tale da poter realizzare un correlatore semplice, come spiegato in [8]. Le stesse tecniche di stima proposte in [9, 11] si basano su tali sequenze e relativi correlatori.

Al fine di ridurre la latenza da preambolo e aumentare la risoluzione in frequenza per frequency offset di entità maggiore, si ricorre a sequenze di lunghezza 64 tenendo presente che in questo caso i picchi risulteranno di 3 dB meno pronunciati, rispetto al caso con sequenze di lunghezza 128 presentati in [1], andando ad aumentare le probabilità di *false allarm* o di *peak undetection*.

Seguendo la strategia proposta da Popovic [8] utilizzando la sequenza generatrice e la sequenza dei ritardi che segue

$$W_n = [+1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; -1] \tag{2.10}$$

$$D_n = [32 \quad 8 \quad 16 \quad 1 \quad 2 \quad 4] \tag{2.11}$$

I preamboli saranno del tipo

$$\Pi_A = \frac{\boldsymbol{\alpha}}{\sqrt{2}}, \tag{2.12}$$

$$\Pi_B = \frac{\boldsymbol{\beta}}{\sqrt{2}}. \tag{2.13}$$

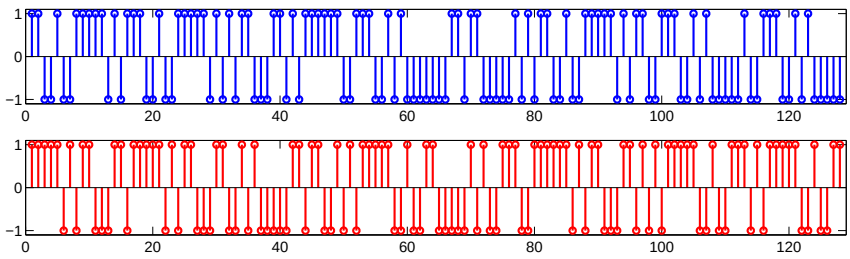


Figura 2.5: Sequenze complementari di Golay con generatori W_n e D_n.

L'autocorrelazione di $\boldsymbol{\alpha}$ e la cross–correlazione normalizzate sono invece illustrate in Fig. 2.6

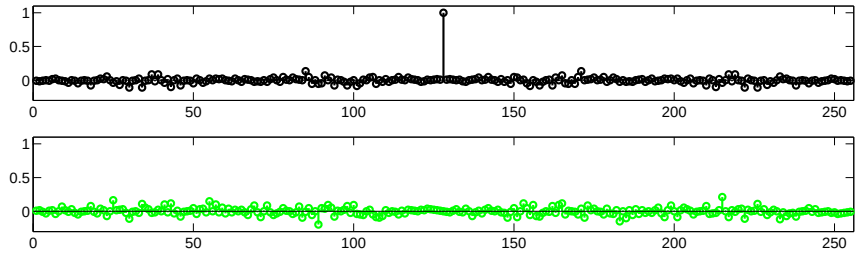


Figura 2.6: Autocorrelazione $r(\boldsymbol{\alpha}, \boldsymbol{\alpha})$ (sopra) e cross–correlazione $r(\boldsymbol{\alpha}, \boldsymbol{\beta})$ (sotto) normalizzate.

Alcuni aspetti relativi a questa tipologia di preambolo sono: manca il prefisso ciclico al fine di ridurre ulteriormente la durata di questi preamboli; essendo sequenze più brevi non sono dotati di offset di tipo $+j\mathbf{1}$ che affievolirebbe troppo il valore di picco di correlazione; di controparte questo comporta che il segnale in trasmissione durante il preambolo attraversa ripetutamente lo zero.

Pilots

La sequenza pilota o pilot P è formata da simboli noti trasmessi su due slot temporali di simbolo che equivalgono ad un blocco STBC. I pilot vengono trasmessi a intervalli regolari. La struttura proposta è illustrata in Fig. 2.7.

$\cdots$	$d_1 \quad d_2$	$d_3 \quad d_4$	$p_{11} \quad p_{12}$	$d_5 \quad d_6$	$d_7 \quad d_8$	$\cdots$
$\cdots$	$-d_2^* \quad d_1^*$	$-d_4^* \quad d_3^*$	$p_{21} \quad p_{22}$	$-d_6^* \quad d_5^*$	$-d_8^* \quad d_7^*$	$\cdots$

Figura 2.7: Struttura della sequenza pilota.

In tale struttura riconosciamo la sequenza pilota che puó essere rappresentata da una matrice P space–time. Nel caso di trasmissione DQPSK, per mantenere in trasmissione la costellazione di Fig. 2.3 con le stesse transizioni, possiamo usare la matrice

$$P = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} = \frac{\mathcal{E}_s}{\sqrt{2}} \begin{bmatrix} 1 & e^{j\frac{3}{4}\pi} \\ e^{j\frac{3}{4}\pi} & 1 \end{bmatrix}. \tag{2.14}$$

La forma della matrice è dettata dal fatto di voler evitare il passaggio per lo zero tra due simboli consecutivi. Si notano subito alcune proprietà fondamentali in questa matrice:

- è formata da simboli ad ampiezza costante dello stesso valore dei simboli di dati d_i mantenendo omogeneo il flusso di trasmissione;
- la matrice P è invertita da una matrice altrettanto semplice

$$P^{-1} = \mathcal{E}_s^{-1} \begin{bmatrix} e^{-j\frac{\pi}{4}} & e^{-j\frac{\pi}{2}} \\ e^{-j\frac{\pi}{2}} & e^{-j\frac{\pi}{4}} \end{bmatrix}. \tag{2.15}$$

rendendo l'operazione di aggiornamento della stima di canale semplice e immediata.[3]

2.3 Filtro Interpolatore

Lo schema di interpolazione deve essere tale da garantire che l'impulso in banda base ottenuto soddisfi ai requisiti imposti dalla maschera in trasmissione. Per fare ciò ricorreremo ad un'interpolazione dei simboli α_{nN} (con $n = 1, 2$ e $N = I, Q$) ad un dominio discreto più veloce e in seguito, tramite un convertitore digitale/analoogico DAC passeremo al dominio continuo. Lo schema equivalente di questa struttura è dato in Fig. 2.8.

Figura 2.8: Schema equivalente dell'interpolatore.

Un DAC che potrebbe servire ai nostri scopi è l'AD9707 che opera ad una frequenza di campionamento massima pari a $F_{DAC} = T_{DAC}^{-1} = 175$ MHz e converte segnali numerici con precisione 14 bit. Il filtro passa–basso a valle del sistema serve per eliminare le repliche in frequenza del segnale a multipli di F_{DAC} e sarà progettato in funzione a questo scopo.

Criterio di Nyquist

Per minimizzare l'ISI (interferenza di intersimbolo) si può imporre il criterio di Nyquist ovvero progettare $g(t)$ che soddisfi a

$$g * g(nT_s) = 0 \ \forall n \neq 0 \tag{2.16}$$

ossia autocorrelato (correlazione con l'impulso di matching) e campionato non dia luogo a repliche fuori dal periodo di decisione. Tale proprietà nel dominio della frequenza si traduce in $\mathrm{rep}_{F_s}|G(f)|^2 \equiv 1$. Il rispetto di questo criterio permette quindi un a durata dell'impulso lunga a piacere vincolata dal solo ritardo massimo di sistema.

Per fare ciò si sceglie l'impulso di Nyquist equiripple a fase minima (non a fase lineare) e per soddifare i criteri della maschera occorre limitarsi a una frequenza di simbolo pari a $F_s = 140$ kHz, che impone un tempo di simbolo $T_s = 7.143$ μs, ciò garantisce un bitrate lordo secondo quanto riportato in Tab. 2.3.

A causa dei limiti di progettazione dei filtri di Nyquist a fase minima, occorre spezzare la progettazione dell'impulso in due fasi come illustrato in Fig. 2.9.

[3]Ricordiamo che la stima di canale comporta la valutazione dei quattro valori della matrice di canale $H(t)$.

Mappatura	Max Bitrate [kb/s]
QPSK	280.00
8PSK	420.00
16APSK	560.00

Tabella 2.3: Bitrate lordo massimo trasmissibile per un segnale numerico limitato dalla maschera di Fig. 1.1 usando impulso di Nyquist.

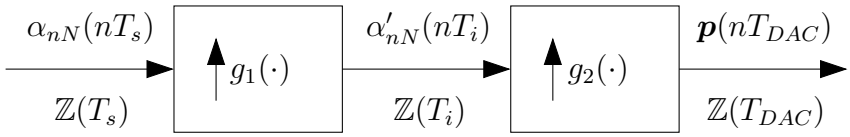


Figura 2.9: Schema progettazione in cascata del filtro interpolatore $g(t)$.

Imponendo la frequenza $F_i = 1/T_i = 700$ kHz otteniamo quello che e' l'impulso di Nyquist $g_1(nT_i)$ a fase minima di ordine 42. Tale forma d'onda va successivamente interpolata nel dominio temporale del DAC che per commensurabilità tra valori lo scegliamo con frequenza di conversione $F_{DAC} = 21$ MHz così da avere $F_{DAC}/F_i = 30$.

Il secondo impulso $g_2(nT_{DAC})$ viene progettato in modo da soddisfare i requisiti di maschera, e l'abbiamo ottenuto come filtro di Nyquist normale di ordine 226.

L'impulso è raffigurato nel tempo e in frequenza in Fig. 2.10.

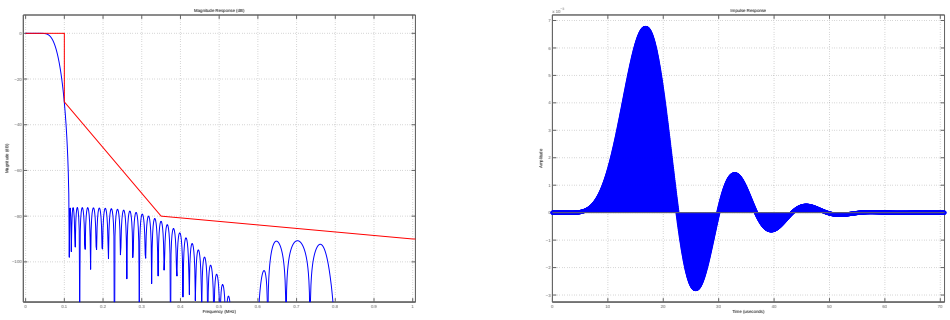


Figura 2.10: Sinistra: modulo della risposta in frequenza dell'impulso di Nyquist a fase minima $|G(f)|$ (blu) che soddisfa i requisiti della maschera (rosso). Destra: impulso $g(t)$ nel dominio del tempo.

Tale impulso è ottenuto come convoluzione interpolata nel dominio più fitto $\mathbb{Z}(T_{DAC})$ tra le due forme d'onda

$$g(nT_{DAC}) = g_1 * g_2(nT_{DAC}) = \sum_k g_1(nT_{DAC} - kT_{DAC}) g_2(kT_{DAC}) \qquad (2.17)$$

dove naturalmente $g_1(nT_{DAC} - kT_{DAC}) = 0$ ogniqualvolta $n - k$ non è multiplo di 30 essendo tale impulso definito nei tempi lenti. Il risultante è un impulso di 1487 campioni che danno una durata complessiva

$$d(g) = 1487 T_{DAC} = 70.81 \mu\text{s} \ (\simeq 10 T_s) \qquad (2.18)$$

Salta subito all'occhio la mancata simmetria che ha l'impulso nel tempo. Questo comporta che la fase dell'impulso non è lineare e il filtro in ricezione deve essere ottenuto per ribaltamento $g_-(t) = g(-t)$, sarà quindi a fase massima.

Rimane da valutare l'impatto di interferenza di intersimbolo che questo impulso genera. Tale effetto è rappresentato in Fig. 2.11

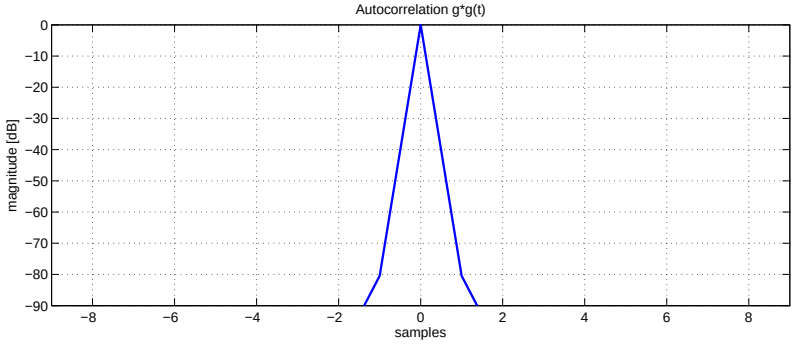


Figura 2.11: Autocorrelazione dell'impulso di Nyquist $g * g_-(nT_s)$ nel dominio del tempo campionato sul periodo di simbolo T_s.

Vediamo dunque un impatto dell'ISI molto ridotto in questo caso e riportiamo in Tab. 2.4 un confronto tra l'Impulso di Nyquist e l'impulso sinc finestrato.

Type	F_s [kHz]	length [T_s]	ISI_{max} [dB]	ISI cum. [dB]
Window	180	22	29.03	6.33
Nyquist	140	10	**80.45**	**73.64**

Tabella 2.4: Confronto tra impulsi sugli effetti dell'ISI.

Ricapitolando con impulsi di Nyquist, essendo che a causa di vincoli di progettazione più forti risulta necessario ridurre il rate di trasmissione, dall'altro lato abbiamo che possiamo trasmettere impulsi di durata quasi dimezzata (più che dimezzata se si tiene conto della sovrapposizione sui periodi di simbolo contigui) e con un impatto quasi ISI free.

2.4 Sezione RF

Il/i segnale/i $y_{ip}(t)$ dovrebbe essere tale da soddisfare i requisiti spettrali per cui il Filtro A non dovrebbe essere necessario.

L'amplificatore sarà un dispositivo in grado di amplificare in maniera lineare[4] all'interno del range di trasmissione.

Per quanto riguarda il Filtro B dovrebbe essere necessario per mitigare gli effetti fuori banda dovuti alle non linearità residue dell'amplificatore in trasmissione. La risposta in frequenza del Filtro B è illustrata in maniera sommaria in Fig. 2.12.

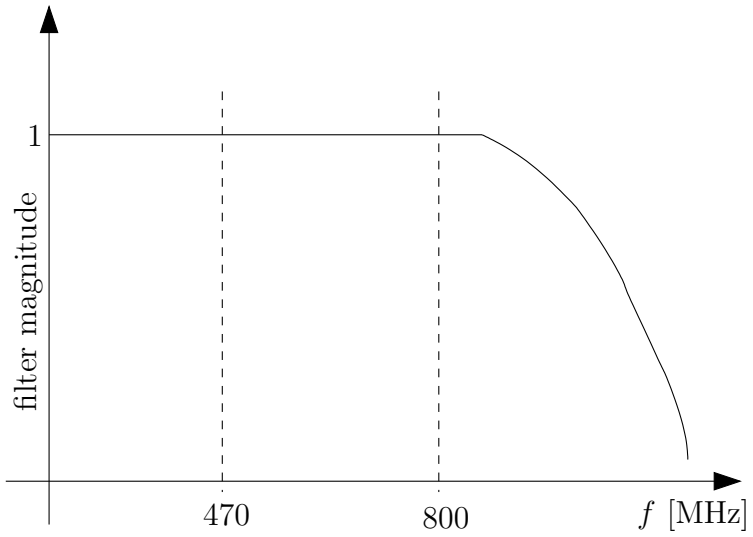


Figura 2.12: Risposta in frequenza del Filtro B.

Si tratta di un filtro passa basso molto semplice.

[4] Al più mettendo una predistorsione.

2.5 Realizzazione

La parte di implementazione è stata divisa in due grandi blocchi. Il primo blocco è chiamato BitMapping in cui i dati ricevuti vengono appositamente modificati codificandoli, mappandoli e raggruppandoli nel frame. Il secondo blocco è dedicato al filtraggio e interpolazione del segnale per renderlo adatto alle normative e per riuscire ad avere una buona trasmissione.

Il trasmettitore implementato avrà le seguenti caratteristiche:

- Accetta in ingresso sequenze da 10 bit (20 bit compressione 1/2) a 48Kbit/s.
- Aggiunge un bit di parità come CRC.
- Codifica i dati con modulazione 16APSK.
- Il frame è composto dal preambolo e da 11 segmenti di dati intervallati dai pilot. Ogni segmento dati è composto da 132 simboli(ref. fig. 2.4).

2.5.1 BitMapping

I dati che arrivano in ingresso vengono elaborati e codificati attraverso quattro blocchi prima di essere passati al filtro interpolatore. I blocchi sono i seguenti:

- Blocco *parity*
- Blocco *divide*
- Blocco $STBC$
- Blocco *piler*

Blocco *parity*

Il blocco *parity* ha il compito di aggiungere ai 10 bit d'ingresso un bit di parità dispari che sarà utile per il controllo del segnale in ricezione. Il bit viene generato tramite lo XOR di tutti i 10 bit. Viene poi aggiunto ai 10 bit ricevuti e passati al blocco successivo con un ritardo di un periodo di clock. Con il clock di $48KHz$ si ha un ritardo di $T_d = \frac{1}{48KHz} \simeq 21\mu s$.

Blocco *divide*

Questo blocco ha il compito di dividere gli 11 bit che arrivano in pacchettini di 4 bit per codificarli successivamente. Questo blocco riceve due clock diversi in ingresso uno per la sincronia dei bit in arrivo (48Khz) e uno per la sincronia dei bit in uscita (132KHz).

Il blocco riceve 4 sequenze da 11 bit (44 bit in totale) e le raggruppa su un vettore. Nel frattempo si inviano in uscita 4 bit per volta del vettore con frequenza $132KHz$. La sequenza è tale che per evitare la disallineamento della sequenza, che potrebbe portare ad un conflitto di lettura e scrittura dei bit,

ogni qual volta si completano le 4 sequenze da 11 bit in uscita si sono inviate 11 sequenze da 4bit, con un ritardo massimo di un ciclo di clock a $48KHz$. Per realizzare tale implementazione si sono usate due Macchine a Stati Finiti. La prima è composta da 4 stati ed è sincronizzata alla frequenza di arrivo dei campioni (48KHz). Ad ogni fronte memorizza nel vettore da 44bit gli 11 bit ricevuti in ingresso. La seconda macchina a stati finiti ha invece 11 stati ed è sincronizzata alla frequenza di 132KHz. Inoltre, l'avvio è ritardato rispetto alla prima Macchina a Stati Finiti di $21\mu s$[5] per evitare conflitti di lettura e scrittura degli stessi bit. Ad ogni stato invia in uscita 4 bit della vettore di memorizzazione temporanea. Il ritardo complessivo è quindi dato dal ritardo iniziale imposto paria a $21\mu s$.

Di seguito si mostra il diagramma temporale che eseguono le due macchine a stati e implementato dal file *divide.vhdl*.

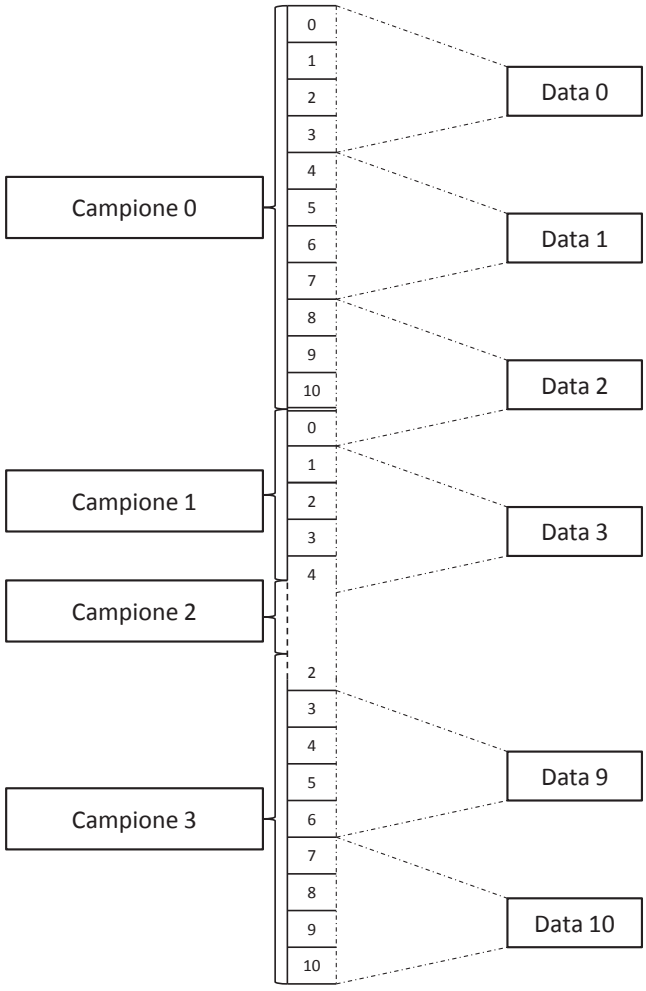


Figura 2.13: Diagramma temporale di *divide.vhdl*.

[5]Periodo un ciclo di clock a 48KHz

Blocco *STBC*

Il blocco STBC ha due compiti oltre a mappare i bit ricevuti nella costellazione rappresentata dalla codifica 16APSK, divide il flusso di dati per l'invio con due antenne.

I dati che riceve vengono codificati tramite la mappa descritta dalla costellazione in figura 2.14. Successivamente i dati vengono modificati in modo tale da rispettare la matrice di Alamouti 2x2:

$$S_A(\boldsymbol{\alpha}) = \frac{1}{\sqrt{2}} \begin{bmatrix} \alpha_1 & \alpha_2 \\ -\alpha_2^* & \alpha_1^* \end{bmatrix} \tag{2.19}$$

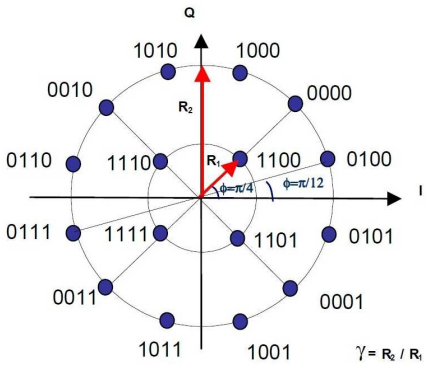


Figura 2.14: Costellazione codifica 16APSK.

Le operazioni di codifica sono effettuate grazie all'utilizzo di due ROM, in cui sono contenuti rispettivamente, i valori della parte reale e i valori della parte immaginaria del vettore.

Le ROM sono dello stesso tipo e hanno 16 allocazioni di memoria. Le sequenze da 4 bit, provenienti dal blocco *divide*, sono utilizzate come indirizzamento delle locazioni di memoria, attuando cosí sia la mappatura della costellazione che la codifica STBC.

Per l'implementazione come già detto in precedenza è stata usata la modulazione 16QPSK, non essendo di tipo differenziale, ha probabilità 1/4 di passare per lo zero. Il passaggio per lo zero genera un aumento delle non linearità dovute agli amplificatori.

Per ovviare a ció basta sostituire i valori contenuti nelle due ROM per la codifica 16APSK, con altri valori rappresentanti una altra codifica, di tipo differenziale o altro, che ne minimizza o elimini del tutto il problema. I dati nella ROM sono codificati con termini da 7 bit dato che risultano essere un valore ottimale per la rappresentazione della costellazione.

Blocco *piler*

L'ultimo blocco che compone la parte di codifica del segnale è implementato dal blocco *piler*. Questo blocco si occupa di ricevere i campioni mappati ed incapsularli nel frame, come descritto nel paragrafo 2.2.

L'implementazione è simile alla precedente del blocco *divide*. Il blocco è diviso in due parti; una parte dedicata all'aggiornamento, che ha il compito di

memorizzare su una RAM i dati che arrivano con frequenza di $132KHz$, e una macchina a stati finiti, creata in modo tale da generare il frame impilando in modo ciclico il preambolo, i pilot e i segmenti di campioni mappati.

La costruzione del frame deve avvenire in modo tale da evitare i conflitti di lettura e scrittura e ripetere il preambolo a distanza tale da non ritardare il flusso dati.

Per evitare tutto ció, si è reso compatibile l'invio del Frame con l'aggiornamento della RAM. Inoltre deve esserci un adattamento dalla frequenza di arrivo dei dati (132KHz), con quella di trasmissione del Frame (140KHz), quest'ultima imposta dall'analisi fatta precedentemente del filtro interpolatore. A questo proposito, l'aggiornamento della RAM deve avvenire nello stesso tempo dell'invio del Frame ($T_{FRAME} = T_{AGG}$).

Imponendo l'eguaglianza si puó trovare un risultato ottimale

$$T_{FRAME} = \frac{N \cdot data}{132KHz} = \frac{(Preambol + M \cdot pilot + N \cdot data)}{140KHz} = 11ms \quad (2.20)$$

con $N = 11$, $M = N + 1 = 12$ e $data = 132$ simboli.

Oltre ad avere la sincronizzazione dei due processi, evitando problematiche relative a ritardi o conflitti, si trasmette sempre un numero intero di campioni (528) evitando ulteriori problemi, relativi al mancato allineamento, nella fase di ricostruzione del campione dalle parole da 4 bit.

Si ha quindi che il sistema aggiorna la RAM con i campioni che riceve in ingresso e successivamente li estrae per e li impila nel Frame grazie alla macchina a stati finiti.

La macchina a stati finiti ha i seguenti stati:

1. Stato A: Invia il Preambolo contenuto in una ROM. (64 Symboli)
2. Stato B: Invia i Pilots. (2 Symboli)
3. Stato C: Invia i dati memorizzati nella RAM. (132 Symboli)

La macchina a stati parte dallo stato ″A″ e invia il Preambolo, poi continua a ripetere per 11 volte la sequenza ″B″, ″C″ inviando cosí i dati Contenuti nella RAM. Infine ripetendo un ulteriore volta l'invio del preambolo stato ″B″ completa il Frame. Una sequenza esemplificativa del funzionamento della Macchina a Stati Finiti è riportato in figura 2.15.

	FRAME									FRAME		
State	A	B	C	B	C		B	C	B	A	B	C
Send	Preambolo	Pilots	Data 0	Pilots	Data 1		Pilots	Data 10	Pilots	Preambolo	Pilots	Data 0

Figura 2.15: Esempio di funzionamento della Macchina a Stati Finiti utilizzata per l'invio.

Nelle figura seguenti si mostra lo schema a blocchi generato dallo Xilinx ISE del package chiamato *bitmapping.vhdl*.

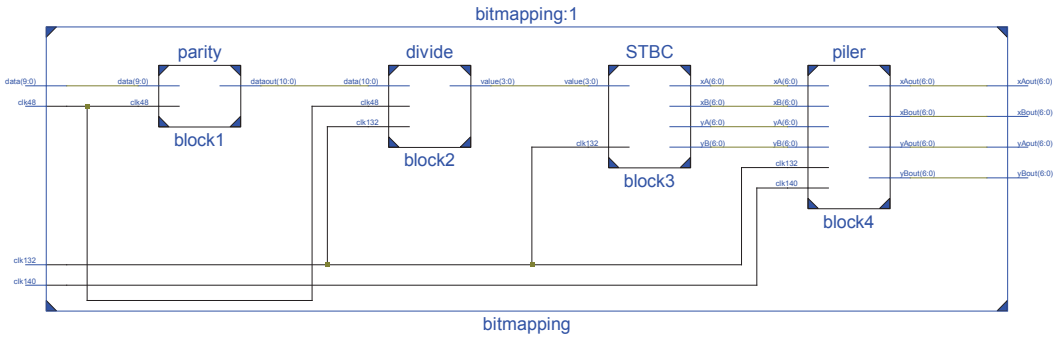


Figura 2.16: Schema a blocchi RTL del sistema di elaborazione del segnale.

bitmapping Project Status			
Project File:	Trasmetitore.xise	**Parser Errors:**	No Errors
Module Name:	bitmapping	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	182	301,440	1%	
Number of Slice LUTs	1,330	150,720	1%	
Number of occupied Slices	452	37,680	1%	
Number of LUT Flip Flop pairs used	1,355			
Number with an unused Flip Flop	1,214	1,355	30%	
Number with an unused LUT	25	1,355	2%	
Number of fully used LUT-FF pairs	116	1,355	66%	
Number of unique control sets	30			
Number of slice register sites lost to control set restrictions	22	301,440	1%	
Number of bonded IOBs	41	600	1%	
Number of DSP48E1s	0	768	0%	

Figura 2.17: *Design Summary* del sistema di elaborazione del segnale.

2.5.2 Filtro interpolatore

Il filtro interpolatore è il filtro di Nyquist presentato in precedenza (ref. par. 2.3)con la sola differenza che il salto di frequenza è inferiore. La frequenza finale del filtro non è di 21MHz ma di 2.1MHz dato che la si è ritenuta piče sufficiente per la trasmissione. Avendo una frequenza finale minore permette la riduzione della selettività del filtro complessivo. Il filtro è dunque composto dalla cascata di due filtri distinti: uno a 44 tappi e uno a 21 tappi.

Il primo filtro, che compone la cascata, ha 44 tappi ed è di Nyquist con modulo e andamento temporale riportato precedentemente in figura 2.9.

Il filtro viene preceduto da un interpolatore, che porta il segnale da 140KHz a 700KHz. L'interpolatore semplicemente aggiunge dei campioni nulli ai campioni base. Questo tipo di interpolazione crea delle repliche in frequenza che verranno successivamente filtrate dal filtro di Nyquist. Un secondo interpolatore, simile al precedente, è posto prima del secondo filtro per l'adattamento della frequenza da 700KHz a 2100KHz.

Il secondo Filtro, formato da 21 tappi, è un semplice filtro passa basso che è utile per eliminare le repliche del segnale che si creano tramite l'interpolazione.

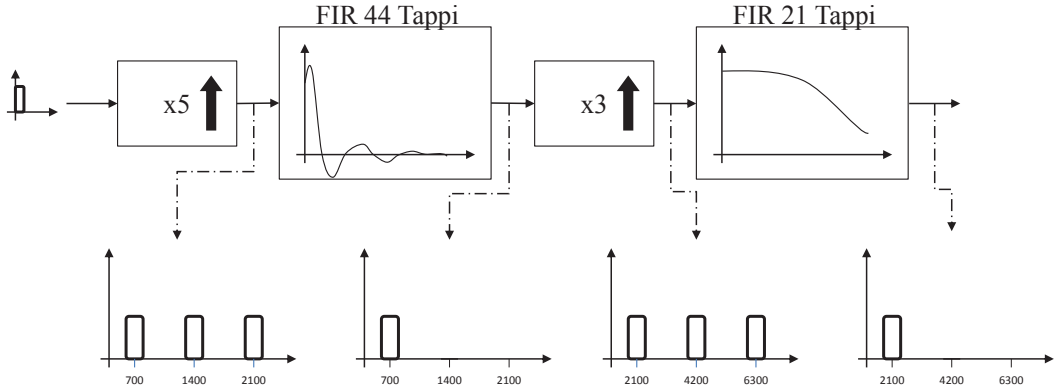


Figura 2.18: Diagramma esemplificativo dell'azione del filtro interpolatore.

Gli interpolatori sono formati da una Macchina a Stati Finiti. Nel primo interpolatore ($x5$) la Macchina a Stati Finiti è formata da due stati (″A″,″B″). Lo stato ″A″ invia in uscita il dato ricevuto in modo sincrono con l'arrivo del dato in l'ingresso. Lo stato ″B″ invia semplicemente un campione nullo all'uscita, ma ripetendosi per quattro volte attua l'interpolazione del segnale.

Il secondo interpolatore ($x3$) ha invece una Macchina a Stati Finiti formata da tre stati(″A″,″B″,″C″). Il primo stato (″A″) manda in uscita i dati che arrivano in ingresso, mentre gli stati ″B″ e ″C″ in modo analogo all'interpolatore precedente, attuano l'interpolazione inviando due valori nulli prima del campione successivo.

Entrambi i filtri sono di tipo FIR con rispettivamente 44 e 21 coefficienti. Il primo filtro è sincronizzato con una frequenza di 700KHz (5x140KHz) il secondo filtro invece ad una frequenza di 2.1MHz (3x700KHz).

Questi possono essere realizzati in due modi distinti: con l'implementazione manuale che si interpreta come la somma dei coefficienti moltiplicati per il termine ritardato, oppure, tramite l'utilizzo dei Core IP.

Un filtro FIR è un tipo di filtro di elaborazione del segnale la cui risposta impulsiva è di durata finita, perchè tende a zero in un tempo finito. Questo è in contrasto con i filtri a risposta infinita IIR, che hanno un feedback interno e possono continuare a rispondere per tempi indeterminati di solito decadi. La risposta impulsiva di un filtro FIR del N-esimo ordine ha normalmente N+1 coefficienti mentre i successivi sono tutti nulli. Il filtro FIR è una convoluzione discreta del sgnale d'ingresso x con una serie di coefficienti b. Per un filtro FIR , l'output è una somma ponderata del termine corrente e un numero finito di valori precedenti. Matematicamente il filtro puó essere definito come

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + \cdots + b_N \cdot x[n-N]$$

dove $x[n]$ è il segnale d'ingresso, $y[n]$ è il segnale d'uscita, b_i sono i coefficienti del filtro e N è l'ordine del filtro.

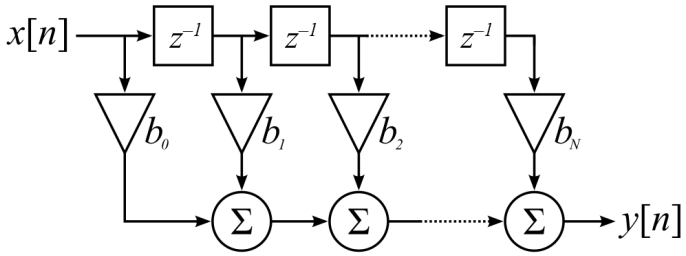


Figura 2.19: Diagramma di flusso di un filtro FIR a tempo discreto.

Il diagramma di flusso del segnale riportato in figura 2.19 mostra l'algoritmo e suggerisce un'architettura composta da blocchi di ritardo 'Z' e da moltiplicatori. I Moltiplicatori mostrati nel diagramma di flusso corrispondono a una serie moltiplicatori paralleli con uno degli ingressi fissato dai coefficienti del filtro. Per ogni tappo del filtro sono utilizzati un coefficiente, un elemento di ritardo e un moltiplicatore. Tutti vengono successivamente sommati con una serie di somme.

I Core IP sono delle strutture create da Xilinx che generano un'implementazione ottimale di una determinata funzione, nel nostro caso il filtro FIR.

Nella fattispecie, sapendo che il filtro in trasmissione deve essere realizzato in maniera ottimale per rientrare nelle specifiche ETSI, si prevede immediatamente che ci sarà un utilizzo maggiore di LUTs nell'implementazione manuale rispetto all'implementazione ottimale generata dai Core IP. Questo è dovuto dai bit dedicati ad esprimere i coefficienti dei filtri. In particolare per esprimere i coefficienti del filtro, ed avere una interpretazione corretta, devono essere espressi con almeno 13 bit. Se si conta che all'ingresso arrivano dei vettori da 7 bit i moltiplicatori utilizzati sono 13x7 e i sommatori devono essere da 20 bit. Essendoci relativamente pochi tappi si ha che il consumo non è enorme, ma l'implementazione dedicata a Core IP sicuramente ottimizza in maniera adeguata questa architettura. I coefficienti dei filtri sono riportati in appendice A. All'uscita del primo filtro si ha una riduzione della definizione del vettore

portandolo da 20 a 7 bit. Si ha una riduzione da 20 a 14 bit dopo il secondo filtro. L'uscita con definizione a 14 bit è determinata dall'utilizzo del DAC AD9707 che opera a frequenza massima di 175MHz convertendo segnali con precisione 14bit. In figura 2.20 sono riportati i moduli dei filtri implementati.

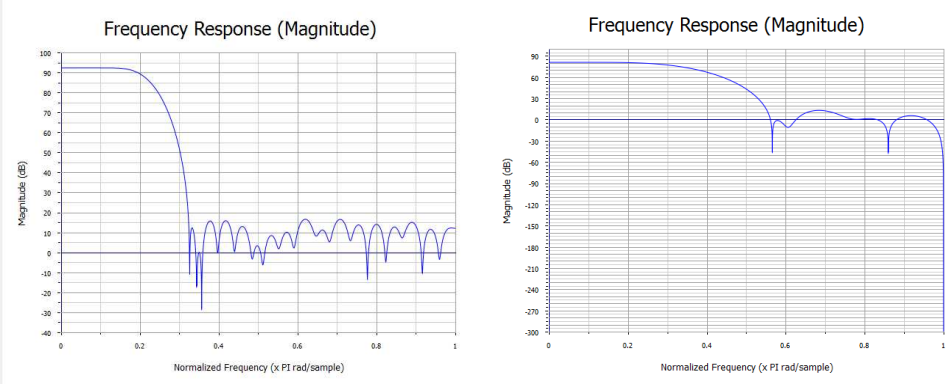


Figura 2.20: Moduli della risposta in frequenza dei due filtri di trasmissione.

Nei *Design Summary* delle due implementazioni riportati sotto, si vede come l'utilizzo dei Core IP provochi un utilizzo maggiore, ma ottimale, del DSP integrato nella Virtex 6 a vantaggio di una riduzione delle slice LUTs utilizzate.

TOP Project Status			
Project File:	Trasmetitore.xise	**Parser Errors:**	No Errors
Module Name:	top	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	5,539	301,440	1%	
Number of Slice LUTs	11,899	150,720	7%	
Number of occupied Slices	3,464	37,680	9%	
Number of LUT Flip Flop pairs used	12,388			
Number with an unused Flip Flop	7,174	12,388	57%	
Number with an unused LUT	489	12,388	3%	
Number of fully used LUT-FF pairs	4,725	12,388	38%	
Number of unique control sets	55			
Number of slice register sites lost to control set restrictions	125	301,440	1%	
Number of bonded IOBs	69	600	1%	
Number of DSP48E1s	0	768	0%	

Figura 2.21: *Design Summary* del sistema con Filtro ad implementazione manuale.

TOP Project Status			
Project File:	Trasmetitore.xise	**Parser Errors:**	No Errors
Module Name:	TrasmitterIP	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	1,751	301,440	1%	
Number of Slice LUTs	2,195	150,720	1%	
Number of occupied Slices	935	37,680	2%	
Number of LUT Flip Flop pairs used	2,540			
Number with an unused Flip Flop	1,271	2,540	50%	
Number with an unused LUT	345	2,540	13%	
Number of fully used LUT-FF pairs	924	2,540	36%	
Number of unique control sets	115			
Number of slice register sites lost to control set restrictions	217	301,440	1%	
Number of bonded IOBs	69	600	1%	
Number of DSP48E1s	64	768	0%	

Figura 2.22: *Design Summary* del sistema con Filtro CORE IP.

Il privilegiare una o l'altra implementazione viene determinato dall'FPGA che si vuole utilizzare. Ad esempio se si sostituisce la Virtex 6, da noi utilizzata, con una Spartan 3E $S1200E - 4FTG256C$, ottimale per un implementazione del trasmettitore a basso costo, si privilegia l'implementazione senza Core IP. La scelta è vincolata dalla bassa presenza di moduli DSP, in questo caso Moltiplicatori 18x18, che non ne permette l'utilizzo.

TransmitterIP Project Status	
Project File:	Trasmetitore.xise
Module Name:	TransmitterIP
Target Device:	xc3s1200e-4ft256
Product Version:	ISE 13.2
Design Goal:	Balanced
Design Strategy:	Xilinx Default (unlocked)
Environment:	System Settings

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3299	8672	38%
Number of Slice Flip Flops	4303	17344	24%
Number of 4 input LUTs	9005	17344	51%
Number of bonded IOBs	69	190	36%
Number of BRAMs	5	28	17%
Number of MULT18X18SIOs	52	28	185%
Number of GCLKs	6	24	25%

Figura 2.23: *Design Summary* del sistema con Filtro CORE IP su Spartan3.

Clock Generator

Per la sincronia dei vari blocchi e del filtro interpolatore è stato creato un *ClockGenerator*. Partendo dalla frequenza iniziale di 8.4MHz[6] si generano

[6]Frequenze ritenuta ideale per le operazioni di trasmissione

le frequenze per la sincronia utilizzate nei vari blocchi ossia 48KHz, 132KHz, 140KHz, 700KHz e 2.1MHz. Tutte le frequenze sono multiple della frequenza madre 8.4MHz, ad eccezione della frequenza di 132KHz.

Le frequenze multiple più vicine a 132KHz sono 131.25Hz e 133.33KHz, le quali vengono generate contando rispettivamente 64 e 63 cicli di clock. Quindi per la creazione della frequenza di 132KHz si utilizza una Macchina a Stati Finiti che conta in successione 63 e 64 cicli di clock per generare, in media, una frequenza di 132Khz. La macchina ha 11 stati e ripete il conteggio per le due frequenze nel modo seguente:

A B A A B A A B A A B

con A relativo alla frequenza di 131.25Hz (64 cicli) e B relativo alla frequenza di 133.33KHz (63 cicli).

Si vede dall'equazione seguente come questa particolare combinazione porti alla generazione in media del periodo di clock di 132KHz voluto.

$$T_{132KHz} = \frac{7 \cdot 64 + 4 \cdot 63}{8.4MHz} = 7.576\mu s \tag{2.21}$$

Il cambio di frequenza permettere la sincronia perfetta del blocco *piler*, evitando così conflitti di lettura e scrittura contemporanea.

In uscita al trasmettitore si avranno due clock uno per la sincronia con le parole in uscita dal filtro e in più il clock a 48KHz che verrà utilizzato per sincronizzare l'ADC che invia i campioni in ingresso al trasmettitore.

clkGen Project Status			
Project File:	Trasmetitore.xise	**Parser Errors:**	No Errors
Module Name:	clkGen	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	53	301,440	1%	
Number of Slice LUTs	67	150,720	1%	
Number of occupied Slices	22	37,680	1%	
Number of LUT Flip Flop pairs used	69			
Number with an unused Flip Flop	21	69	30%	
Number with an unused LUT	2	69	2%	
Number of fully used LUT-FF pairs	46	69	66%	
Number of unique control sets	2			
Number of slice register sites lost to control set restrictions	11	301,440	1%	
Number of bonded IOBs	6	600	1%	
Number of DSP48E1s	0	768	0%	

Figura 2.24: *Design Summary* di *ClockGenerator*.

Infine ecco il *Design Summary* e lo schema RTL del trasmettitore completo senza l'utilizzo dei Core IP. Per i risultati operativi si rimanda all'appendice C.

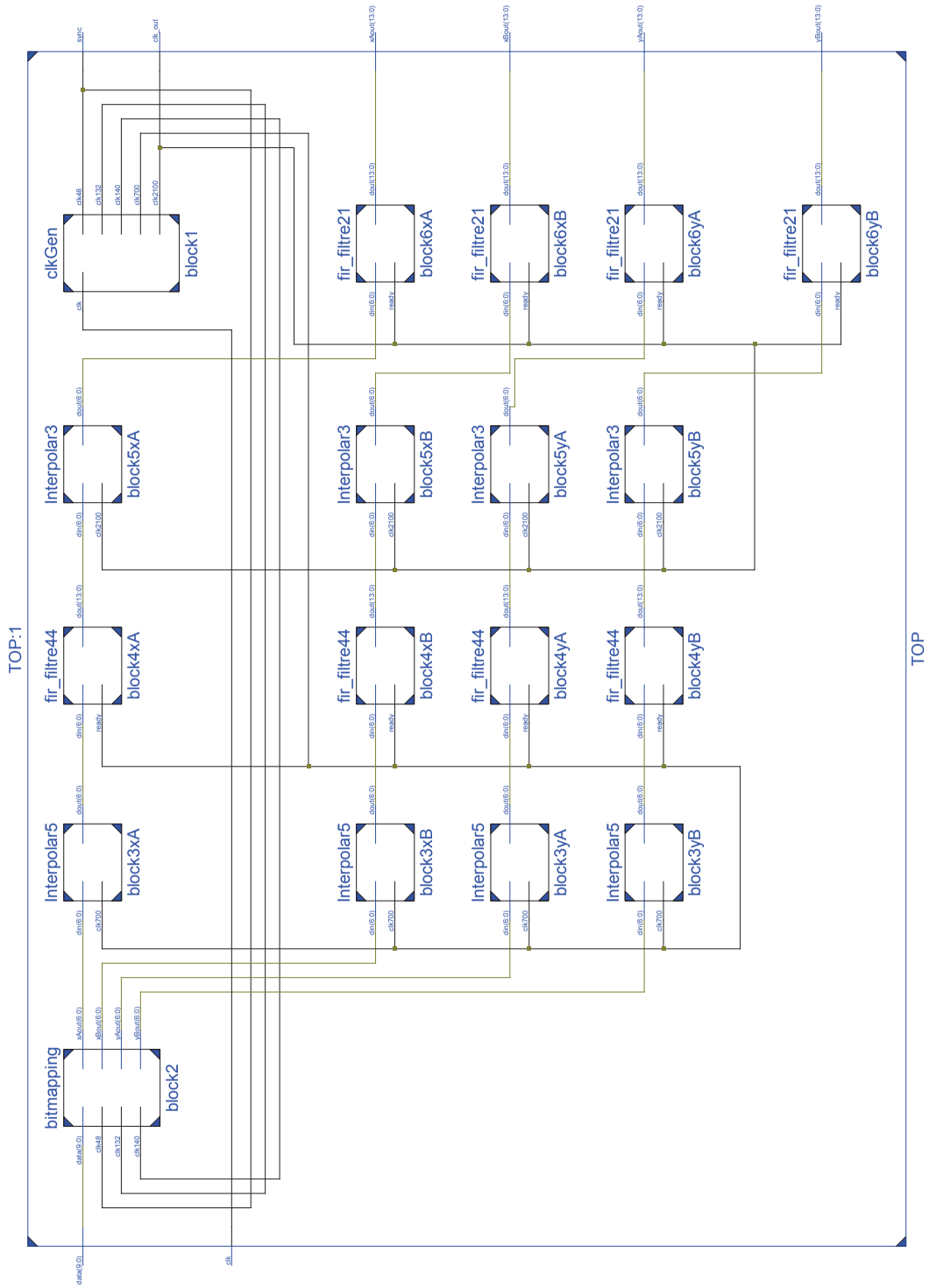


Figura 2.25: Schema a blocchi RTL del Trasmettitore.

TOP Project Status			
Project File:	Trasmetitore.xise	**Parser Errors:**	No Errors
Module Name:	top	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	5,539	301,440	1%	
Number of Slice LUTs	11,899	150,720	7%	
Number of occupied Slices	3,464	37,680	9%	
Number of LUT Flip Flop pairs used	12,388			
Number with an unused Flip Flop	7,174	12,388	57%	
Number with an unused LUT	489	12,388	3%	
Number of fully used LUT-FF pairs	4,725	12,388	38%	
Number of unique control sets	55			
Number of slice register sites lost to control set restrictions	125	301,440	1%	
Number of bonded IOBs	69	600	1%	
Number of DSP48E1s	0	768	0%	

Figura 2.26: *Design Summary* del Trasmettitore.

Capitolo 3

Il Ricevitore

In questa sezione, al contrario della precedente, è strutturata in modo tale che per ogni blocco prima si presenta una piccola trattazione teorica in cui ne viene spiegato il funzionamento e successivamente ne segue l'implementazione sull'FPGA.

L'intero circuito di ricezione si basa su uno schematico retroazionato, questo è composto in modo tale da permettere un incertezza maggiore sulle frequenze e sui tempi del circuito, che verranno successivamente corretti per far funzionare in modo adeguato l'intero sistema ricevente. Di seguito (fig.3.1) si illustra lo schema di principio del ricevitore che successivamente verrà diviso in due parti: *"Ricezione"*, *"Sincronizzazione, Stima e Decodifica"*.

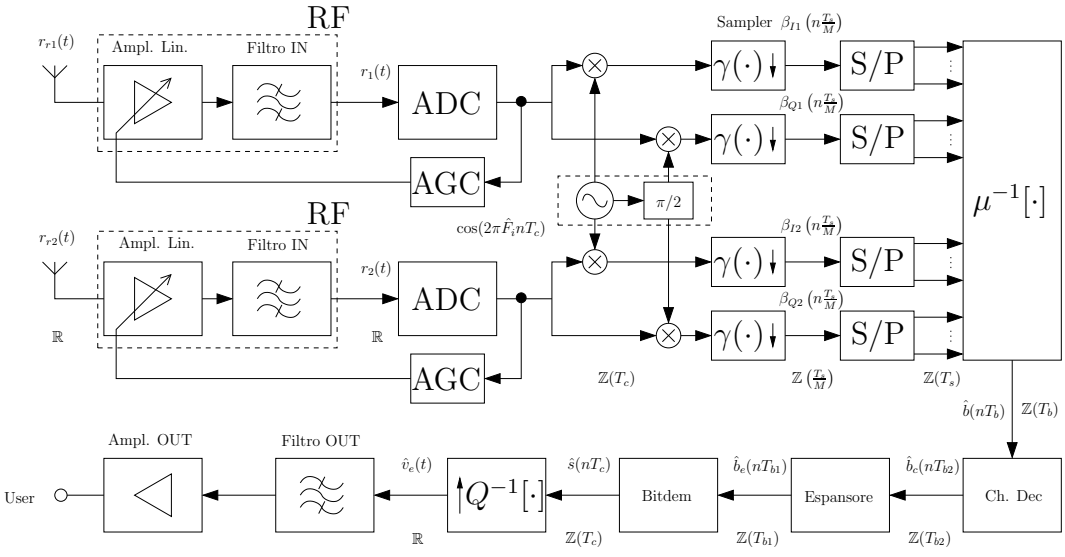


Figura 3.1: Schema di principio del ricevitore.

Nella fase di *"Ricezione"* [par. 3.1] si mostreranno le parti principali del circuito a radio frequenza e di come verrà utilizzata la parte digitale per l'intermodulazione e per regolare l'amplificazione del segnale analogico nel dominio reale.

Nella parte di *"Sincronizzazione, Stima e Decodifica"* [par. 3.2] si mostrerà il vero cuore del ricevitore in cui si avrà la sincronizzazione del frame in

ricezione e si analizzeranno i massimi locali per riuscire ad agganciare la frequenza corretta, e si avrà anche una continua modifica della stima di canale in modo da permettere una ricezione più precisa. Infine si tratterà la decodifica dei bit ricevuti in modo da riuscire ad estrarre correttamente i dati trasmessi.

3.1 Ricezione

L'intero circuito di ricezione è anch'esso di tipo retroazionato in modo tale da comandare i guadagni dei componenti che compongono la sezione RF.

3.1.1 SezioneRF

Ogni antenna ricevente avrà una catena di ricezione schematizzata nello schema a blocchi di Fig. 3.2. La catena ha lo scopo di asservire dinamicamente il segnale RF fino all'ingresso del convertitore A/D che successivamente verrà collegato alla Virtex 6.

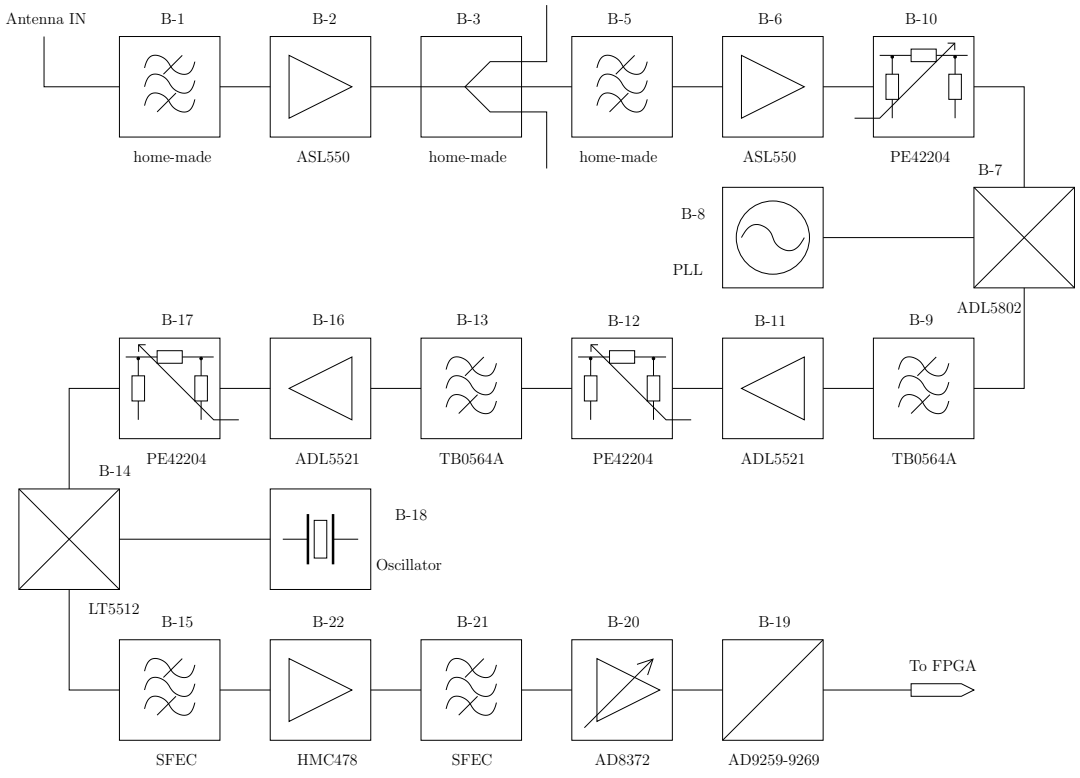


Figura 3.2: Schema a blocchi della parte RF del ricevitore.

3.1.2 Specifiche dei Dispositivi Analogici

In questo paragrafo andiamo a visualizzare nel dettaglio tutte le specifiche d'interesse dei dispositivi che compongono la catena della parte RF del ricevitore illustrata in Fig. 3.2.

Filtri B-1 e B-5 e Splitter B-3

Si tratta di circuiteria realizzata all'interno di Wisycom.

Il filtro B-1 è un filtro a banda larga (470 ÷ 870 MHz) che comporta un'insertion loss di 1 dB.

Lo splitter B-3 e' un dispositivo che serve a mandare la replica del segnale ad un secondo ramo del ricevitore ed un terzo ramo per collegare in cascata un secondo ricevitore. Sul segnale esso porta un'attenuazione di 8 dB.

Il filtro B-5 invece è un filtro a banda stretta (30 MHz) che presenta una perdita di inserzione di 6 dB ed è centrato su portante configurabile in tensione tra 470 e 830 MHz per selezionare il canale audio che andiamo a demodulare.

Amplificatore ASL550

È un amplificatore posto in prefiltraggio (B-2) e postfiltraggio (B-6) in selezione al canale. Ha la proprietà di godere di ottima linearità su una banda estremamente ampia (50 MHz ÷ 1 GHz), molto importante in questo punto della catena dove il segnale utile può essere collocato in un'estensione spettrale larga (vedi specifiche del sistema 1.1).

Le caratteristiche principali[1] del dispositivo sono:

- guadagno G = 16 dB (nominale da 17.1 a 16.9);
- cifra di rumore NF = 2.3 dB (nominale 2);
- intercetta al 3^{zo} ordine OIP3 = 40 dBm (nominale 43);
- compressione P1dB = 22 dBm (nominale 25).

Attenuatore PE43204

È un dispositivo posto in vari stadi della catena che consente dei livelli di attenuazione regolabili. Servono per meglio adattare il livello del segnale all'interno del flusso RF in funzione della potenza del segnale d'ingresso ricevuto. I suoi principali parametri si riassumono in:

- insertion loss IL = 1 dB (0.6 ÷ 0.7);
- attenuazione minima A_{min} = 0 dB;
- attenuazione massima A_{max} = 18 dB;
- step S = 6 dB;
- (IIP3 = 61 dBm);
- (P1dB = 28 ÷ 30 dBm);
- switching delay = 26 ns.

[1] I valori riportati fanno riferimento ai parametri indicati da Enzo Frigo mentre tra parentesi vengono indicati i valori riportati sui *data sheets* dei dispositivi in questione.

Mixer ADL5802

Questo mixer è collegato ad un PLL sintonizzabile e serve per portare il segnale ad una frequenza intermedia di 400 MHz. Le caratteristiche principali all'uscita di questo dispositivo sono:

- G = 1 dB (1.6);
- NF = 11 dB;
- OIP3 = 27 dBm (IIP3 nominale = 28 dm);
- P1dB = 13 dBm.

Filtri a Banda Stretta TB0564A e Amplificatori ADL5521

Prima di passare alle basse frequenze, il segnale subisce due stadi di filtraggio, amplificazione e attenuazione variabile.

Il filtraggio serve per selezionare la banda di interesse su cui si trova il segnale utile ed eliminare la maggior parte del rumore fuori banda. Le caratteristiche dei filtri sono:

- I.L. = 6 dB;
- F_c = 400 MHz;
- BW = 250 kHz.

L'amplificazione avviene a guadagno costante mediante amplificatori a bassa rumorosità che funzionano bene alle frequenze medio/alte. Le sue specifiche sono:

- G = 22 dB (20.8 dB a 900 MHz);
- NF = 1.3 dB (0.8 dB a 900 MHz);
- OIP3 = 34 dBm (37 dBm a 900 MHz);
- P1dB = 18 dBm (21.8 dBm a 900 MHz).

Mixer LT5512

Il secondo mixer è collegato ad un risonatore che produce un tono a 389.3 MHz con l'obiettivo di riportare il segnale audio alla frequenza di 10.7 MHz. Le caratteristiche principali all'uscita di questo dispositivo sono:

- G = 1 dB;
- NF = 11 dB;
- OIP3 = 20 dBm;
- P1dB = 10 dBm (10.5 a 10 $\div$ 500 MHz).

Filtri a Bassa Frequenza SFEC e Amplificatore HMC478

Nella parte finale della catena, quando il segnale è riportato alla frequenza di 10.7 MHz, avviene un'operazione di filtraggio accurata, operata mediante filtri ceramici che operano per l'appunto attorno a quella frequenza. Qui utilizziamo filtri aventi le seguenti specifiche:

- I.L. = 6 dB;
- F_c = 10.7 MHz;
- BW = 230 ÷ 330 kHz (dipende dal modello).

L'amplificatore utilizzato in questo stadio invece è caratterizzato da:

- G = 23 dB (24);
- NF = 2 dB (2.5);
- OIP3 = 34 dBm (31);
- P1dB = 18 dBm (16).

Amplificatore a Guadagno Variabile AD8372

Localizzato all'ultimo stadio prima dell'ingresso alla conversione analogico–digitale, questo dispositivo, dotato di ampia dinamica, permette di aggiustare la potenza del segnale in ingresso al convertitore A/D.

- G = -6 ÷ 32 dB (-9 ÷ 32);
- S = 1 dB;
- NF = 8 dB (7.8 a 32 dB);
- OIP3 = 34 dBm (32 a 32 dB);
- P1dB = 18 dBm (18.1 a 32 dB);
- switching delay = 20 ns (per un gain–step di 6 dB)

ADC AD9259

AD9259 è un dispositivo dotato di quattro linee di conversione, rendendo necessario un solo chip per ricevitore a due canali, è dotato di conversione seriale a 14 bit/sample.

3.1.3 Strategia Operativa

Per giungere ad una strategia operativa, riportiamo tutti i dati utili al link–budget su una tabella riassuntiva, vedi Tab. 3.1.

	G_{max} dB	G_{min} dB	G^T_{max} dB	G^T_{min} dB	NF dB	NF^T_{max} dB	NF^T_{min} dB	OIP^3 dBm	P_{1dB} dBm
ANT	0	0	0	0	0.0	0.00	0.00		
B-1	-1	-1	-1	-1	1.0	0.00	0.00		
B-2	16	16	15	15	2.3	2.74	2.74	40	22
B-3	-8	-8	7	7	8.0	3.11	3.11		
B-5	-6	-6	1	1	6.0	4.22	4.22		
B-6	16	16	17	17	2.3	5.05	5.05	40	22
B-10	**-1**	**-19**	16	-2	1.0	5.05	5.05		
B-7	1	1	17	-1	11.0	13.34	5.43	27	13
B-9	-6	-6	11	-7	6.0	14.04	5.50		
B-11	22	22	33	15	1.3	14.33	5.54	34	18
B-12	**-1**	**-19**	32	-4	1.0	14.33	5.54		
B-13	-6	-6	26	-10	6.0	15.39	5.54		
B-16	22	22	48	12	1.3	15.80	5.54	34	18
B-17	**-1**	**-19**	47	-7	1.0	15.81	5.54		
B-14	1	1	48	-6	11.0	19.83	5.54	20	10
B-15	-6	-6	42	-12	6.0	20.34	5.54		
B-22	23	23	65	11	2.0	20.69	5.54	34	18
B-21	-6	-6	59	5	6.0	20.70	5.54		
B-20	**32**	**-6**	91	-1	8.0	20.76	5.54	34	18
B-19	D	D	D	D	D	D	D	D	D

Tabella 3.1: Riassunto dei parametri principali per la catena RF in ricezione.

Le prime due colonne si riferiscono a guadagno massimo e guadagno minimo portato da ciascun dispositivo. Naturalmente, solo dispositivi a guadagno/attenuazione variabile hanno questi due valori distinti.[2] La terza e la quarta colonna riporta invece il guadagno cumulativo di catena massimo e minimo.

In quinta colonna segue il valore della cifra di rumore di ciascun dispositivo mentre nelle due seguenti abbiamo la cifra di rumore cumulativa, calcolata secondo la formula di Friis, vedi [10], dell'analisi in cascata della cifra di rumore

$$\mathrm{nf} = \mathrm{nf}_1 + \sum_{i=2}^{n} \frac{\mathrm{nf}_i - 1}{\prod_{j=1}^{i-1} \mathrm{g}_j} = \mathrm{nf}_1 + \frac{\mathrm{nf}_2 - 1}{\mathrm{g}_1} + \frac{\mathrm{nf}_3 - 1}{\mathrm{g}_1 \mathrm{g}_2} + \cdots + \frac{\mathrm{nf}_n - 1}{\mathrm{g}_1 \cdots \mathrm{g}_n} \qquad (3.1)$$

dove nf e g sono i parametri di cifra di rumore e guadagno dei dispositivi in cascata espressi in grandezza lineare. La formula evidenzia come, qualora il guadagno complessivo della catena sia crescente col numero dei dispositivi, la figura di rumore dei dispostivi a monte della catena impatta maggiormente sulla figura di rumore complessiva. Chiaramente, il valore della cifra di rumore

[2] Tali dispositivi sono evidenziati in *bold face*.

totale massimo, fa riferimento alla catena con i valori di guadagno minimi e viceversa.

Le ultime due colonne evidenziano i parametri relativi alle non–linearità dei dispositivi interessati (amplificatori e mixer).

Controllo del Guadagno

Lo schema di ricezione RF del segnale esaminato consente che si possa operare un controllo sul guadagno dinamico allo scopo di mantenere pressochè costante il livello di potenza del segnale che entra nell'ADC. Il controllo può essere effettuato variando il guadagno dei tre attenuatori e dell'amplificatore posto all'ultimo stadio prima del convertitore. In Tab. 3.2 si illustra lo schema che permette di operare all'interno dei vincoli dovuti a livelli di saturazione imposti dai dispositivi della catena.

I_{min} dBm	I_{max} dBm	G_1 dB	G_2 dB	G_3 dB	G_A dB	O_{min} dBm	O_{max} dBm	NF dB	SNR dB	SNR dB
-104	-84	0	0	0	32	-13	7	5.54	10.46	30.46
-84	-66	0	0	0	32÷14	6	7	5.54	30.46	48.46
-66	-48	0	0	-18	32÷14	6	7	5.57	48.43	66.43
-48	-30	0	-18	-18	32÷14	6	7	7.02	64.98	82.98
-30	2	-18	-18	-18	32÷0	6	7	20.76	69.24	101.24

Tabella 3.2: Strategia di *gain settings*.

Le prime due colonne indicano i range di operatività, nelle quattro successive la rispettiva impostazione dei valori di guadagno per i dispositivi regolabili, sesta e settima colonna mostrano il range di potenza che ci si attende all'ingresso dell'ADC, la colonna successiva mostra il valore di cifra di rumore complessiva equivalente che caratterizza la catena, mentre le ultime due colonne danno un'indicazione del rapporto segnale rumore (SNR) equivalente[3] all'ingresso del ricevitore. Il rumore termico è valutato essere $P_W = -120$ dBm.

Questa strategia è orientata ad agire maggiormente sull'amplificatore all'ultimo stadio che consente step di 1 dB mantenendo la potenza del segnale d'ingresso all'ADC pressochè costante per una dinamica di 86 dB (-84÷2 dBm al segnale d'ingresso all'antenna in ricezione) ed inoltre presenta uno switch–time inferiore rispetto agli attenuatori. Questi ultimi vengono utilizzati solamente per i due valori estremi di attenuazione (0 e -18 dB) diventando a tutti gli effetti degli attenuatori binari (On/Off).

Al di fuori del range dinamico la catena può funzionare bene a potenze inferiori di -84 dBm fino a -104 dBm al di sotto della quale l'SNR risulta minore di 10 dB rendendo poco affidabile il segnale ricevuto. Per potenze di ingresso superiori ai 2 dBm, tra l'altro estremamente poco probabili, alcuni stadi della catena cominciano a lavorare in zona di saturazione.

[3]che tiene conto della cifra di rumore.

B_1	B_2	C_1^1	C_2^1	C_1^2	C_2^2	C_1^3	C_2^3
L	L	L	L	L	L	L	L
L	H	L	L	L	L	H	H
H	L	L	L	H	H	H	H
H	H	H	H	H	H	H	H

Tabella 3.3: Tabella della verità per pilotare gli attenuatori.

Occorreranno dunque 6 bit per pilotare il guadagno del amplificatore B-20 mentre sarà sufficiente un circuito logico con ingresso a 2 bit per pilotare gli attenuatori come illustrato in Tab. 3.3 e Fig. 3.3

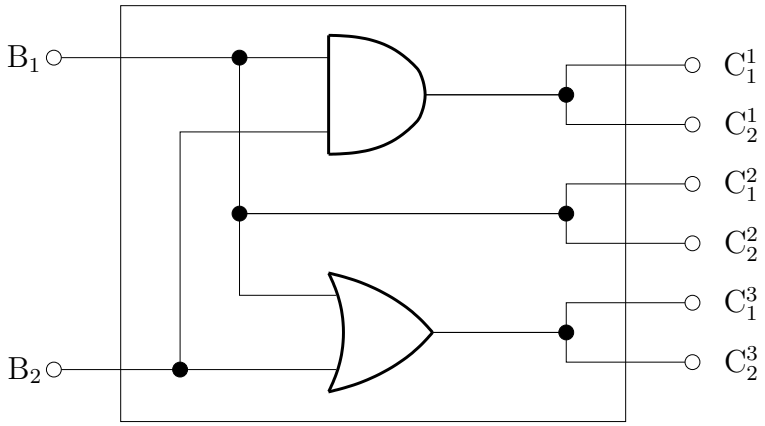


Figura 3.3: Circuito logico che implementa la tabella della verità di Tab 3.3.

Logica di Controllo del Guadagno

Di seguito è riportatolo schema del Controllo Automatico del Guadagno fig.3.4 al ricevitore in cui si hanno tre blocchi principali:

- $\mathcal{B}^n/\mathcal{A}$: blocco che porta i dati da seriali a paralleli
- ENV : Blocco che valuta l'inviluppo del segnale
- AGC : Blocco che regola gli attenuatori e l'amplificatore.

La strategia utilizzata è quella di aggiornare il guadagno della catena, non campione per campione, bensì ogni millisecondo in accordo col periodo di campionamento del canale nella sua misura nonchè il periodo di pilot in cui poi il canale sarà effettivamente stimato. Quindi il blocco ENV cerca il massimo in un intervallo $T_p = 1$ ms del segnale ricevuto, tenendo presente del ritardo di gruppo, secondo la formula

$$\mathrm{ENV}_m = \max_{t \in \left[\frac{d\phi}{d\omega}, T_p\right]} |s(t + mT_p)| \tag{3.2}$$

dove ENV_m rappresenta il valore di inviluppo riferito al m–esimo tronco di 1 ms di segnale. Il valore dell'inviluppo sarà utilizzato dal blocco AGC, che in

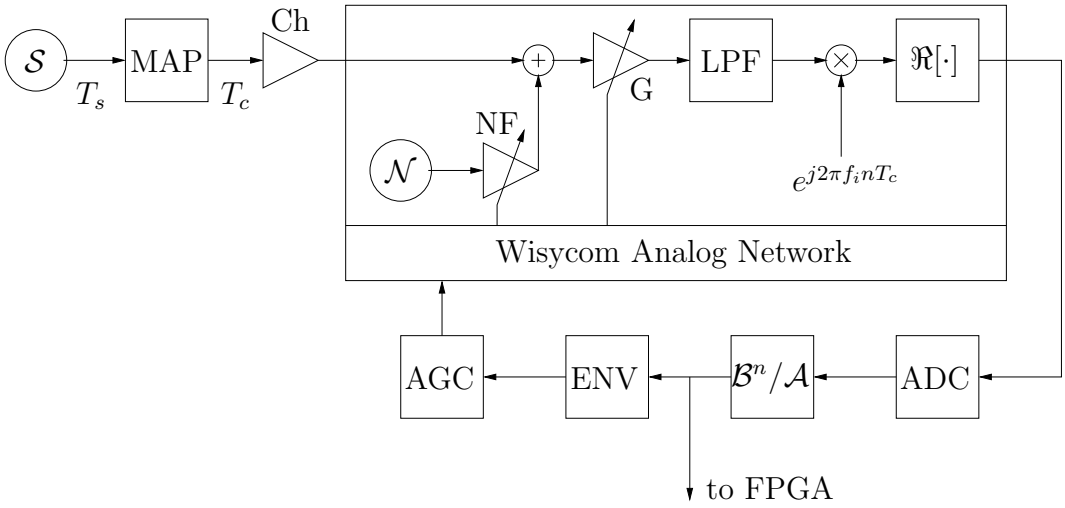


Figura 3.4: Schema di Controllo Automatico del Guadagno (AGC) al ricevitore che tiene conto del filtraggio sul segnale.

questo caso verrà aggiornato ad ogni T_p il cui obiettivo sarà esclusivamente quello di seguire la variabilità del canale più che quella del segnale utile stesso.

Il blocco AGC è caratterizzato da una fase di start up durante la quale, in pochi passi, si sintonizza sul guadagno più adeguato. Successivamente l'aggiustamento del guadagno è differenziato in 5 possibili esiti illustrato in Fig. 3.5, che permettono di abbassare il guadagno se il segnale è troppo forte e aumentarlo se il segnale è troppo debole.

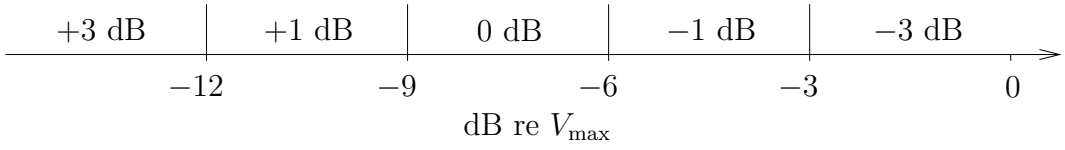


Figura 3.5: Regioni di regolazioni del guadagno impostate dall'AGC.

3.1.4 Implementazione del AGC

Seriale/Parallelo

Dal ADC escono i bit in modo seriale e hanno due linee di clock che permettono la sincronizzazione. Il Clock dati (DCO) per riuscire a catturare all'istante esatto i valori in uscita e il Clock del Frame (FCO) per segnalare che in uscita si ha una nuova parola. Di seguito in Fig. 3.6 viene riportato il diagramma temporale del convertitore.

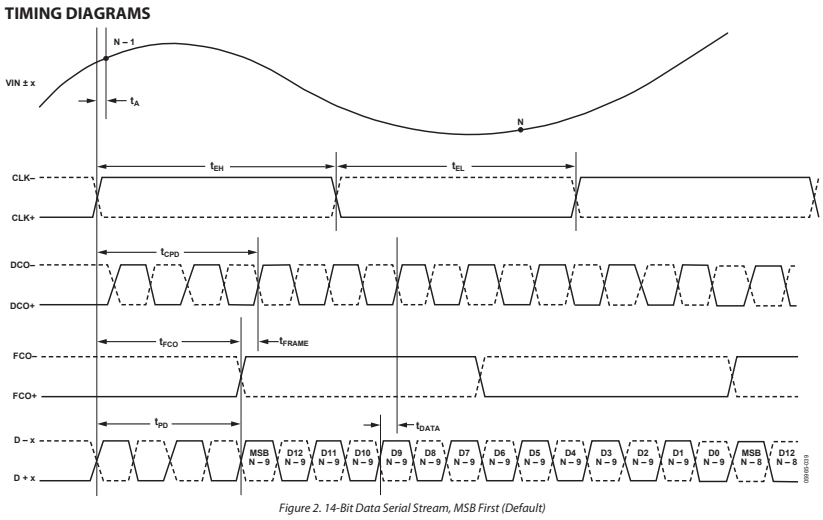


Figura 3.6: Timing Diagrams del ADC AD9259.

L'idea per la conversione seriale parallelo è molto semplice: sincronizzarsi sui fronti di salita della linea $DCO+$ per i bit pari e sui fronti di salita del $DCO-$ per i bit dispari. Infine abilitare l'uscita ad ogni fronte di salita del $FCO+$, e comporre l'intera parola da 14 bit.

Di seguito si presenta lo schema a blocchi ideale del sistema,e il design summary dell'implementazione.

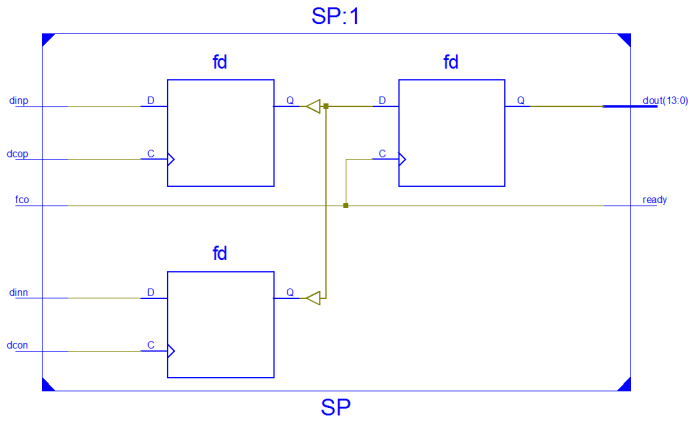
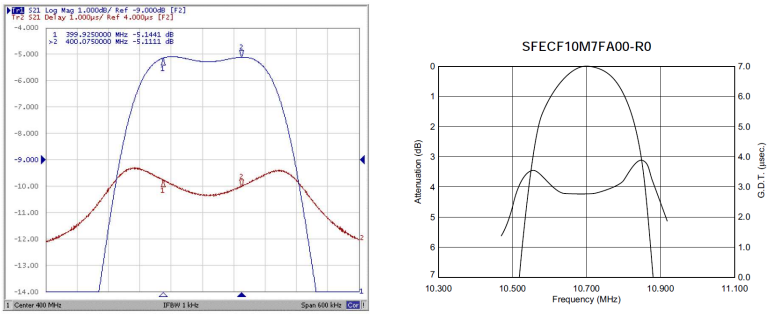


Figura 3.7: Schema a blocchi generato da XilinX ISE del blocco SP.

Blocco ENV

Come già riportato in precedenza il blocco ENV segue i massimi locali e invia il massimo al blocco che si occupa di regolare il guadagno. Inoltre il blocco ENV tiene conto del transitorio presente dovuto dalla risposta dei filtri alle variazioni repentine del guadagno degli attenuatori.

Il fattore che indica il ritardo complessivo è dato dal ritardo di gruppo di ogni filtro. I filtri che concorrono a generare il transitorio sono quattro uguali due a due. I due filtri di tipo TB0564A (B-11, B-13) introducono un ritardo di gruppo di circa $4\mu s$. Vale lo stesso per i filtri SFEC10M7FA00 (B-15, B-20). In figura 3.8 si riportano le caratteristiche dei due tipi di filtro.



(a) *Filtro TB0564A.* (b) *Filtro SFEC10M7FA00.*

Figura 3.8: Caratteristiche dei filtri della catena dell'AGC.

Per garantire che il transitorio sia esaurito, si imposta un'attesa di $50us$; in questo modo, ad ogni aggiornamento del guadagno, si ignorano i primi 1400 campioni pari al 5% del totale (28000) che arriva in $1ms$. In figura 3.9 si riporta l'esempio dell'effetto dovuto al transitorio e il metodo correttivo usato nella ricerca dell'inviluppo.

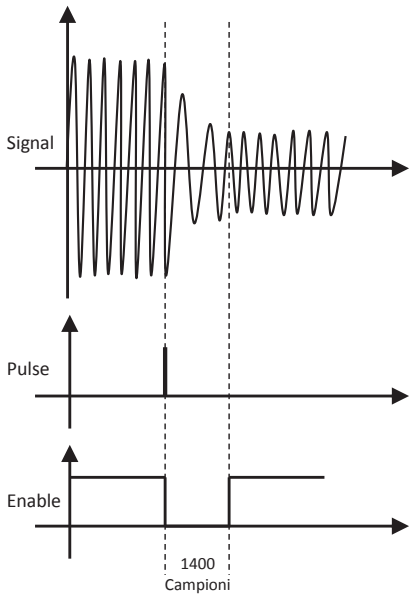


Figura 3.9: Esempio di transitorio dei dati in ingresso.

Il blocco ENV è stato realizzato tramite l'utilizzo di una Macchina a Stati Finiti con due soli stati. Il primo stato è dedicato all'attesa (stato ″A″), in cui si attende l'arrivo di 1400 campioni. Il secondo stato è di rilevamento (stato ″B″) in cui si rileva il picco del modulo del segnale. Nello stato ″B″, oltre alla rilevazione del picco, si monitora l'andamento del clock con periodo 1ms. Quando arriva un fronte dal clock con 1 ms si ha l'abilitazione dell'uscita e si ottiene l'aggiornamento del picco e si passa allo stato di attesa ″A″. Il *Design Summary* del blocco *ENV.vhd* implementato è riportato di seguito.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	ENV	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	41	301,440	1%	
Number of Slice LUTs	70	150,720	1%	
Number of occupied Slices	21	37,680	1%	
Number of LUT Flip Flop pairs used	70			
Number with an unused Flip Flop	36	70	51%	
Number with an unused LUT	0	70	0%	
Number of fully used LUT-FF pairs	34	70	48%	
Number of unique control sets	3			
Number of slice register sites lost to control set restrictions	7	301,440	1%	
Number of bonded IOBs	30	600	5%	
Number of DSP48E1s	0	768	0%	

Figura 3.10: *Design Summary* generato da XiliX ISE del blocco ENV.

Blocco AGC

L'ultimo blocco che comprende la parte di ricezione è diviso in due parti. La prima parte (AGC) verifica che il valore proveniente dal blocco ENV sia compreso in una delle 5 aree (rif. fig. 3.5) e una volta eseguita la comparazione utilizza il dato in ingresso come indirizzamento delle posizione di memoria di una ROM. Nella ROM sono contenuti i valori di amplificazione e attenuazione. La seconda parte(AGCtoAD8372) è adibita all'invio dei dati al AD8372 in maniera seriale. Il blocco AGC, implementato da *AGC.vhdl*, è composto da una Macchina a Stati Finiti con 4 stati principali. La Macchina a Stati Finiti è stata creata in modo tale che all'accensione abbia una convergenza rapida al valore corretto. Partendo dal valore di $-14db$, pari a metà della massima amplificazione, ogni stato lo affina con un valore sempre minore. Il primo stato affina di $+/-24db$ il guadagno, il secondo di $+/-12db$ e il terzo di $+/-3db$. Questi primi tre stati non verranno più utilizzati dato che la Macchina a Stati Finiti si assesta sull'ultimo stato. In questo stato apporta modifiche di $0db$, $+/-1db$ o $+/-3db$, ad ogni arrivo del picco, rispettando le regioni di decisione illustrate in figura 3.5.

Il valore di amplificazione, come già detto precedentemente, è contenuto in una ROM. L'indirizzamento viene eseguito utilizzando il valore dei amplificazione corretto. Nella ROM sono memorizzati 9 bit, 6 bit relativi al valore di amplificazione tra -6 e 32db e 3 bit per determinare l'inserimento o meno dell'attenuatore corrispondente. Di seguito si riportano i valori memorizzati all'interno della ROM.

$\mathbf{G_{max}}$ (dB)	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$	**Ampl.** (dB)	**HEX**		$\mathbf{G_{max}}$ (dB)	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$	**Ampl.** (dB)	**HEX**
-60	1	1	1	-6	1C4		-13	0	1	1	23	0E1
-59	1	1	1	-5	1C5		-12	0	1	1	24	0E2
-58	1	1	1	-4	1C6		-11	0	1	1	25	0E3
-57	1	1	1	-3	1C7		-10	0	1	1	26	0E4
-56	1	1	1	-2	1C8		-9	0	1	1	27	0E5
-55	1	1	1	-1	1C9		-8	0	1	1	28	0E6
-54	1	1	1	0	1CA		-7	0	1	1	29	0E7
-53	1	1	1	1	1CB		-6	0	1	1	30	0E8
-52	1	1	1	2	1CC		-5	0	0	1	13	057
-51	1	1	1	3	1CD		-4	0	0	1	14	058
-50	1	1	1	4	1CE		-3	0	0	1	15	059
-49	1	1	1	5	1CF		-2	0	0	1	16	05A
-48	1	1	1	6	1D0		-1	0	0	1	17	05B
-47	1	1	1	7	1D1		0	0	0	1	18	05C
-46	1	1	1	8	1D2		1	0	0	1	19	05D
-45	1	1	1	9	1D3		2	0	0	1	20	05E
-44	1	1	1	10	1D4		3	0	0	1	21	05F
-43	1	1	1	11	1D5		4	0	0	1	22	060
-42	1	1	1	12	1D6		5	0	0	1	23	061
-41	1	1	1	13	1D7		6	0	0	1	24	062
-40	1	1	1	14	1D8		7	0	0	1	25	063
-39	1	1	1	15	1D9		8	0	0	1	26	064
-38	1	1	1	16	1DA		9	0	0	1	27	065
-37	1	1	1	17	1DB		10	0	0	1	28	066
-36	1	1	1	18	1DC		11	0	0	1	29	067
-35	1	1	1	19	1DD		12	0	0	1	30	068
-34	1	1	1	20	1DE		13	0	0	1	31	069
-33	1	1	1	21	1DF		14	0	0	1	32	06A
-32	1	1	1	22	1E0		15	0	0	0	15	019
-31	1	1	1	23	1E1		16	0	0	0	16	01A
-30	1	1	1	24	1E2		17	0	0	0	17	01B
-29	1	1	1	25	1E3		18	0	0	0	18	01C
-28	1	1	1	26	1E4		19	0	0	0	19	01D
-27	1	1	1	27	1E5		20	0	0	0	20	01E
-26	1	1	1	28	1E6		21	0	0	0	21	01F
-25	1	1	1	29	1E7		22	0	0	0	22	020
-24	1	1	1	30	1E8		23	0	0	0	23	021
-23	1	1	1	31	1E9		24	0	0	0	24	022
-22	1	1	1	32	1EA		25	0	0	0	25	023
-21	0	1	1	15	0D9		26	0	0	0	26	024
-20	0	1	1	16	0DA		27	0	0	0	27	025
-19	0	1	1	17	0DB		28	0	0	0	28	026
-18	0	1	1	18	0DC		29	0	0	0	29	027
-17	0	1	1	19	0DD		30	0	0	0	30	028
-16	0	1	1	20	0DE		31	0	0	0	31	029
-15	0	1	1	21	0DF		32	0	0	0	32	02A
-14	0	1	1	22	0E0		33	×	×	×	n.i.	n.i

Tabella 3.4: Valori della ROM dell'AGC

In tabella 3.4 si vede nella prima colonna il valore del guadagno desiderato. Nelle colonne A_1, A_2, A_3 si indica con '1' se l'attenuatore corrispondente della catena è inserito '0' se è disinserito. Nella colonna ″Ampl.″ si indica il guadagno del amplificatore AD8372 e in ″HEX″ il valore esadecimale memorizzato nella ROM. I valori degli attenuatori sono inviati direttamente in uscita ma con logica inversa. Si collega la prima uscita a C_1^3, C_2^3, la seconda a C_1^2, C_2^2 e l'ultima a C_1^1, C_2^1(ref. fig 3.3).

Il secondo blocco che compone AGC si incarica invece di inviare il valore del guadagno all'amplificatore AD8372. L'AD8372 è un componente che al suo interno ospita due amplificatori variabili che regolano il guadagno in base al valore ricevuto in modo seriale. A questo scopo si è creato il blocco AGCtoAD8372 che ha il compito di inviare il dato ricevuto dal blocco AGC al dispositivo. In figura 3.11 si mostra il timing per la scrittura del valore di amplificazione del dispositivo.

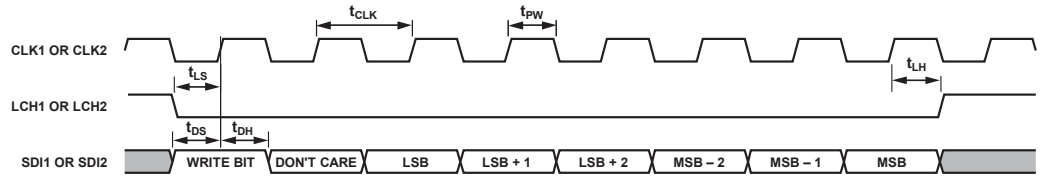


Figura 3.11: Timing della scrittura del valore di ampificazione su AD8372.

Il blocco AGCtoAD8372 è formato da una semplice macchina a stati che esegue le operazioni mostrate in figura 3.11. La sincronia viene effettuata inviando al dispositivo il clock di arrivo dei dati dall'ADC (FCO) negato sulla linea CLK. Portando la linea LCH a livello logico basso e inviando sulla linea dei dati seriali $SDIx$[4] un '1' si abilita la scrittura sul dispositivo del valore di guadagno. Successivamente invia uno '0' per dividere l'inizio della scrittura con la sequenza del livello di guadagno.

Per i successivi colpi di clock si manterrà sempre LCH a livello logico basso mentre si invierà su $SDIx$ il valore del guadagno desiderato, dal bit meno significativo (LSB) a quello più significativo (MSB). Infine si riporta la linea LCH a livello logico alto per permettere la variazione dell'amplificazione.

Per un gestione migliore del ricevitore completo tutti i blocchi descritti precedentemente sono stati raccolti in un package. Al suoi interno, inoltre, è contenuto un ulteriore blocco che genera un impulso ogni 28000 campioni corrispondente a circa $1ms$. Quest'ultimo permette la sincronizzazione del blocco che rileva il picco (Blocco ENV) con il blocco AGC. Se all'arrivo dell'impulso c'è stata una modifica al valore di guadagno, avvia l'aggiornamento del valore di amplificazione.

[4]Con x = 1 o 2 in base all'amplificatore relativo alla prima o seconda antenna. Al contrario le linee CLK e LCH sono condivise per entrambi gli amplificatori.

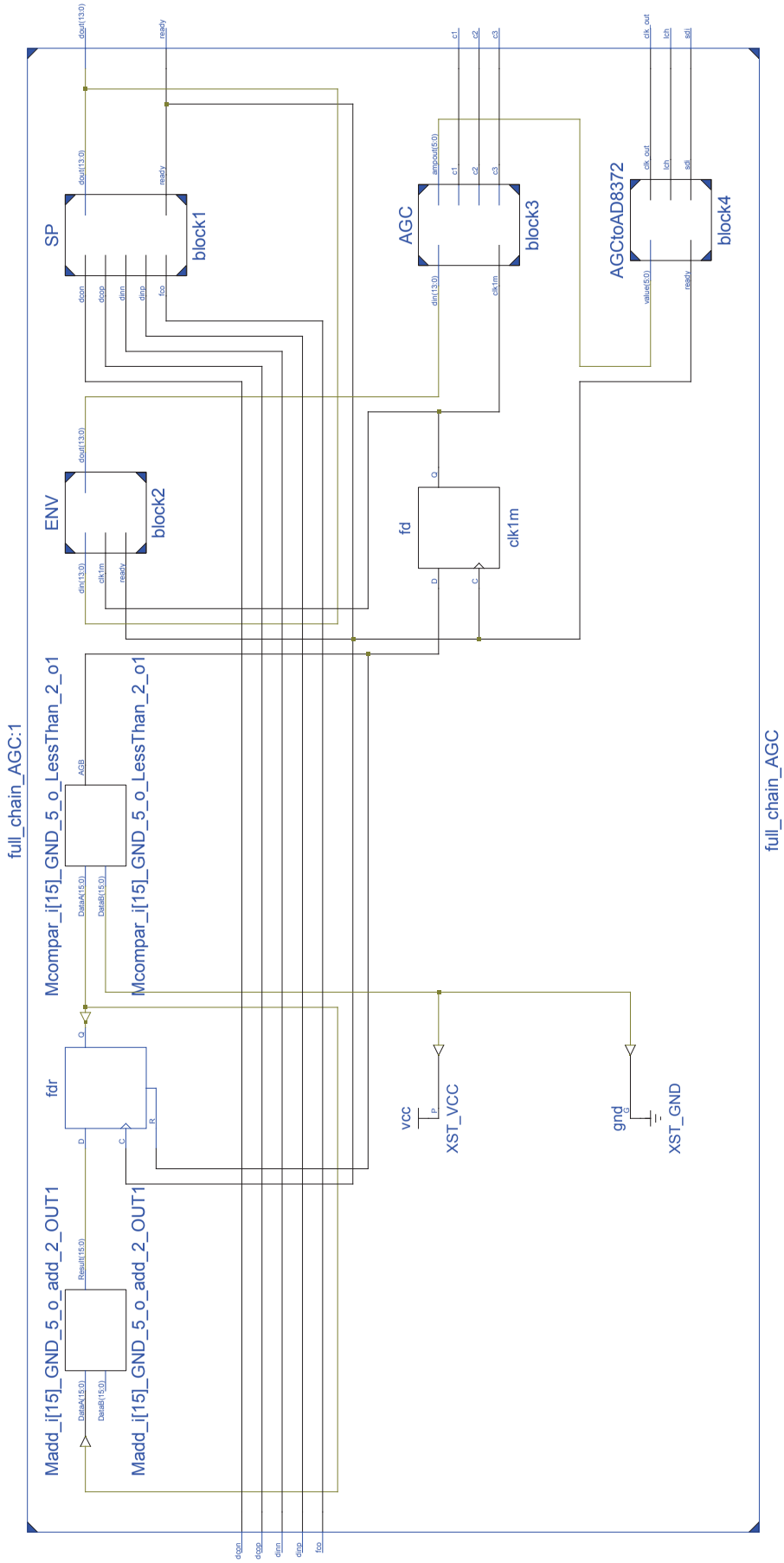


Figura 3.12: Schema a Blocchi RTL contenete la catena di AGC.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	full_chain_AGC	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	100	301,440	1%	
Number of Slice LUTs	134	150,720	1%	
Number of occupied Slices	76	37,680	1%	
Number of LUT Flip Flop pairs used	174			
Number with an unused Flip Flop	90	174	51%	
Number with an unused LUT	40	174	22%	
Number of fully used LUT-FF pairs	44	174	25%	
Number of unique control sets	8			
Number of slice register sites lost to control set restrictions	20	301,440	1%	
Number of bonded IOBs	26	600	4%	
Number of DSP48E1s	0	768	0%	

Figura 3.13: *Design Summary* della catena AGC

3.2 Sincronizzazione, Stima e Decodifica

Questa parte è il vero cuore del ricevitore e anch'esso è diviso in sotto parti per una più semplice implementazione e successiva verifica. Lo schema che andremo ad utilizzare è di tipo retroazionato. Uno schema a blocchi esemplificativo del sistema è rappresentato in figura 3.14. Lo schema retroazionato, come già detto precedentemente, permette agli oscillatori di essere meno precisi, correggendo la loro imprecisione grazie alla retroazione.

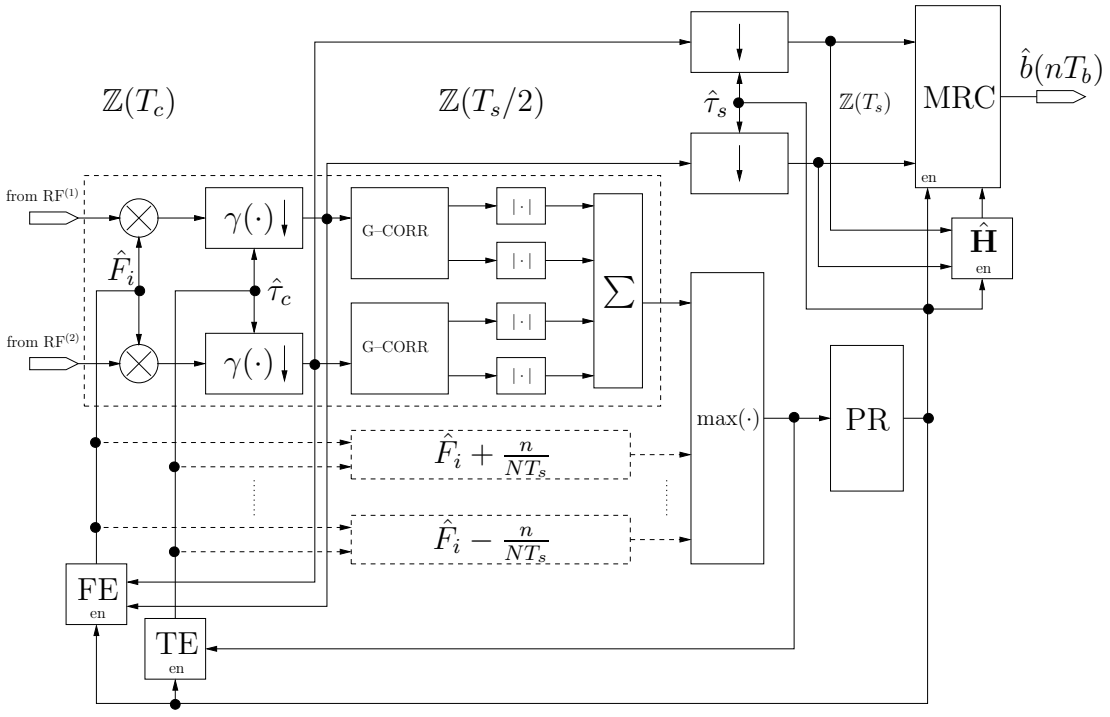


Figura 3.14: Schema a blocchi del sincronizzatore e stima di canale retroazionato.

Lo schema, inoltre, ha un pregio di lavorare sul tempo intermedio $\mathbb{Z}(T_s/2)$ che permette oltre che una maggiore gestione del segnale anche un minor consumo dato dalla frequenza inferiore di commutazione.

Va in primo luogo chiarita la presenza di blocchi tratteggiati identificati dalla dicitura $\hat{F}_i \pm \frac{n}{NT_s}$ che rappresentano repliche della struttura centrale dello schema, essa stessa tratteggiata, dove al posto del mixer alla frequenza stimata, troviamo versioni traslate di multipli della quantità $F_0 = \frac{1}{NT_s}$ con N lunghezza del preambolo. Il motivo di queste repliche è conseguenza del fatto che un sistema di correlazione sulle sequenze di Golay ha una sua risposta in frequenza che è data dall'espressione [9]

$$C(f) = \frac{\sin^2 N\pi f T_s}{N \sin^2 \pi f T_s} \tag{3.3}$$

presentando degli zeri proprio all'altezza delle frequenze multiple di F_0. Se noi utilizziamo sequenze di Golay lunghe $N = 64$ per un tempo di simbolo $T_s = \frac{1}{140}$

ms otteniamo $F_0 = 2.1875$ kHz del tutto insufficienti a coprire una banda di ± 5 kHz con un solo correlatore. Per rilevare dunque con certezza il picco su un frequency offset di quell'entità occorre utilizzare correlatori sintonizzati pure a $\pm nF_0$ con $n = \pm 1, \pm 2$ per un totale di 5 blocchi di correlazione. Con una struttura del genere riusciamo a rilevare picchi di correlazione all'interno di una banda maggiore. In Fig. 3.15 viene messo a confronto un schema a singola catena di correlazione con quello a 5 catene.

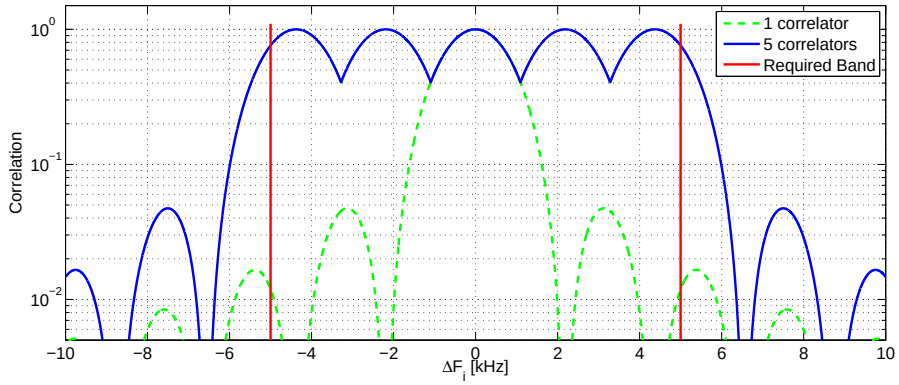


Figura 3.15: Andamento del picco di correlazione in funzione dell'offset di frequenza ΔF_i.

A questo punto tutti e 5 i flussi entrano nel blocco che sceglie il massimo istantaneo e su tale segnale viene eseguito il rilevamento del picco per poter sincronizzarci con il frame in trasmissione. Questo segnale di *sync* non serve solo a temporizzare il campionamento sui tempi $\mathbb{Z}(T_s)$ nonchè il blocco di stima di canale ($\hat{\mathbf{H}}$) e di detection (MRC), ma serve altresì ad abilitare altri due blocchi in retroazione importantissimi per il funzionamento del ricevitore: lo stimatore di frequency offset (FE) e lo stimatore dell'istante di campionamento o *timing* (TE) che descriveremo nel seguito. La parte comprendente la stima di canale ($\hat{\mathbf{H}}$), di detection (MRC) e il campionatore fanno parte del blocco di decodifica trattato nel paragrafo seguente (par. 3.2.4).

Nella fase di implementazione il ricevitore è stato diviso in tre sottoparti principali:

- Package Acquisizione
- Package Correlazione
- Elementi per la Sincronia
- Package Interpretazione

La prima parte si compone del mixer digitale che porta il segnale in banda base e poi il segnale viene filtrato per il matching filter, che è una versione ribaltata e traslata dell'impulso in trasmissione.

La seconda parte è formata dai blocchi di correlazione che permettono di trovare il picco di correlazione per l'individuazione del centro dell'occhio.

La terza parte comprende tutti quei blocchi più generali che sono utili per la sincronizzazione trovando la frequenza o la durata del frame in trasmissione.

L'ultima si occupa della decodifica dei vettori che arrivano derivando da essi i bit trasmessi e ricostruendo il campione.

3.2.1 Acquisizione

Questo blocco è descritto tramite il file *acquisition.vhd* e contiene il blocco Mixer e il Filtro Campionatore. Questo blocco verrà successivamente replicato per ogni blocco di correlazione associandogli la frequenza di centro banda a cui si aggancerà la portante per riportare il segnale in banda base.

Mixer

Il mixer è realizzato da due blocchi principali; il primo blocco è chiamato Local Oscillator e genera i campioni di seno e coseno alla frequenza desiderata. Il secondo blocco invece è formato semplicemente dai due moltiplicatori che moltiplicano il segnale in ingresso ai valori provenienti dall'Local Oscillator.

In dettaglio, il Local Oscillator è creato tramite un algoritmo Cordic che genera le sequenze di seno e coseno campionate a 28MHz[5]. Un dato temporaneo tiene traccia della posizione dell'angolo del vettore e si incarica l'algoritmo Cordic di generare i valori per il processo che riporta il segnale in banda base. Uno schema molto utilizzato è quello di usare l'algoritmo Cordic per la rotazione del dato passato in ingresso. In questo caso peró essendoci due flussi, uno per antenna, l'utilizzo del secondo circuito che implementa l'algoritmo Cordic genera un implementazione con una occupazione risultante superiore.

L'algoritmo Cordic utilizzato è di tipo Rotation, che ha la particolarità di generare in uscita i valori di seno e coseno dell'angolo passatogli in ingresso. L'angolo verrà generato dal blocco Frequency Estimator che vedremo successivamente.

[5]Frequenza di arrivo dei dati in ingresso al ricevitore.

L'algoritmo Cordic fornisce un metodo iterativo per eseguire rotazioni vettoriali da angoli arbitrari usando solo operazioni di shift e addizione. L'algoritmo è derivato dalla trasformazione di rotazione generale:

$$x' = x \cdot \cos\phi - y \cdot \sin\phi$$
$$y' = y \cdot \cos\phi + x \cdot \sin\phi$$

che ruota un vettore in un piano cartesiano dell'angolo ϕ. Queste possono essere modificate in modo tale che:

$$x' = \cos\phi \cdot [x - y \cdot \tan\phi]$$
$$y' = \cos\phi \cdot [y + x \cdot \tan\phi]$$

Se la rotazione dell'angolo viene peró limitata in modo tale che $\tan(\phi)=\pm 2^{-i}$, la moltiplicazione per il termine tangente è ridotta a semplici operazioni di shift. La rotazione per angoli arbitrari è ottenibile mediante piccole rotazioni sempre minori. Se la decisione ad ogni iterazione, i, è in quale delle due direzioni ruotare al posto di ruotare o meno, allora il termine $\cos(\delta)$ diventa costante (perchè $\cos(\delta) = \cos(-\delta)$). Le rotazioni iterative possono quindi essere espresse con

$$x_{i+1} = K_i \cdot [x_i - y_i \cdot d_i \cdot 2^{-i}]$$
$$y_{i+1} = K_i \cdot [y_i + x_i \cdot d_i \cdot 2^{-i}]$$

dove

$$K_i = \cos(\arctan \cdot 2^{-i}) = \frac{1}{\sqrt{1 + 2^{-2i}}}$$
$$d_i = \pm 1$$

Rimuovendo la scala costante dalle equazioni iterative produce un algoritmo di shift e somma per la rotazione vettoriale. Il prodotto di K_i puó essere replicato altrove nel sistema o trattato come parte di un sistema di elaborazione di guadagno. Tale prodotto si avvicina a 0.6073 quando le iterazioni tendono ad infinito. Pertanto l'algoritmo di rotazione ha un guadagno pari a 1.647. Il guadagno esatto dipende dal numero di iterazioni e obbedisce alla relazione

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

L'angolo di rotazione è definito in modo univoco dalla sequenza delle direzioni delle rotazioni elementari. Questa sequenza puó essere rappresentata da un vettore di decisione. Il set dei vettori possibili è una misura angolare su un sistema basato su valori prefissati di arcotangente. Un migliore metodo di conversione si realizza usando sommatori/sottrattori supplementari che accumulano gli angoli di rotazione ad ogni iterazione. Gli angoli elementari possono essere espressi in qualsiasi unità. L'accumulazione dell'angolo aggiunge un'ulteriore equazione all'algoritmo:

$$z_{i+1} = z_i - d_i \cdot \arctan(2^{-i})$$

Ovviamente, in casi in cui l'angolo è utile in base arcotangente quest'ultima equazione non è necessaria. La rotazione Cordic di solito viene gestita in due modalità. La prima, denominata Rotation, ruota il vettore d'ingresso di un angolo specifico dato come argomento. La seconda modalità , chiamata Vectoring, ruota il vettore d'ingresso dall'asse x registrando l'angolo necessario a quella rotazione.

In modalità Rotation, l'angolo accumulatore viene inizializzato con l'angolo di rotazione desiderato. La decisione di come ruotare il vettore, a ciascuna iterazione, è tale da diminuire l'entità dell'angolo residuo. Quindi la rotazione dell'angolo si basa sul segno dell'angolo residuo dopo ogni passaggio. Per la modalità Rotation le equazioni dell'algoritmo Cordic sono

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \arctan(2^{-i})$$

dove

$$d_i = \begin{cases} -1 & se \qquad z_i < 0 \\ \\ +1 & altrimenti \end{cases}$$

che portano ai seguenti risultati

$$x_n = A_n \cdot [x_0 \cdot \cos(z_0) - y \cdot \sin(z_0)]$$
$$y_n = A_n \cdot [y_0 \cdot \cos(z_0) + x \cdot \sin(z_0)]$$
$$z_n = 0$$
$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

In modalità Vectoring, invece, è il vettore che viene ruotato finchè l'angolo non è tale da allineare il risultato con l'asse delle ascisse. Le operazioni ad ogni iterazione cercano di ridurre al minimo la componente y lasciando la decisione della rotazione al segno del residuo. Se l'accumulatore è inizialmente zero basterà contare l'angolo di traslazione alla fine delle iterazioni. L'algoritmo Cordic Vectoring rispetta le seguenti equazioni.

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \arctan(2^{-i})$$

dove

$$d_i = \begin{cases} +1 & se \qquad y_i < 0 \\ \\ -1 & altrimenti \end{cases}$$

che portano ai seguenti risultati

$$x_n = A_n \cdot \sqrt{x_0^2 + y_0^2}$$
$$y_n = 0$$
$$z_n = z_0 + \arctan(\frac{y_0}{x_0}$$
$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

Quindi nell'implementazione del LocalOscillator si utilizza l'algoritmo Cordic Rotation. In modo contrario alla teoria l'algoritmo Cordic viene realizzato in modo un po' diverso. L'angolo di accumulazione parte nullo e il vettore parte dall'asse delle ascisse. Ad ogni iterazione si somma o si sottrae il valore dell'angolo relativo all'iterazione finchè l'angolo risultante risulta paragonabile a quello d'ingresso. Per realizzare l'oscillatore, infine, si ha bisogno di una uscita ciclica del vettore, e quindi viene memorizzato l'angolo di rotazione sommando ad ogni iterazione il valore passatogli dal Frequency Enable.

Dato che le operazioni devono essere effettuate ad una frequenza di almeno $13x28MHz$ si rende necessario l'utilizzo di una architettura a Pipeline. Ad ogni iterazione si memorizzano i dati dell'angolo di rotazione di x e y e l'angolo del residuo in modo tale si possano attuare le operazioni di shift e somma in base al valore dell'angolo. Le somme per ogni iterazione sono fisse e vengono eseguite in parallelo. I dati escono dopo 13 cicli di clock e vengono successivamente moltiplicati per i campioni che arrivano per attuare la decodifica.

Il Mixer quindi riceve in ingresso i dati con definizione 14 bit provenienti dal blocco SP, vengono poi moltiplicati per i vettori del mixer anch'essi di 14 bit risultando quindi in uscita un vettore da 28 bit. Il vettore da 28 bit viene poi scalato a 14bit mantenendo invariata la precisione, ma sufficiente da ridurre il consumo di risorse. Di seguito si riporta lo schema a blocchi del Mixer completo e il *Design Summary* dell'implementazione.

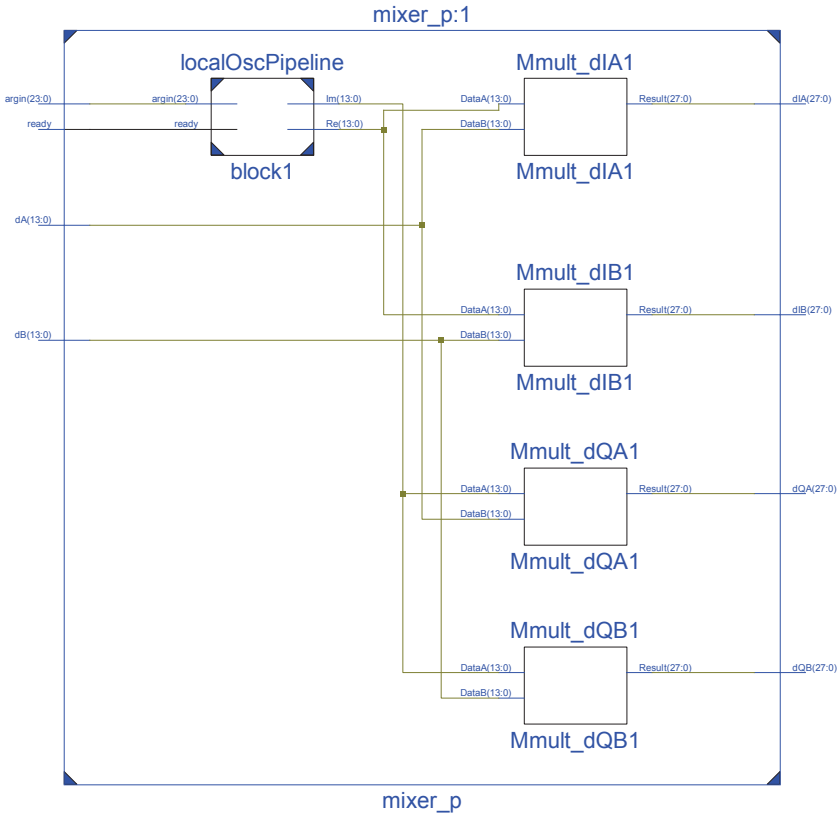


Figura 3.16: Schema a blocchi RTL del Mixer.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	Mixer	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	988	301,440	1%	
Number of Slice LUTs	3,198	150,720	2%	
Number of occupied Slices	1,124	37,680	2%	
Number of LUT Flip Flop pairs used	3,372			
Number with an unused Flip Flop	2,451	3,372	72%	
Number with an unused LUT	174	3,372	5%	
Number of fully used LUT-FF pairs	747	3,372	22%	
Number of unique control sets	6			
Number of slice register sites lost to control set restrictions	29	301,440	1%	
Number of bonded IOBs	165	600	27%	
Number of DSP48E1s	0	768	0%	

Figura 3.17: *Design Summary* del Mixer.

Filtraggio

La parte di filtraggio inizialmente fu concepita per essere un solo grande filtro di forma ribaltata e traslata del filtro di trasmissione. Ma la sua implementazione sarebbe stata troppo esosa di risorse e da una prima analisi avrebbe occupato più del 50% degli CLB dell'FPGA. Il segnale, successivamente al filtro, per permettere l'analisi dei valori al centro dell'occhio di correlazione, viene campionato vanificando così l'utilizzo di un filtro cosí selettivo.

Per ovviare a questo problema il filtro è stato diviso in due parti: un filtro da 21 tappi e uno da 98 tappi con un campionamento intermedio. La struttura del filtro è riportata in figura 3.18. Si nota che dopo il primo filtro il segnale viene campionato per 20 e successivamente al secondo filtro per 5, portandolo da una frequenza di 28MHz ad una frequenza finale di 280KHz.

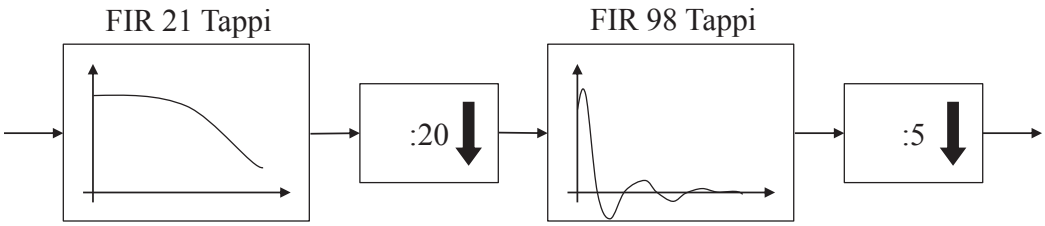


Figura 3.18: Schema a blocchi del filtro di ricezione.

Il primo filtro (21 tappi) è semplicemente un filtro passa basso che elimina eventuali componenti residue fuori banda. Il filtro a 98 tappi è il filtro di Nyquist utilizzato in trasmissione ribaltato, ma essendo a frequenza diversa, convoluzionato con un filtro passa basso dello stesso tipo del precedente. Questo espediente è stato adottato per adattare il filtro creato per una frequenza di 700KHz alla frequenza utilizzata successivamente al campionamento di 1400KHz. Di seguito (fig. 3.19) si riportano i moduli della risposta in frequenza del due filtri.

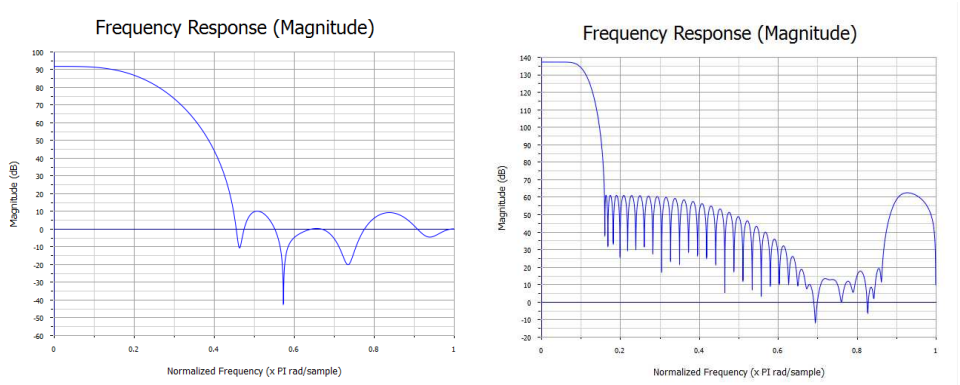


Figura 3.19: Moduli della risposta in frequenza dei due filtri di ricezione.

Anche in questo caso, come successo precedentemente nell'implementazione del filtro in trasmissione, ci si è resi conto dell'enorme impatto che questi hanno sull'utilizzo di risorse dell'FPGA.

L'implementazione ″classica″ dei due filtri, ha la stessa forma base già utilizzata in precedenza; i dati della risposta impulsiva sono memorizzati in una ROM, all'arrivo dei campioni vengono caricati i coefficienti e si eseguono le operazioni di ritardo e somma tipiche di un filtro FIR. In cascata ai filtri ci sono i campionatori che sono composti semplicemente da un elemento di memoria che memorizza il dato ogni 20 o 5 cicli di clock per il campionamento per 20 o per 5.

Al contrario se si utilizza l'implementazione tramite Core IP le operazioni di campionamento vengono eseguite all'interno del blocchetto e quindi non si necessita di ulteriore componentistica per eseguire l'operazione. Per semplificare l'implementazione i filtri sono stati inglobati in un package chiamato CampFiltre e in figura 3.20 e 3.21 si mostrano i *Design Summary* delle due diverse implementazioni.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	CampFiltreM	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	9,513	301,440	3%	
Number of Slice LUTs	28,386	150,720	18%	
Number of occupied Slices	7,758	37,680	20%	
Number of LUT Flip Flop pairs used	29,105			
Number with an unused Flip Flop	19,933	29,105	68%	
Number with an unused LUT	719	29,105	2%	
Number of fully used LUT-FF pairs	8,453	29,105	29%	
Number of unique control sets	8			
Number of slice register sites lost to control set restrictions	31	301,440	1%	
Number of bonded IOBs	86	600	14%	
Number of DSP48E1s	0	768	0%	

Figura 3.20: *Design Summary* del Filtro di ricezione ad implementazione manuale.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	CampFiltreIP	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	1,518	301,440	1%	
Number of Slice LUTs	753	150,720	1%	
Number of occupied Slices	396	37,680	1%	
Number of LUT Flip Flop pairs used	985			
Number with an unused Flip Flop	10	985	1%	
Number with an unused LUT	232	985	23%	
Number of fully used LUT-FF pairs	743	985	75%	
Number of unique control sets	8			
Number of slice register sites lost to control set restrictions	20	301,440	1%	
Number of bonded IOBs	86	600	14%	
Number of DSP48E1s	86	768	11%	

Figura 3.21: *Design Summary* del Filtro di ricezione a Core IP.

Sapendo che i filtri saranno replicati per ogni antenna e per ogni blocco di correlazione si privilegerà l'implementazione con meno utilizzo di area. Purtroppo in questo caso nessuna delle due implementazioni è da considerarsi ottima.

Anche se nell'implementazione con Core IP si ha un basso utilizzo di LUTs, si ha un consumo del 11% delle slice DSP, il quale, considerando le repliche che serviranno nel ricevitore ($2x5 = 10$), sale e sfora di poco il totale dei blocchi DSP contenuti nella Virtex 6. Allo stesso modo se si utilizza solo l'implementazione ″classica″ si utilizza oltre il 100% delle LUTs dell'FPGA.

Si deve quindi trovare un implementazione mista del blocco in modo da avere un utilizzo controllato dell'FPGA. Dopo una prima analisi si vede come i coefficienti del filtro a 21 tappi sono rappresentabili con l'utilizzo di soli $14Bit$ e quindi è possibile utilizzare l'implementazione ″classica″ senza gravare eccessivamente sull'area usata. Al contrario i coefficienti del filtro a 98 tappi hanno una variabilità più alta e quindi si è pensato di utilizzare un'implementazione a Core IP. Quindi si utilizzerà l'implementazione classica per il primo filtro e quella a Core IP per il secondo portando all'implementazione del filtro seguente:

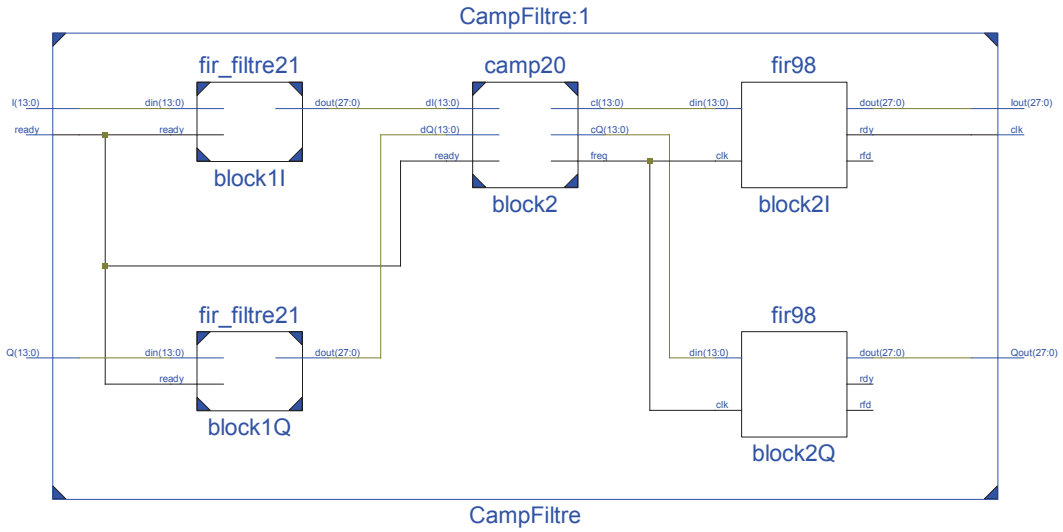


Figura 3.22: Schema a blocchi RTL del blocco CampFiltre.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	CampFiltre	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	2,810	301,440	1%	
Number of Slice LUTs	4,865	150,720	3%	
Number of occupied Slices	1,477	37,680	3%	
Number of LUT Flip Flop pairs used	5,256			
Number with an unused Flip Flop	3,037	5,256	57%	
Number with an unused LUT	391	5,256	7%	
Number of fully used LUT-FF pairs	1,828	5,256	34%	
Number of unique control sets	5			
Number of slice register sites lost to control set restrictions	16	301,440	1%	
Number of bonded IOBs	86	600	14%	
Number of DSP48E1s	44	768	5%	

Figura 3.23: *Design Summary* del blocco CampFiltre.

Se si fa un semplice calcolo dell'utilizzo finale delle LUTs e del blocco DSP si vede che solo la parte di filtraggio utilizzerà il 30% di slice LUTs e il 50% di slice DSP. A questo punto dato il consumo eccessivo delle risorse dell'FPGA soprattutto se si considera un eventuale ricevitore stereo si riducono i blocchi di correlazione da 5 a 3.

Con la divisione del filtro peró nasce un problema relativo al campionamento del segnale. La parte di campionamento originariamente era eseguita

successivamente al filtro. In questo modo era semplice riuscire a campionare i dati d'uscita in modo tale che il campione si trovasse proprio al centro dell'occhio di correlazione. Purtroppo la divisione del filtro in due step, se da un lato ha diminuito l'utilizzo di risorse, dall'altro ha reso impossibile l'utilizzo del campionamento ″dinamico″ successivo al filtro.

Si è introdotto, quindi, un ulteriore blocco prima dei filtri che ritarda di una determinata quantità i dati in arrivo.

Cosí facendo si puó mantenere il campionamento fisso dei filtri pari a 100 e si modificherà l'uscita dei campioni dal blocco che li precede per creare l'effetto desiderato. Il blocco, chiamato shifter, riceve in ingresso un valore di ritardo del campione dal picco di correlazione che verrà generato dal blocco Time Estimetor (TE) che vedremo successivamente. Il valore indicherà l'uscita prescelta di un registro a scorrimento FIFO in cui sono memorizzati temporaneamente i campioni provenienti dal mixer. I campioni memorizzati sono 100 e sono il valore massimo dello scostamento dal centro dell'occhio di correlazione generato dal TE.

Infine si riportano lo schema a blocchi e *Design Summary* finale del package Acquisition.

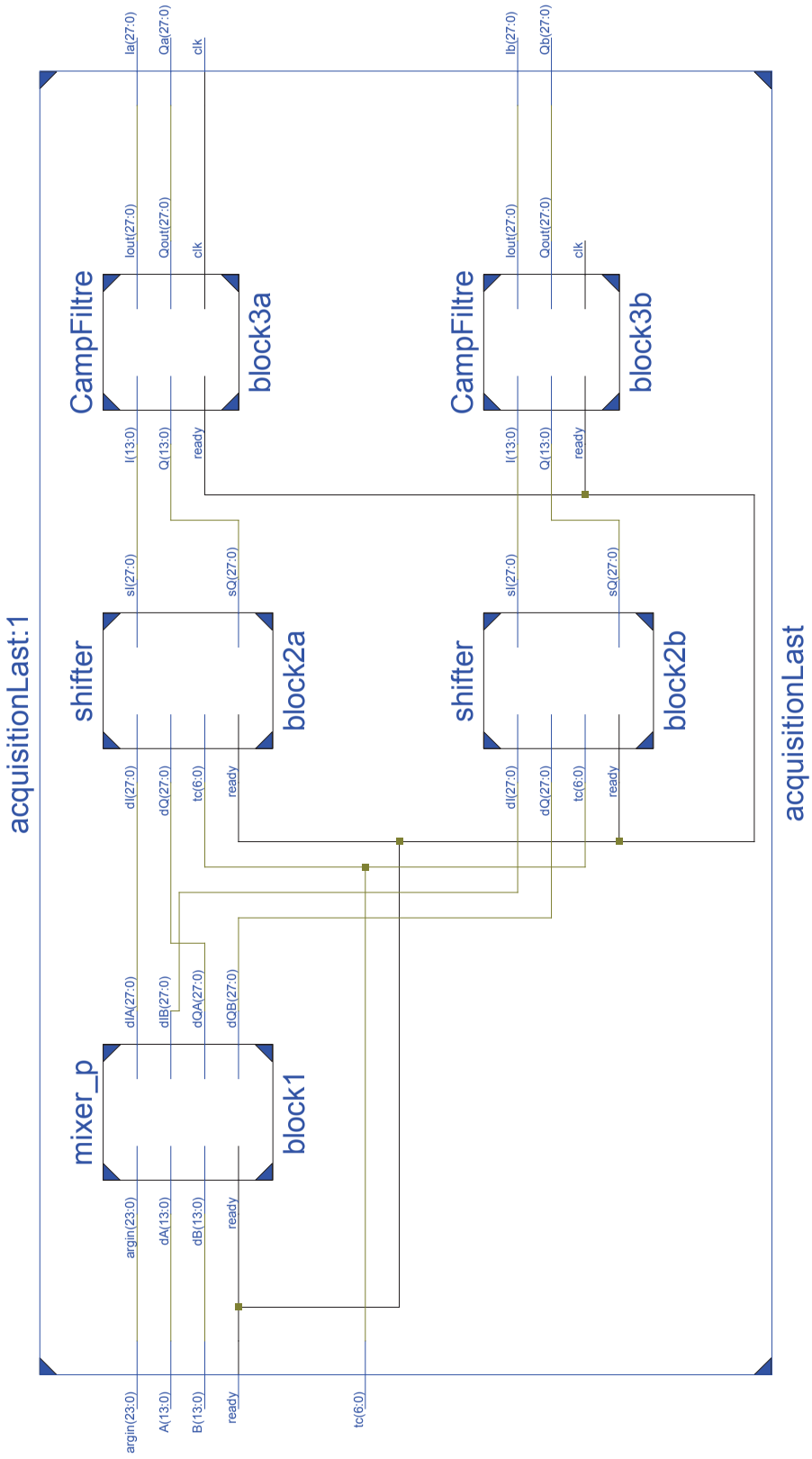


Figura 3.24: Schema a blocchi RTL del blocco Acquisition.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	acquisition	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	12,190	301,440	4%	
Number of Slice LUTs	15,693	150,720	10%	
Number of occupied Slices	4,876	37,680	12%	
Number of LUT Flip Flop pairs used	17,954			
Number with an unused Flip Flop	8,034	17,954	44%	
Number with an unused LUT	2,261	17,954	12%	
Number of fully used LUT-FF pairs	7,659	17,954	42%	
Number of unique control sets	9			
Number of slice register sites lost to control set restrictions	39	301,440	1%	
Number of bonded IOBs	173	600	28%	
Number of DSP48E1s	88	768	11%	

Figura 3.25: *Design Summary* del blocco Acquisition.

3.2.2 Correlazione

La parte di correlazione si compone del correlatore di Golay, del Modulo e della Somma. La somma dei moduli verrà inviata poi ai blocchi di sincronia.

Golay Correlator

Il correlatore di Golay esegue le stesse operazioni che si utilizzano nella generazione del preambolo (ref. 2.2). In questo modo si esegue l'autocorrelazione del simbolo e si genera il picco necessario alla sincronizzazione dell'intero ricevitore. L'implementazione del correlatore è stata eseguita tramite la traduzione del circuito riportato in figura 3.26. Con D si esprimono i ritardi e con W si attua il cambio di segno.

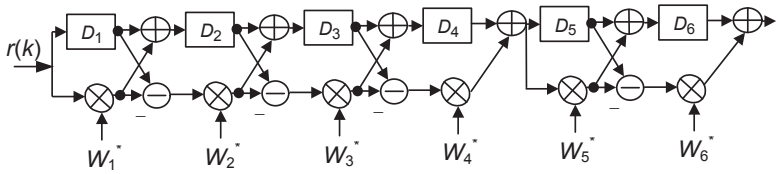


Figura 3.26: Schema a blocchi di principio del Correlatore di Golay.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	gCorr	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	492	301,440	1%	
Number of Slice LUTs	480	150,720	1%	
Number of occupied Slices	143	37,680	1%	
Number of LUT Flip Flop pairs used	500			
Number with an unused Flip Flop	12	500	89%	
Number with an unused LUT	20	500	0%	
Number of fully used LUT-FF pairs	468	500	10%	
Number of unique control sets	2			
Number of slice register sites lost to control set restrictions	8	301,440	1%	
Number of bonded IOBs	85	600	4%	
Number of DSP48E1s	0	768	0%	

Figura 3.27: *Design Summary* del blocco gCorr.

Modulo e Somma

Il modulo e la somma sono effettuati sui segnali d'uscita del Golay Correator di entrambe le antenne. Ogni correlatore ha due sequenze d'uscita quindi il blocco relativo avrà otto ingressi.

Nello specifico si avranno due ingressi, uno per ogni antenna, che a sua volta si dividono in reale ed immaginario diventando quattro. Ogni segnale, infine, passando attraverso i correlatori, si raddoppia e si ottengono otto segnali che dovranno essere elaborati da questo blocco.

Il modulo di un numero complesso è il risultato di

$$ans = \sqrt{Re^2 + Im^2} \tag{3.4}$$

per facilitare le operazioni si è svolto semplicemente il quadrato dei moduli, evitando il calcolo della radice quadrata. Questo ha permesso di semplificare notevolmente l'implementazione e di ridurre l'utilizzo di LUTs. Ogni segnale prima di essere sommato, viene elevato al quadrato semplicemente moltiplicandolo per se stesso. Nello schema RTL (fig. 3.30) si vede in modo più chiaro come è stata pensata l'implementazione.

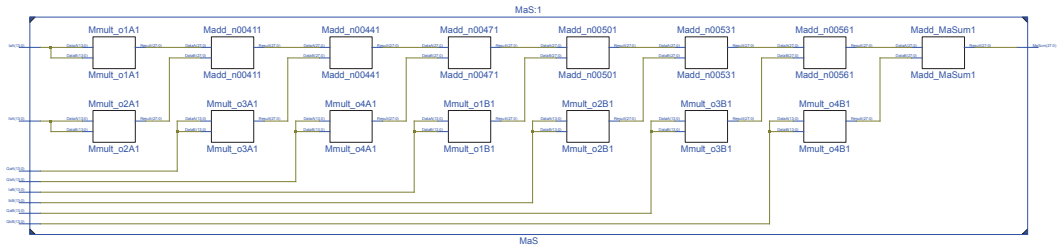


Figura 3.28: Schema a blocchi RTL del blocco Modulo e Somma.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	MaS	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	0	301,440	0%	
Number of Slice LUTs	1,364	150,720	1%	
Number of occupied Slices	354	37,680	1%	
Number of LUT Flip Flop pairs used	1,364			
Number with an unused Flip Flop	1,364	1,364	100%	
Number with an unused LUT	0	1,364	0%	
Number of fully used LUT-FF pairs	0	1,364	0%	
Number of unique control sets	0			
Number of slice register sites lost to control set restrictions	8	301,440	1%	
Number of bonded IOBs	140	600	4%	
Number of DSP48E1s	0	768	0%	

Figura 3.29: *Design Summary* del blocco Modulo e Somma.

Il blocco che verifica la correlazione riceve in ingresso i dati provenienti dalle due antenne e ne calcola la correlazione e ne esegue il modulo e la somma. Di seguito si riporta lo schematico completo del package *correlation*.

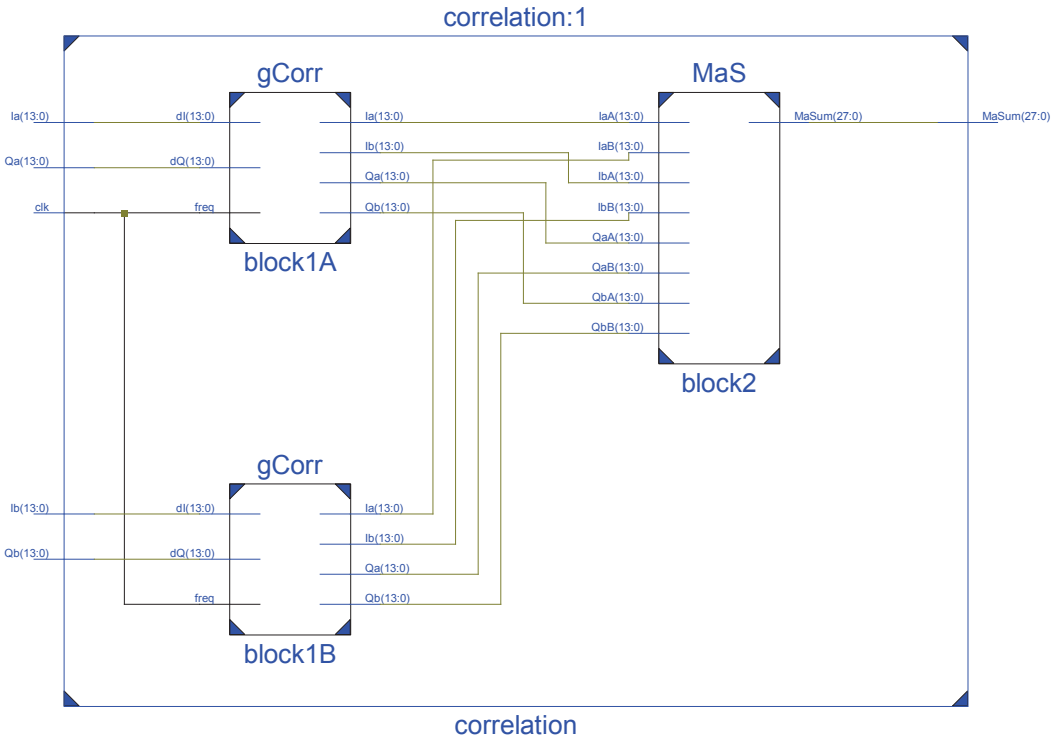


Figura 3.30: Schema a blocchi RTL del package Correlation.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	Correlation	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	984	301,440	1%	
Number of Slice LUTs	2,315	150,720	1%	
Number of occupied Slices	645	37,680	1%	
Number of LUT Flip Flop pairs used	2,355			
Number with an unused Flip Flop	1,387	2,355	58%	
Number with an unused LUT	40	2,355	1%	
Number of fully used LUT-FF pairs	928	2,355	39%	
Number of unique control sets	2			
Number of slice register sites lost to control set restrictions	0	301,440	0%	
Number of bonded IOBs	85	600	14%	
Number of DSP48E1s	0	768	0%	

Figura 3.31: *Design Summary* del package Correlation.

3.2.3 Sincronia

Questa sezione dello schema si compone di diverse parti che sono fondamentali per la realizzazione della sincronizzazione. Essa si compone di:

- Massimo
- Peak Detector
- Frequency Estimator
- Time Estimator

Massimo

La parte di massimo si occupa nel rilevare il massimo tra i segnali provenienti dai 5 correlatori. Il blocco è semplice ma basilare per la verifica del picco data dal Peak Detector, e del Time Estimator.

Di seguito si riporta semplicemente uno schema a blocchi che mostra l'algoritmo per la ricerca del massimo.

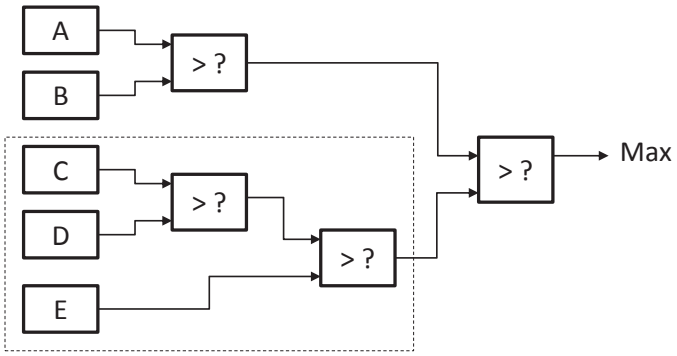


Figura 3.32: Schema a blocchi dell'algoritmo di ricerca.

Nell'implementazione si è visto come i blocchi di filtraggio e di correlazione occupassero in maniera eccessiva le risorse dell'FPGA quindi si è optato per una riduzione dei blocchi di correlazione portandoli da 5 a 3. Risulta quindi necessaria la modifica del blocco di controllo del massimo con confronto su solo tre valori d'ingresso. Di seguito si riporta il *Design Summary* del blocco Max3 che realizza l'algoritmo della parte tratteggiata nella figura.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	max_3	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	0	301,440	0%	
Number of Slice LUTs	57	150,720	1%	
Number of occupied Slices	23	37,680	1%	
Number of LUT Flip Flop pairs used	57			
Number with an unused Flip Flop	57	57	100%	
Number with an unused LUT	0	57	0%	
Number of fully used LUT-FF pairs	0	57	0%	
Number of unique control sets	0			
Number of slice register sites lost to control set restrictions	0	301,440	0%	
Number of bonded IOBs	112	600	14%	
Number of DSP48E1s	0	768	0%	

Figura 3.33: *Design Summary* del Massimo.

Peak Detector

In questo blocco il segnale che entra viene confrontato con una sua versione filtrata attraverso un filtro $g_{th}(\cdot)$ che ha lo scopo di valutare la soglia ottimale[6] per il rilevamento del picco.

Non appena dal confronto con la soglia si rileva un picco, viene eseguito un ulteriore confronto tra tale valore e il successivo per capire su quale campione stia esattamente il massimo del segnale. Da un confronto tra questi campioni si decide su quale istante inizia il frame: se il campione precedente era più alto allora dal successivo inizia il frame, altrimenti si aspettano 2 campioni successivi prima di dare il segnale di *enable*.

In Fig. 3.34 viene riportato lo schema di questo blocco.

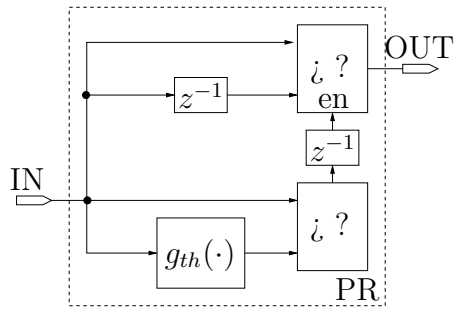


Figura 3.34: Schema a blocchi del rilevatore di picco.

[6]Ottimale nel senso che minimizza le probabilità di *false allarm* e di *peak undetection*.

La risposta impulsiva del filtro di soglia è data da

$$g_{th}(nT_s/2) = K_{th}, \quad \text{per } 0 \leq n \leq 127 \tag{3.5}$$

con K_{th} costante di ottimizzazione che abbiamo empiricamente stimato valere 0.024.

Nell'implementazione il filtro di soglia è stato ottimizzato creando un blocco che controlla la media del segnale in modo tale da riuscire ad identificare il picco in modo univoco senza avere falsi positivi. Per semplicità le operazioni di somma sono eseguite sui valori scalati del termine K_{th}.

Per il valore di confronto si esegue semplicemente una divisione sui 128 valori per ottenere il valore medio. Solo se la verifica della condizione che il valore in ingresso è superiore alla media è verificata si abilita il confronto tra il valore precedente e il successivo. Infine se tutti confronti danno esito positivo allora si ha la commutazione dell'uscita che identifica un picco.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	PR	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	62	301,440	1%	
Number of Slice LUTs	417	150,720	1%	
Number of occupied Slices	117	37,680	1%	
Number of LUT Flip Flop pairs used	417			
Number with an unused Flip Flop	355	417	85%	
Number with an unused LUT	0	417	0%	
Number of fully used LUT-FF pairs	62	417	14%	
Number of unique control sets	5			
Number of slice register sites lost to control set restrictions	30	301,440	1%	
Number of bonded IOBs	30	600	5%	
Number of DSP48E1s	0	768	0%	

Figura 3.35: *DesignSummary* del Peak Detector.

Frequency Estimator

Questo blocco si occupa di stimare l'entità dell'offset di frequenza presente sul dispositivo in ricezione rispetto alla frequenza nominale. Esso sfrutta il fatto che i simboli ricevuti relativi al preambolo sono uguali in gruppo di 4 a meno del segno e, per l'appunto, dell'offset di frequenza che andremo a stimare. Quindi, per ciascuno di questi gruppi di 4, una volta corretto il segno mediante la moltiplicazione per un coefficiente $c_\ell = \pm 1$,è possibile valutare l'offset di fase secondo la relazione

$$\Delta\phi = \arg(c_\ell r_\ell \cdot (c_{\ell-1} r_{\ell-1})^*) \tag{3.6}$$

ottenendo per ciascun gruppo 3 stime. Naturalmente la stima del offset di frequenza è ottenuto dall'offset di fase mediante la relazione

$$\Delta F_i = \frac{\Delta\phi}{2\pi T_s}. \tag{3.7}$$

In fine la stima dell'offset di frequenza si ottiene mediando sui valori di stima parziale: tanto più saranno questi valori e maggiore sarà l'accuratezza della stima. In Fig. 3.36 è riportato lo schema a blocchi del dispositivo.

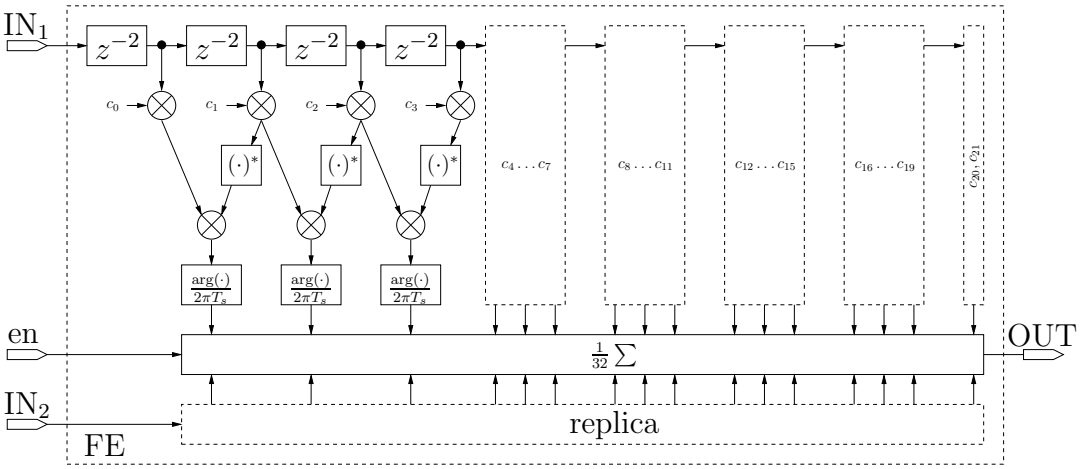


Figura 3.36: Schema a blocchi dello stimatore dell'offset di frequenza.

I ritardi doppi sono frutto del fatto che qui operiamo sui tempi $T_s/2$. Nello schema di rifermento si sono usati 6 blocchi di stima per entrambi i segnali ricevuti sulle due antenne per facilitarne l'implementazione. Si sono usati quindi 5 blocchi con quattro valori e tre stime, mentre il sesto è formato solo dalla variazione di due valori e solo una stima. Questo comporta che la stima viene eseguita mediando su 32 valori. I coefficienti c_ℓ utilizzati in riferimento alle sequenze di Golay descritte dalle Eq. (2.10)–(2.11) sono

$$\{c_0 \dots c_{15}\} = \{+\,-\,+\,+\;\;-\,+\,+\,+\;\;+\,-\,+\,+\;\;-\,+\,+\,+\} \tag{3.8}$$

Il segnale di *enable* ha lo scopo di attivare il dispositivi stimatore, solo quando la stima viene eseguita nell'istante corretto.

Nella fase di implementazione la realizzazione della forma sopra descritta, oltre ad essere di complessità notevole, avrebbe occupato moltissime risorse. Per contrastare l'abuso di slice si è pensato di utilizzare anche in questo caso l'algoritmo Cordic. Al contrario del precedente i confronti vengono eseguiti sui vettori per ricavare il valore dell'argomento. Da questo con una serie di somme algebriche si ottiene il valore finale che verrà successivamente mediato per ottenere il valore adatto ad essere inviato al mixer.

Si ha quindi, prendendo come riferimento lo schema di figura 3.36, che le operazioni eseguite dalla moltiplicazione dei vettori e successivamente da $\frac{\arg(\cdot)}{2\pi T_s}$ sono eseguite da due Cordic che ricavano il valore dell'argomento dei due vettori e li somma algebricamente $ARG1 - ARG2$.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	FE	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	8,565	301,440	3%	
Number of Slice LUTs	21,289	150,720	14%	
Number of occupied Slices	6,478	37,680	17%	
Number of LUT Flip Flop pairs used	21,350			
Number with an unused Flip Flop	14,415	21,350	67%	
Number with an unused LUT	60	21,350	1%	
Number of fully used LUT-FF pairs	6,875	21,350	32%	
Number of unique control sets	186			
Number of slice register sites lost to control set restrictions	964	301,440	1%	
Number of bonded IOBs	143	600	23%	
Number of DSP48E1s	0	768	0%	

Figura 3.37: *Design Summary* del Frequency Estimator con 32 argomenti.

Il risultato dell'implementazione riportato in figura 3.37 è risultato allo stesso modo troppo avaro di risorse a questo proposito dunque abbiamo eseguito le operazioni non su 32 ma solo su 16 argomenti, otto per ogni antenna, dimezzando cosí le slice LUTs utilizzate.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	FE_low	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	4,261	301,440	1%	
Number of Slice LUTs	10,592	150,720	7%	
Number of occupied Slices	3,223	37,680	8%	
Number of LUT Flip Flop pairs used	10,622			
Number with an unused Flip Flop	7,172	10,622	67%	
Number with an unused LUT	30	10,622	1%	
Number of fully used LUT-FF pairs	3,420	10,622	32%	
Number of unique control sets	98			
Number of slice register sites lost to control set restrictions	480	301,440	1%	
Number of bonded IOBs	143	600	23%	
Number of DSP48E1s	0	768	0%	

Figura 3.38: *DesignSummary* del Frequency Estimator con 16 argomenti (FE_low).

Time Estimator

La stima dell'istante di campionamento sfrutta il fatto che i due valori adiacenti al picco di correlazione, situati agli istanti $\pm T_s/2$ rispetto ad esso, in caso di *timing* esatto assumono lo stesso valore mentre se il secondo è maggiore del primo istante di campionamento dobbiamo campionare più avanti e viceversa, se è minore occorre campionare più indietro. Lo schema è illustrato in Fig. 3.39.

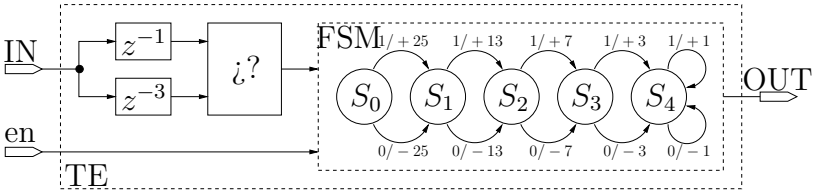


Figura 3.39: Schema a blocchi dello stimatore dell'istante di campionamento.

Rispetto al segnale di *enabling*, che avviene 2 campioni dopo il picco, il confronto va eseguito tra il campione precedente e il successivo. L'aspetto fondamentale di questo blocco è la macchina a stati finiti (FSM) che serve a

velocizzare la convergenza verso il valore corretto di campionamento. Partendo dal presupposto che il rapporto $\frac{T_s/2}{T_c} = 100$ si ha che nel peggiore dei casi il campionamento all'accensione del dispositivo si trovi ad una distanza massima di $50T_c$ dal centro dell'occhio di campionamento. Seguendo una logica di convergenza veloce ossia a passi successivi di lunghezza dimezzata

$$\begin{array}{lcl} S_0 & \rightarrow & \pm 25\ T_c \\ S_1 & \rightarrow & \pm 13\ T_c \\ S_2 & \rightarrow & \pm 7\ T_c \\ S_3 & \rightarrow & \pm 3\ T_c \\ S_4 & \rightarrow & \pm 1\ T_c \end{array} \tag{3.9}$$

e quindi una volta raggiunto l'ultimo stato in 4 passi, la macchina continua ad apportare correzioni di $\pm 1T_c$ per volta. Con questo algoritmo si converge attorno ad un valore ottimale in pochi passi, ed eventuali correzioni fini vengono in seguito aggiunte per mantenere l'ottimalità nel tempo. Anche in questo caso, il segnale di *enable* ha lo scopo di attivare il dispositivo stimatore, solo quando la stima viene eseguita nell'istante corretto.

La macchina a stati finiti è stata poi tradotta in linguaggio VHDL, e ha prodotto il seguente *Design Summary*.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	TE	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	63	301,440	1%	
Number of Slice LUTs	87	150,720	1%	
Number of occupied Slices	24	37,680	1%	
Number of LUT Flip Flop pairs used	87			
Number with an unused Flip Flop	24	87	27%	
Number with an unused LUT	0	87	0%	
Number of fully used LUT-FF pairs	63	87	72%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	1	301,440	1%	
Number of bonded IOBs	36	600	6%	
Number of DSP48E1s	0	768	0%	

Figura 3.40: *DesignSummary* del Time Estimator.

3.2.4 Interpretazione

L'ultima parte che compone il ricevitore è composta da:

- *Campionatore Finale*: campiona i dati in base alla ricezione dell'impulso del picco di correlazione
- *Channel Estimator* : tiene traccia dell'arrivo delle segnalazioni pilota nel flusso e aggiorna istantaneamente la stima di canale.
- *Maximum Ratio Combining* stabilisce con la massima probabilità il vettore relativo ai bit trasmessi.
- *Decoder* dal vettore ricava i bit inviati e ricostruisce il campione.

Inizialmente il blocco di MRC aveva il compito di stabilire anche con massima probabilità i bit trasmessi, ma per avere una versatilità maggiore si è deciso di separarlo in due creando il blocco di MRC e il blocco di decodifica *decoder*. Infatti, se si vuole sostituire la costellazione di mappatura dei bit in trasmissione e/o cambiare il numero di bit del campione, basterà cambiare il blocco di decodifica *decoder*, mettendone uno adatto alle nuove caratteristiche. Si mantengono quindi inalterate le più delicate operazioni di interpretazione e di estrazione del vettore.

Campionatore Finale

Il campionatore finale non fa altro che campionare il flusso di dati solo quando il segnale riceve il picco di correlazione. Riceve i bit dal blocco di acquisizione e solo quando il Peak Detector rileva un picco si aggiorna l'uscita e il valore viene mandato al Maximum Ratio Combining.

Le simulazioni effettuate sul blocco sono riportate in appendice C e il *Design Summary* è riportato in figura 3.41.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	CampF	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	2	301,440	1%	
Number of Slice LUTs	1	150,720	1%	
Number of occupied Slices	2	37,680	1%	
Number of LUT Flip Flop pairs used	2			
Number with an unused Flip Flop	0	10,622	0%	
Number with an unused LUT	1	10,622	50%	
Number of fully used LUT-FF pairs	1	10,622	50%	
Number of unique control sets	2			
Number of slice register sites lost to control set restrictions	14	301,440	1%	
Number of bonded IOBs	227	600	37%	
Number of DSP48E1s	0	768	0%	

Figura 3.41: *Design Summary* del Campionatore Finale.

Channel Estimator

Questo blocco, all'avvio, tiene traccia di quando arrivano le segnalazioni pilota nel flusso di informazione, e si occupa di aggiornare istantaneamente la stima di canale. Ricevuti i quattro simboli relativi alla segnalazione pilota, che in termini matriciali corrispondono a $P_r(n) \simeq \mathbf{H}(nT_p)P$, per ottenere la stima di canale basterà moltiplicare a destra quanto ricevuto per la matrice inversa P^{-1} che abbiamo visto nell'equazione (2.15) ottenendo

$$\hat{\mathbf{H}}(nT_p) = P_r(n)P^{-1}. \tag{3.10}$$

Il blocco implementato è composto da due parti una parte che decodifica i quattro segnali pilota che arrivano, dando l'informazione della stima. L'implementazione è stata realizzata tramite una macchina a stati finiti composta da 3 stati (A,B,C). Nello stato A rimane in attesa di un impulso di enable che identifica l'inizio della segnalazione pilota e ne memorizza la coppia di campioni che arriva. Nello stato B riceve la seconda coppia del segnale pilota. Nello stato C elabora i dati ricevuti e invia in uscita la stima di canale.

La seconda parte che compone il blocco è composta da un contatore che verifica l'arrivo dei pilot, mantenendosi sincronizzato con il segnale provenienti dal *Peak Detector*.

Ogni volta che identifica l'arrivo delle segnalazioni di pilot abilita la macchina a stati. Il segnale a cui viene dato il compito di abilitare la macchina a stati è il segnale chiamato *Enable* che rimane alto per 4 impulsi di clock in modo da permettere la stima del canale.

Oltre a ricavare la stima di canale genera un segnale di inibizione che comanderà i blocchi successivi. Quando il segnale di inibizione sarà portato a livello logico alto all'arrivo dei pilot o del preambolo, si inibiranno le operazioni successive in modo tale da interpretare solo i simboli relativi ai campioni.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	H	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	475	301,440	1%	
Number of Slice LUTs	513	150,720	1%	
Number of occupied Slices	224	37,680	1%	
Number of LUT Flip Flop pairs used	650			
Number with an unused Flip Flop	186	650	28%	
Number with an unused LUT	137	650	21%	
Number of fully used LUT-FF pairs	327	650	50%	
Number of unique control sets	23			
Number of slice register sites lost to control set restrictions	157	301,440	1%	
Number of bonded IOBs	339	600	56%	
Number of DSP48E1s	0	768	0%	

Figura 3.42: *Design Summary* del Channel Estimator.

Maximum Ratio Combining

Il blocco MRC si occupa di stabilire con massima probabilità quale fosse il vettore relativo alla costellazione dei bit trasmessi a partire dai flussi di dati $s_i(n) = s_{\text{opt}}^{(i)}(nT_s)$ e dalla stima del canale fornita $\left[\begin{smallmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{smallmatrix}\right] = \hat{\mathbf{H}}(nT_p)$.

Seguendo lo schema di codifica di Alamouti dell'equazione (2.8), le equazioni di MRC, a partire dai simboli ricevuti in due istanti consecutivi, sono date da

$$\hat{s}(n) = \frac{\sqrt{2}}{\|H\|^2}(h_{11}^* s_1(n) + h_{12} s_1^*(n+1) + h_{21}^* s_2(n) + h_{22} s_2^*(n+1)) \quad (3.11)$$

$$\hat{s}(n+1) = \frac{\sqrt{2}}{\|H\|^2}(h_{11}^* s_1(n+1) - h_{12} s_1^*(n) + h_{21}^* s_2(n+1) - h_{22} s_2^*(n)) \quad (3.12)$$

dove $\|H\|^2 = \sum_{ij} |h_{ij}|^2$.

Quindi il vettore relativo alla costellazione 16APSK è dato da:

$$\begin{bmatrix} \hat{b}(k) \\ \hat{b}(k+1) \end{bmatrix} = \begin{bmatrix} \neg\Re(s(n)s^*(n-1)) \\ \neg\Im(s(n)s^*(n-1)) \end{bmatrix} \quad (3.13)$$

con $k = 2n$ e $\neg(x) = 1(-x)$.

Nell'implementazione le operazioni sopra descritte vengono tradotte e semplificate separando le operazioni relative alla parte reale e immaginaria. Di seguito si riporta il segmento di codice relativo alle operazioni sopra descritte.

```
...
sI(0) <= Hi11*ddiA(0) + Hq11*ddqA(0) + Hi12*ddiA(1) + Hq12*ddqA(1) +
         + Hi21*ddiB(0) + Hq21*ddqB(0) + Hi22*ddiB(1) + Hq22*ddqB(1);

sQ(0) <= Hi11*ddqA(0) - Hq11*ddiA(0) - Hi12*ddqA(1) + Hq12*ddiA(1) +
         + Hi21*ddiB(0) - Hq21*ddqB(0) - Hi22*ddiB(1) + Hq22*ddqB(1);

sI(1) <= Hi11*ddiA(1) + Hq11*ddqA(1) - Hi12*ddiA(0) - Hq12*ddqA(0) +
         + Hi21*ddiB(1) + Hq21*ddqB(1) - Hi22*ddiB(0) - Hq22*ddqB(0);

sQ(1) <= Hi11*ddqA(1) - Hq11*ddiA(1) + Hi12*ddqA(0) - Hq12*ddiA(0) +
         + Hi21*ddiB(1) - Hq21*ddqB(1) + Hi22*ddiB(0) - Hq22*ddqB(0);
...
Re <= sI(1)(55 downto 28) * sI(0)(55 downto 28) +
         + sQ(1)(55 downto 28) * sQ(0)(55 downto 28);

Im <= sQ(1)(55 downto 28) * sI(0)(55 downto 28) -
         - sI(1)(55 downto 28) * sQ(0)(55 downto 28);
...
```

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	MRC	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	1,069	301,440	1%	
Number of Slice LUTs	26,639	150,720	17%	
Number of occupied Slices	6,986	37,680	18%	
Number of LUT Flip Flop pairs used	26,784			
Number with an unused Flip Flop	25,715	26,784	96%	
Number with an unused LUT	145	26,784	1%	
Number of fully used LUT-FF pairs	924	26,784	3%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	7	301,440	1%	
Number of bonded IOBs	450	600	75%	
Number of DSP48E1s	0	768	0%	

Figura 3.43: *Design Summary* del blocco MRC.

Decoder

Il blocco *decoder* verifica la posizione del vettore passato dall'MRC. Con una serie di confronti verifica, con il criterio della minima distanza, a quale valore della costellazione è più vicino decodificando cosí il segnale. All'interno del blocco, inoltre sono presenti due Macchine a Stati Finiti, sincronizzate rispettivamente alla frequenza di 140KHz e 48KHz. Le due Macchine a Stati Finiti attuano la ricostruzione del campione. Esse operano in modo esattamente inverso del blocco *divide* presente nel trasmettitore (ref par. 2.5.1).

La prima Macchina a Stati Finiti, composta da 11 stati, impila su un vettore di 44bit i valori decodificati dal criterio di minima distanza. Mentre la seconda Macchina a Stati Finiti, preleva 11 bit[7] alla volta e completa la ricostruzione del campione.

[7] 10 bit di informazione + 1 di parità

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	decoder	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	173	301,440	1%	
Number of Slice LUTs	252	150,720	1%	
Number of occupied Slices	65	37,680	1%	
Number of LUT Flip Flop pairs used	252			
Number with an unused Flip Flop	135	252	53%	
Number with an unused LUT	0	252	0%	
Number of fully used LUT-FF pairs	117	252	46%	
Number of unique control sets	3			
Number of slice register sites lost to control set restrictions	17	301,440	1%	
Number of bonded IOBs	126	600	21%	
Number of DSP48E1s	0	768	0%	

Figura 3.44: *Design Summary* del blocco Decoder.

Infine si riporta lo schematico finale del ricevitore in cui si vedono oltre ai componenti sopra descritti anche dei sommatori per inviare il valore di frequenza corretta ai vari blocchi di correlazione. Successivamente si mostra il ricevitore vero e proprio con le catene di AGC relative alle due antenne.

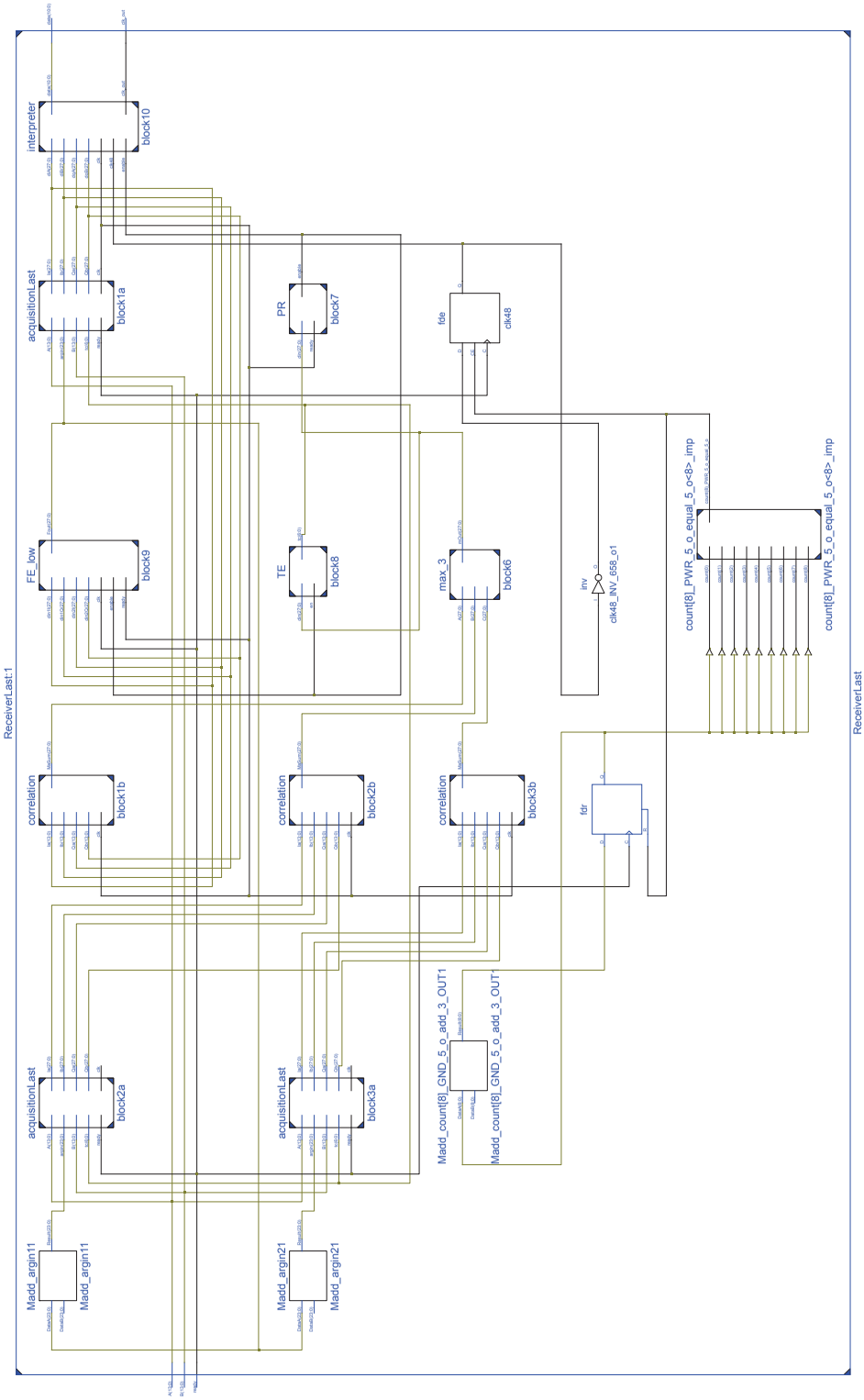


Figura 3.45: Schema a blocchi RTL del package ReductReceiver.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	receiver	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	28,633	301,440	9%	
Number of Slice LUTs	83,585	150,720	55%	
Number of occupied Slices	23,844	37,680	63%	
Number of LUT Flip Flop pairs used	86,651			
Number with an unused Flip Flop	62,472	86,651	63%	
Number with an unused LUT	3,066	86,651	3%	
Number of fully used LUT-FF pairs	21,113	86,651	24%	
Number of unique control sets	153			
Number of slice register sites lost to control set restrictions	749	301,440	1%	
Number of bonded IOBs	41	600	6%	
Number of DSP48E1s	264	768	34%	

Figura 3.46: *Design Summary* del package ReductReceiver.

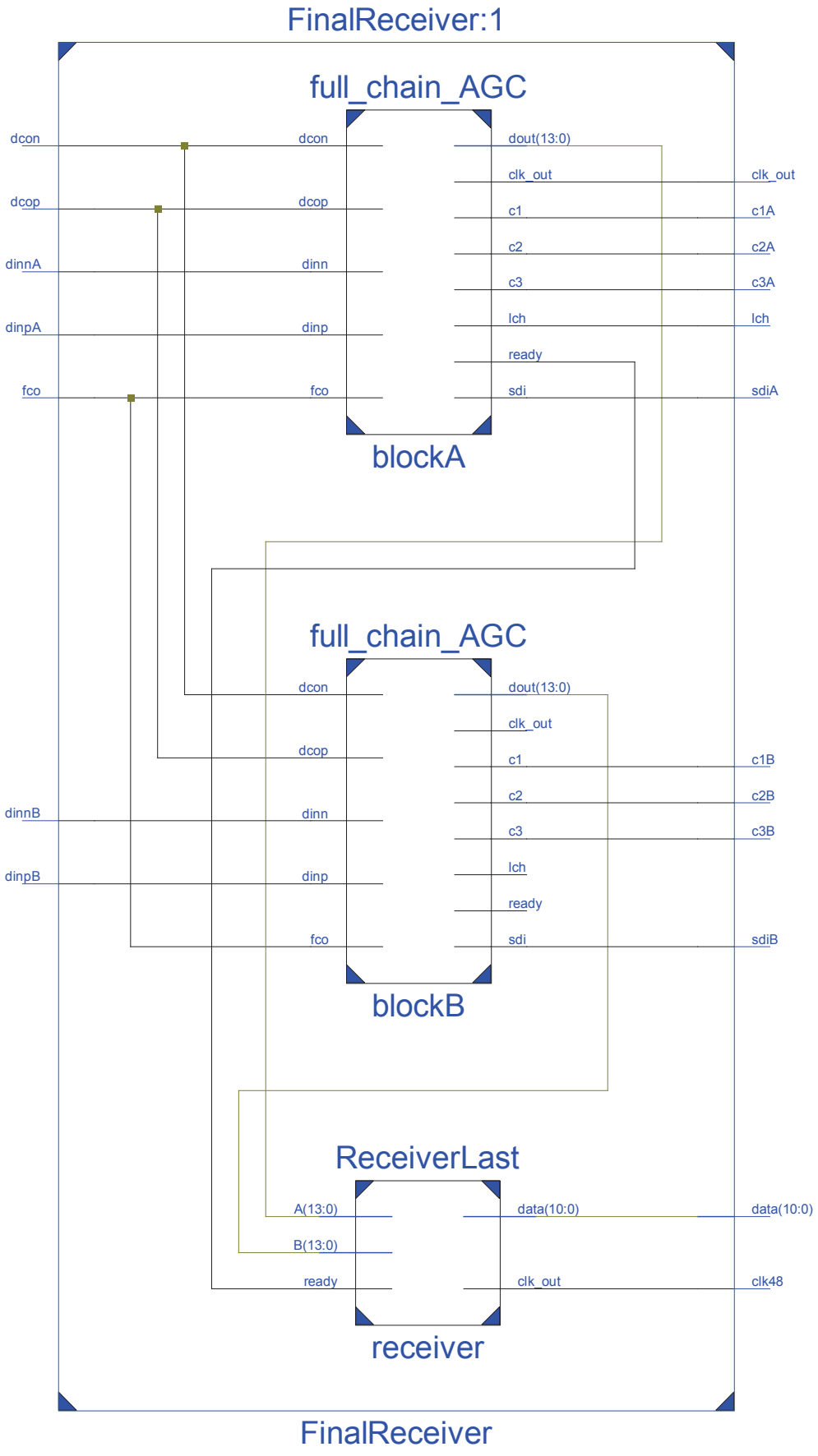


Figura 3.47: Schema a blocchi RTL del ricevitore completo.

ReductReceiver Project Status			
Project File:	ReductReceiver.xise	**Parser Errors:**	No Errors
Module Name:	FinalReceiver	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx240t-1ff1156	**Errors:**	
Product Version:	ISE 13.2	**Warnings:**	
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	28,810	301,440	9%	
Number of Slice LUTs	83,923	150,720	55%	
Number of occupied Slices	24,117	37,680	64%	
Number of LUT Flip Flop pairs used	87,055			
Number with an unused Flip Flop	62,749	87,055	72%	
Number with an unused LUT	3,132	87,055	3%	
Number of fully used LUT-FF pairs	21,174	87,055	24%	
Number of unique control sets	157			
Number of slice register sites lost to control set restrictions	724	301,440	1%	
Number of bonded IOBs	29	600	4%	
Number of DSP48E1s	264	768	34%	

Figura 3.48: *Design Summary* del ricevitore completo.

Conclusioni

In questa tesi si sono realizzati un trasmettitore e un ricevitore per un sistema di ricetrasmissione per radiomicrofoni professionali. La realizzazione finale ha portato alla creazione del trasmettitore e ricevitore analizzati nella trattazione teorica [1]. Come primo obiettivo il sistema doveva avere un ritardo complessivo inferiore ai 2ms.

Dall'analisi simulativa, riportata in appendice C, si vede come il trasmettitore ritardi complessivamente di $300\mu s$. Altri ritardi saranno dovuti alla componentistica esterna. Il ritardo introdotto dalla parte di ricezione è molto più incisivo. Infatti per la sua notevole estensione il ritardo viene calcolato con la somma dei ritardi di ogni blocco, data l'impossibilità di simulare interamente il circuito.

La prima parte che condiziona il ritardo è determinato dalla catena AGC, in cui la risposta viene ritardata solo di un campione dall'arrivo dell'impulso di conclusione del dato dall'ADC.

Successivamente il segnale passa attraverso al blocco di acquisizione formato dal mixer, dallo shifter e dai filtri. Il ritardo complessivo si assesta sui $22\mu s$. Tutti i ritardi sono stati ricavati dalle simulazioni riportate nell'appendice C. Il mixer introduce un ritardo di $35ns$ pari ad un periodo di clock a 28MHz, mentre lo shifter introduce un ritardo di $3.5\mu s$ nel caso peggiore[8]. Il contributo più elevato è da attribuirsi ai filtri, dovuto alle catene di somma e moltiplicazione.

Il segnale successivamente passa attraverso il blocco di interpretazione. Qui il ritardo ingresso/uscita introdotto è pari a $49us$. Il ricevitore, quindi, ritarda complessivamente di $71\mu s$ la trasmissione.

Il blocco Time Estimator è l'unico che interviene sul ritardo del blocco shifter e modifica il ritardo della trasmissione. Gli altri blocchi sono solamente blocchi di retroazione che modificano la risposta dei singoli blocchi senza intervenire sul ritardo. I valori temporali riportati e le simulazioni effettuate sono in ogni caso da ritenersi con il sistema a regime con variazione istantanea dei dati in ingresso.

Complessivamente l'intero sistema di trasmissione ha un ritardo di $371us$ valore condizionato solo dalle catene di filtri di invio e ricezione esterni agli FPGA.

Per la codifica del segnale si sono utilizzati 20 bit che si sono visti ottimali caratterizzando la dinamica del segnale a oltre 120dB. Grazie alla compressione ADPCM pari ad 1/2, alla codifica a 16APSK, e al filtro di Nyquist si è permessa

[8]Ritardo di 100 campioni

la trasmissione senza uscire dalla bandadi trasmissione massima imposta dalla normativa ETSI.

Infine il trasmettitore realizzato occupa 11,899 slice LUTs, permettendone la realizzazione su FPGA di più piccole dimensioni e di famiglie precedenti. Il ricevitore occupando 83,923 slice LUTs e 264 slice DSP48 dedicate restringe di molto la scelta degli FPGA più adatti a questa realizzazione.

In questa tesi non sono stati compiuti test fisici sulla trasmissione e ricezione che si lasceranno a sviluppi futuri. I test fisici suggeriti, oltre alla misura del ritardo di trasmissione, comprenderanno un'analisi dell'impatto sulla distorsione dovuta alla modulazione scelta, sull'effettiva banda occupata dal segnale modulato e sul BER al variare della modulazione, passando da una modulazione 16APSK ad una di tipo differenziale.

In conclusione si riportano le caratteristiche del sistema implementato.

- Numero di bit utili del convertitore ADC: 20bit;
- Dinamica del segnale audio: 120.41 db;
- Bitrate del segnale: 48KHz;
- Compressione dei campioni: ADPCM 1/2;
- Codifica di canale: Parity bit;
- Modulazione: 16APSK;
- Numero antenne in trasmissione: 2;
- Numero antenne in ricezione: 2;
- Ritardo massimo dovuto agli FPGA: $d_{\max} = 371\mu s$;
- Slice LUTs utilizzate al Trasmettitore: 11,899;
- Slice LUTs utilizzate al Ricevitore: 83,923;
- Slice DSP48 utilizzate al Ricevitore: 264;

Appendice A

Informazioni Accessorie

A.1 Termini di fase dell'algoritmo Cordic.

I termini dell'algoritmo Cordic Rotation riferiti al Mixer sono stati creati in modo tale da avere la precisione necessaria ad essere sensibile a incrementi di 50Hz su 10.7MHz. I valori ricavati sono i seguenti.

Gradi	**Valore Binario**
45.00	01000000000000000000
26.56	00100101110010000000 10
14.04	00010100010001000100 10000
7.13	00001010001011000010 00
3.58	00000101000101110101 00
1.79	00000010100010111101 10
0.90	00000001010001011111 00
0.45	00000000101000101111 10
0.22	00000000010100010111 11
0.11	00000000001010001011 11
0.06	00000000000101000101 11
0.03	00000000000010100010 11
0.01	00000000000001010001 01

Tabella A.1: Termini Relativi all'argomento del Cordic Rotation.

I termini relativi all'algoritmo Cordic Vectoring, riferiti al Frequency Estimator, sono stati calcolati come i precedenti per avere un grosso margine di precisione. Con questa definizione si è in grado di rilevare variazioni fino ad 1Hz su 10.7MHz. I valori ricavati sono riportati in tabella A.2.

Gradi	Valore Binario
45.00	00100000000000000000000
26.56	00010010111001000000101
14.04	00001001111110110011100
7.13	00000101000100010001001
3.58	00000010100010110000110
1.79	00000001010001011101011
0.90	00000000101000101111011
0.45	00000000010100010111110
0.22	00000000001010001011111
0.11	00000000000101000101111
0.06	00000000000010100010111
0.03	00000000000001010001011
$14.00 \cdot 10^{-3}$	00000000000000101000101
$7.00 \cdot 10^{-3}$	00000000000000010100010
$3.50 \cdot 10^{-3}$	00000000000000001010001
$1.75 \cdot 10^{-3}$	00000000000000000101000
$0.87 \cdot 10^{-3}$	00000000000000000010100
$0.44 \cdot 10^{-3}$	00000000000000000001010
$0.22 \cdot 10^{-3}$	00000000000000000000101
$0.11 \cdot 10^{-3}$	00000000000000000000010
$0.05 \cdot 10^{-3}$	00000000000000000000001
$0.03 \cdot 10^{-3}$	00000000000000000000000

Tabella A.2: Termini Relativi all'argomento del Cordic Vectoring.

A.2 Coefficienti dei filtri FIR

Di seguito vengono riportati i coefficienti relativi ai filtri FIR di Trasmissione. In particolare nella tabella A.3 sono riportati i coefficienti del filtro da 21 tappi. I coefficienti vengono passati come coefficiente moltiplicativo ai Moltiplicatori che operano le operazioni di elaborazione del segnale.

In tabella A.4 sono riportati i coefficienti del filtro a 44 tappi componente anch'esso del filtro interpolatore di trasmissione.

Coefficiente	**Valore Decimale**	**Valore HEX**
b_0	-4	FFFC
b_1	1	0001
b_2	40	0028
b_3	81	0051
b_4	-6	FFFA
b_5	-280	FEE8
b_6	-459	FE35
b_7	12	000C
b_8	1364	0554
b_9	2966	0B96
b_{10}	3690	0E6A
b_{11}	2966	0B96
b_{12}	1364	0554
b_{13}	12	000C
b_{14}	-459	FE35
b_{15}	-280	FEE8
b_{16}	-6	FFFA
b_{17}	81	0051
b_{18}	40	0028
b_{19}	1	0001
b_{20}	-4	FFFC

Tabella A.3: Coefficienti del Filtro FIR da 21 tappi in trasmissione.

Coefficiente	**Valore Decimale**	**Valore HEX**
b_0	17	0011
b_1	83	0053
b_2	257	0101
b_3	608	0260
b_4	1184	04A0
b_5	1972	07B4
b_6	2863	0B2F
b_7	3654	0E46
b_8	4095	0FFF
b_9	3976	0F88
b_{10}	3223	0C97
b_{11}	1959	07A7
b_{12}	487	01E7
b_{13}	-805	FCDB
b_{14}	-1580	F9D4
b_{15}	-1686	F96A
b_{16}	-1205	FB4B
b_{17}	-411	FE65
b_{18}	349	015D
b_{19}	800	0320
b_{20}	838	0346
b_{21}	543	021F
b_{22}	114	0072
b_{23}	-242	FF0E
b_{24}	-397	FE73
b_{25}	-343	FEA9
b_{26}	-164	FF5C
b_{27}	25	0019
b_{28}	139	008B
b_{29}	151	0097
b_{30}	93	005D
b_{31}	17	0011
b_{32}	-36	FFDC
b_{33}	-50	FFCE
b_{34}	-34	FFDE
b_{35}	-9	FFF7
b_{36}	8	0008
b_{37}	12	000C
b_{38}	8	0008
b_{39}	2	0002
b_{40}	-2	FFFE
b_{41}	-2	FFFE
b_{42}	-1	FFFF
b_{43}	1	0001

Tabella A.4: Coefficienti del Filtro FIR da 44 tappi in trasmissione.

Di seguito vengono riportati i coefficienti relativi ai filtri FIR di Ricezione che si compone di due filtri in cascata il primo da 98 tappi e il secondo da 21 tappi.

Coefficiente	**Valore Decimale**	**Valore HEX**
b_0	-13	FFF3
b_1	-74	FFB6
b_2	-226	FF1E
b_3	-449	FE3F
b_4	-565	FDCB
b_5	-231	FF19
b_6	904	0388
b_7	2907	0B5B
b_8	5350	14E6
b_9	7393	1CE1
b_{10}	8191	1FFF
b_{11}	7393	1CE1
b_{12}	5350	14E6
b_{13}	2907	0B5B
b_{14}	904	0388
b_{15}	-231	FF19
b_{16}	-565	FDCB
b_{17}	-449	FE3F
b_{18}	-226	FF1E
b_{19}	-74	FFB6
b_{20}	-13	FFF3

Tabella A.5: Coefficienti del Filtro FIR da 21 tappi in ricezione.

Coefficiente	Valore Decimale	Valore HEX
b_0	1	000001
b_1	3	000003
b_2	-5	FFFFFB
b_3	-22	FFFFEA
b_4	-6	FFFFFA
b_5	77	00004D
b_6	136	000088
b_7	38	000026
b_8	-197	FFFF3B
b_9	-396	FFFE74
b_{10}	-451	FFFE3D
b_{11}	-377	FFFE87
b_{12}	-189	FFFF43
b_{13}	144	000090
b_{14}	621	00026D
b_{15}	1177	000499
b_{16}	1697	0006A1
b_{17}	2027	0007EB
b_{18}	2011	0007DB
b_{19}	1503	0005DF
b_{20}	432	0001B0
b_{21}	-1188	FFFB5C
b_{22}	-3202	FFF37E
b_{23}	-5331	FFEB2D
b_{24}	-7183	FFE3F1
b_{25}	-8285	FFDFA3
b_{26}	-8195	FFDFFD
b_{27}	-6533	FFE67B
b_{28}	-3150	FFF3B2
b_{29}	1873	000751
b_{30}	8087	001F97
b_{31}	14757	0039A5
b_{32}	20855	005177
b_{33}	25200	006270
b_{34}	26650	00681A
b_{35}	24225	005EA1
b_{36}	17436	00441C
b_{37}	6284	00188C
b_{38}	-8423	FFDF19
b_{39}	-25275	FF9D45
b_{40}	-42152	FF5B58
b_{41}	-56591	FF22F1
b_{42}	-66006	FEFE2A
b_{43}	-68045	FEF633
b_{44}	-61107	FF114D
b_{45}	-44432	FF5270
b_{46}	-18721	FFB6DF
b_{47}	14225	003791
b_{48}	51068	00C77C
b_{49}	87708	01569C
b_{50}	119275	01D1EB
b_{51}	141011	0226D3
b_{52}	148757	024515
b_{53}	139474	0220D2
b_{54}	112086	01B5D6
b_{55}	67227	01069B
b_{56}	8151	001FD7
b_{57}	-60422	FF13FA
b_{58}	-131619	FDFDDD
b_{59}	-198156	FCF9F4
b_{60}	-252109	FC2733
b_{61}	-286540	FBA0B4
b_{62}	-295811	FB7C7D
b_{63}	-276308	FBC8AC
b_{64}	-227277	FC8833
b_{65}	-150068	FDB5CC
b_{66}	-49439	FF3EE1
b_{67}	68769	010CA1
b_{68}	195925	02FD55
b_{69}	324031	04F1BF
b_{70}	443738	06C55A
b_{71}	547987	085C93
b_{72}	629952	099CC0
b_{73}	685984	0A77A0
b_{74}	713766	0AE426
b_{75}	713924	0AE4C4
b_{76}	688728	0A8258
b_{77}	642182	09CC86
b_{78}	579568	08D7F0
b_{79}	506260	07B994
b_{80}	428249	0688D9
b_{81}	350187	0557EB
b_{82}	276784	043930
b_{83}	210654	0336DE
b_{84}	154245	025A85
b_{85}	107917	01A58D
b_{86}	71973	011925
b_{87}	45181	00B07D
b_{88}	26528	0067A0
b_{89}	14193	003771
b_{90}	6752	001A60
b_{91}	2579	000A13
b_{92}	623	00026F
b_{93}	-29	FFFFE3
b_{94}	-24	E8
b_{95}	83	000053
b_{96}	83	000053
b_{97}	28	00001C

Tabella A.6: Coefficienti del Filtro FIR da 98 tappi in ricezione.

A.3 Alberi Gerarchici

Albero gerarchico della distribuzione dei vari file per la generazione del Trasmettitore. In particolare non sono stati ripetuti i vari blocchi che compongono ogni singolo package, ma solamente il blocco richiamato.

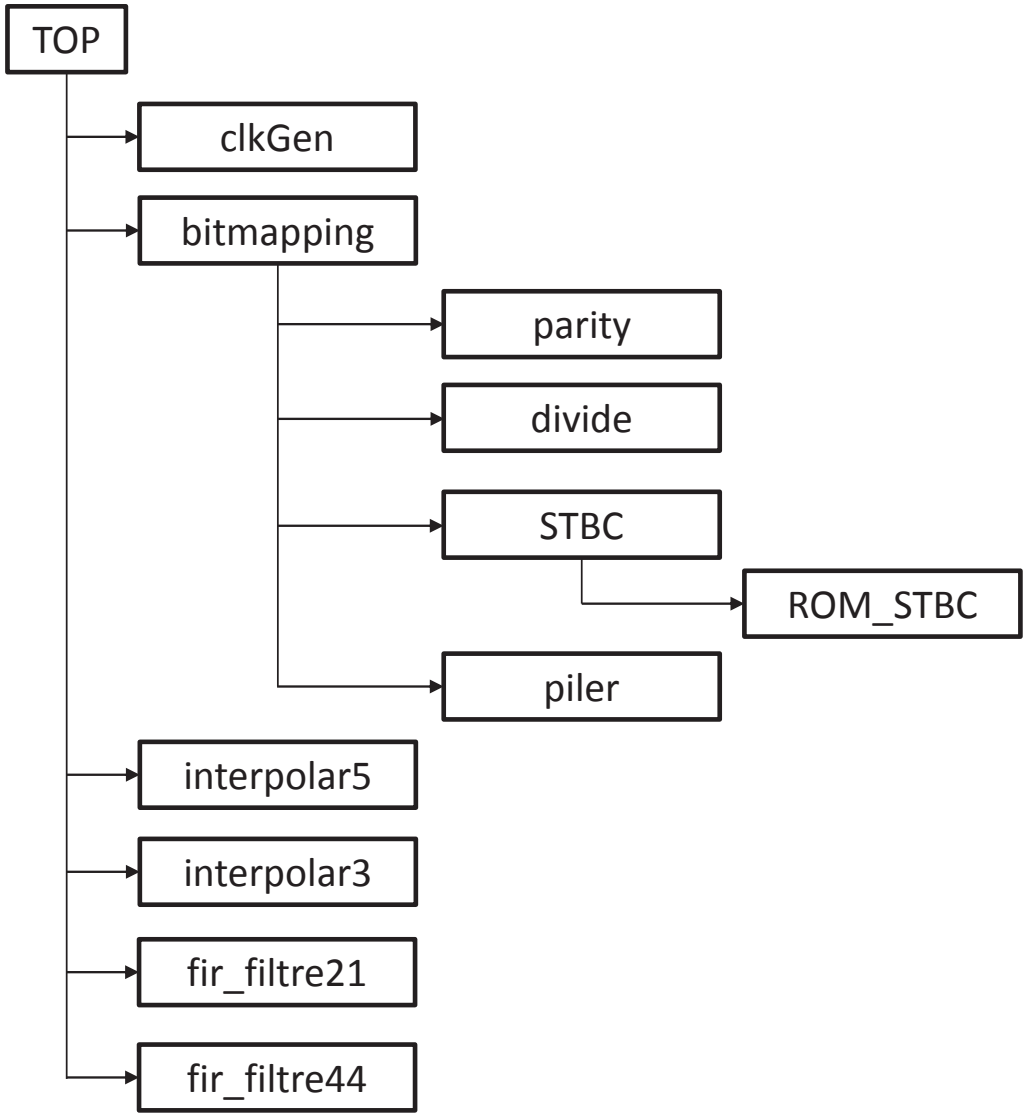


Figura A.1: Gerarchia File del Trasmettitore.

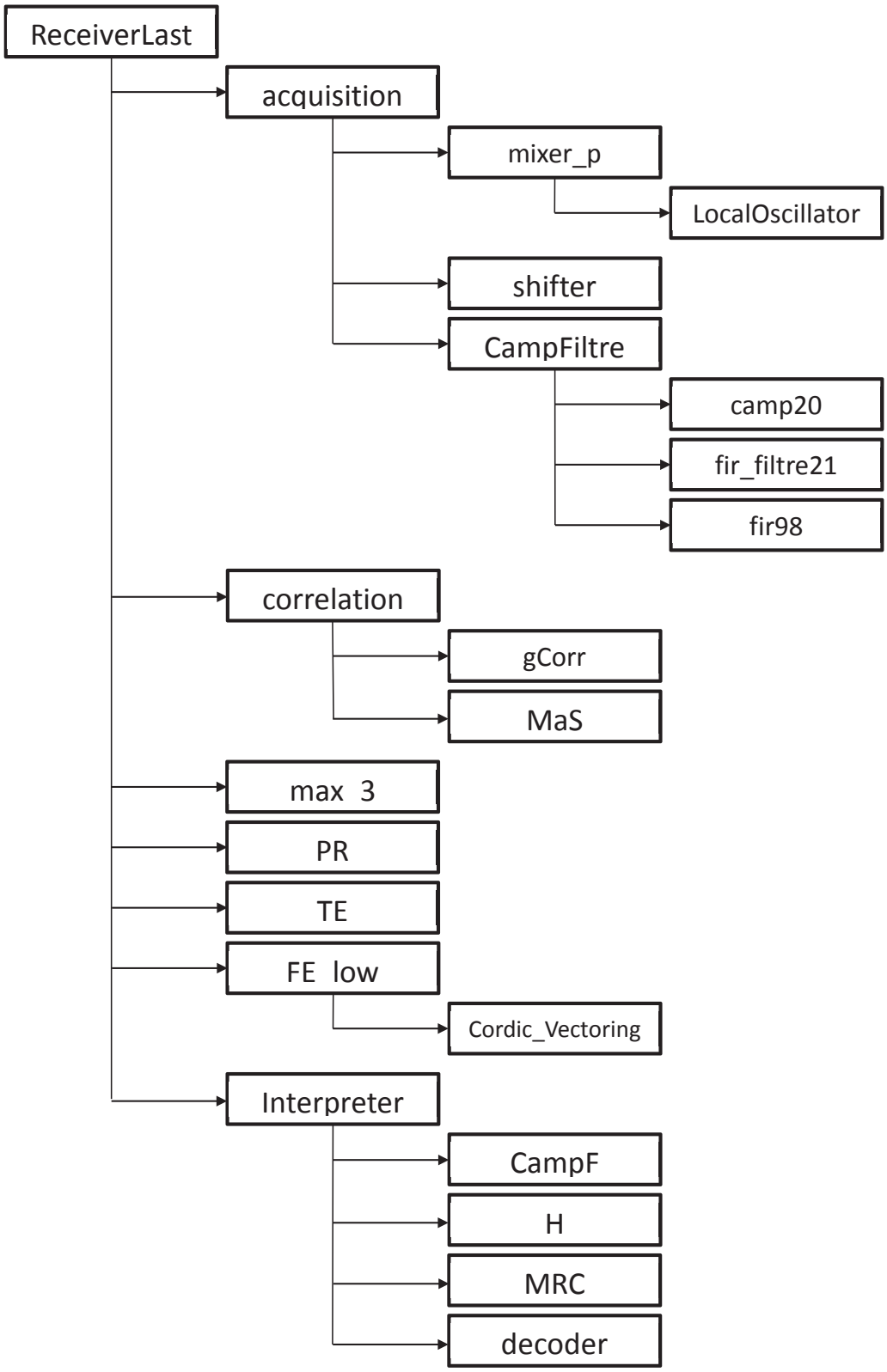


Figura A.2: Gerarchia File del Ricevitore.

A.4 Connettori

Per l'interfacciamento esterno si utilizzerà il connettore FMC HPC presente sulla Virtex6. Di seguito si riportano le connessioni relative al solo ricevitore, essendo il componente più complesso e quindi più sensibile ad eventuali errori di progettazione, ha bisogno di essere verificato. Il ricevitore quindi è connesso nel modo seguente.

Nome	**Pin FPGA**	**Pin FMC**
dinpA	AG22	D11
dinnA	AH22	D12
dinpB	AM18	D14
dinnB	AL18	D15
fco	AP19	D17
dcop	AN27	D20
dcop	AM27	D21
C3A	AC19	G9
C2A	AD19	G10
C1A	AK22	G12
C3B	AJ22	G13
C2B	AM21	G15
C1B	AL21	G16
clk_{out}	AK23	G21
lch	AL24	G24
sdiA	AP27	G24
sdiB	AP26	G25
clk48	K23	H5
$data_{10}$	AC20	H7
$data_9$	AF19	H10
$data_8$	AE19	H11
$data_7$	AK21	H13
$data_6$	AJ21	H14
$data_5$	AM22	H16
$data_4$	AN22	H17
$data_3$	AM23	H19
$data_2$	AL23	H20
$data_1$	AN25	H22
$data_0$	AN24	H23

Tabella A.7: Connessione tra FPGA e FMC HPC della Virtex 6.

Appendice B

Entity VHDL

In questa sezione verrà riportata solamente la parte iniziale del codice VHDL di ogni file dove sono mostrati gli input/output. Questa scelta è derivata dall'impossibilità di riportare interamente il codice VHDL dovuto alla sua enorme estensione.

Entity del trasmettitore.

```
entity TOP is
    Port ( data : in  STD_LOGIC_VECTOR (9 downto 0);
           clk : in  STD_LOGIC;
           xAout : out  STD_LOGIC_VECTOR (13 downto 0);
           yAout : out  STD_LOGIC_VECTOR (13 downto 0);
           xBout : out  STD_LOGIC_VECTOR (13 downto 0);
           yBout : out  STD_LOGIC_VECTOR (13 downto 0);
           sync : out  STD_LOGIC;
           clk_out : out  STD_LOGIC);
end TOP;
```

```
entity clkGen is
    Port ( clk : in  STD_LOGIC;
           clk132 : out  STD_LOGIC;
           clk140 : out  STD_LOGIC;
           clk48 : out  STD_LOGIC;
           clk700 : out  STD_LOGIC;
           clk2100 : out  STD_LOGIC);
end clkGen;
```

```
entity bitmapping is
    Port ( data : in  STD_LOGIC_VECTOR (9 downto 0);
           clk48 : in  STD_LOGIC;
           clk132 : in  STD_LOGIC;
           clk140 : in  STD_LOGIC;
```

```
            xAout : out  STD_LOGIC_VECTOR (6 downto 0);
            yAout : out  STD_LOGIC_VECTOR (6 downto 0);
            xBout : out  STD_LOGIC_VECTOR (6 downto 0);
            yBout : out  STD_LOGIC_VECTOR (6 downto 0));
end bitmapping;

---

entity parity is
    Port ( data : in  STD_LOGIC_VECTOR (9 downto 0);
           clk48 : in  STD_LOGIC;
           dataout : out  STD_LOGIC_VECTOR (10 downto 0));
end parity;

---

entity divide is
    Port ( data : in  STD_LOGIC_VECTOR (10 downto 0);
           clk48 : in  STD_LOGIC;
           clk132 : in  STD_LOGIC;
           value : out  STD_LOGIC_VECTOR (3 downto 0));
end divide;

---

entity STBC is
    Port ( value : in  STD_LOGIC_VECTOR (3 downto 0);
           clk132 : in  STD_LOGIC;
           xA : out  STD_LOGIC_VECTOR (6 downto 0);
           yA : out  STD_LOGIC_VECTOR (6 downto 0);
           xB : out  STD_LOGIC_VECTOR (6 downto 0);
           yB : out  STD_LOGIC_VECTOR (6 downto 0));
end STBC;

---

entity ROM_STBCx is
    Port ( address : in  STD_LOGIC_VECTOR (3 downto 0);
           data : out  STD_LOGIC_VECTOR (6 downto 0));
end ROM_STBCx;

---

entity piler is
    Port ( xA : in  STD_LOGIC_VECTOR (6 downto 0);
           yA : in  STD_LOGIC_VECTOR (6 downto 0);
           xB : in  STD_LOGIC_VECTOR (6 downto 0);
```

```
            yB : in  STD_LOGIC_VECTOR (6 downto 0);
            clk132 : in  STD_LOGIC;
            clk140 : in  STD_LOGIC;
            xAout : out  STD_LOGIC_VECTOR (6 downto 0);
            yAout : out  STD_LOGIC_VECTOR (6 downto 0);
            xBout : out  STD_LOGIC_VECTOR (6 downto 0);
            yBout : out  STD_LOGIC_VECTOR (6 downto 0));
end piler;

---

entity Interpolar5 is
    Port ( din : in  STD_LOGIC_VECTOR (6 downto 0);
           clk700 : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (6 downto 0));
end Interpolar5;

---

entity fir_filtre44 is
    Port ( din : in  STD_LOGIC_VECTOR (6 downto 0);
           ready : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (13 downto 0));
end fir_filtre44;

---

entity Interpolar3 is
    Port ( din : in  STD_LOGIC_VECTOR (6 downto 0);
           clk2100 : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (6 downto 0));
end Interpolar3;

---

entity fir_filtre21 is
    Port ( din : in  STD_LOGIC_VECTOR (6 downto 0);
           ready : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (13 downto 0));
end fir_filtre21;
```

Entity del ricevitore.

```
entity FinalReceiver is
    Port ( dcop : in  STD_LOGIC;
           dcon : in  STD_LOGIC;
           fco : in  STD_LOGIC;

           clk_out : out  STD_LOGIC;
           lch : out  STD_LOGIC;

           dinnA : in  STD_LOGIC;
           dinpA : in  STD_LOGIC;
           c1A : out  STD_LOGIC;
           c2A : out  STD_LOGIC;
           c3A : out  STD_LOGIC;
           sdiA : out  STD_LOGIC;

           dinnB : in  STD_LOGIC;
           dinpB : in  STD_LOGIC;
           c1B : out  STD_LOGIC;
           c2B : out  STD_LOGIC;
           c3B : out  STD_LOGIC;
           sdiB : out  STD_LOGIC;

           data: out  STD_LOGIC_VECTOR (10 downto 0);
           clk48: out STD_LOGIC);
end FinalReceiver;

---

entity full_chain_AGC is
    Port ( dcop : in  STD_LOGIC;
           dcon : in  STD_LOGIC;
           fco : in  STD_LOGIC;
           dinn : in  STD_LOGIC;
           dinp : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (13 downto 0);

           c1 : out  STD_LOGIC;
           c2 : out  STD_LOGIC;
           c3 : out  STD_LOGIC;

           clk_out : out  STD_LOGIC;
           lch : out  STD_LOGIC;
           sdi : out  STD_LOGIC;

           ready : out STD_LOGIC);
```

```
end full_chain_AGC;

---

entity SP is
    Port ( dcop : in  STD_LOGIC;
           dcon : in  STD_LOGIC;
           fco : in  STD_LOGIC;
           dinn : in  STD_LOGIC;
           dinp : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (13 downto 0);
           ready : out STD_LOGIC);
end SP;

---

entity ENV is
    Port ( din : in  STD_LOGIC_VECTOR (13 downto 0);
           ready : in  STD_LOGIC;
           clk1m : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (13 downto 0));
end ENV;

---

entity AGC is
    Port ( din : in  STD_LOGIC_VECTOR (13 downto 0);
           clk1m : in  STD_LOGIC;
           c1 : out  STD_LOGIC;
           c2 : out  STD_LOGIC;
           c3 : out  STD_LOGIC;
           ampout : out  STD_LOGIC_VECTOR (5 downto 0));
end AGC;

---

entity ROM_AGC is
    Port ( address : in  STD_LOGIC_VECTOR (6 downto 0);
           data : out  STD_LOGIC_VECTOR (11 downto 0));
end ROM_AGC;

---

entity AGCtoAD8372 is
    Port ( value : in  STD_LOGIC_VECTOR (5 downto 0);
           ready : in  STD_LOGIC;
           clk_out : out  STD_LOGIC;
```

```
            lch : out  STD_LOGIC;
            sdi : out  STD_LOGIC);
end AGCtoAD8372;

---

entity ReceiverLast is
    Port ( A : in  STD_LOGIC_VECTOR (13 downto 0);
           B : in  STD_LOGIC_VECTOR (13 downto 0);
           ready : in  STD_LOGIC;
           data : out  STD_LOGIC_VECTOR (10 downto 0);
           clk_out: out STD_LOGIC);
end ReceiverLast;

---

entity acquisitionLast is
    Port ( A : in  STD_LOGIC_VECTOR (13 downto 0);
           B : in  STD_LOGIC_VECTOR (13 downto 0);
           ready : in  STD_LOGIC;
           argin : in STD_LOGIC_VECTOR(23 downto 0);
           tc : in STD_LOGIC_VECTOR(6 downto 0);
           Ia : out  STD_LOGIC_VECTOR (27 downto 0);
           Qa : out  STD_LOGIC_VECTOR (27 downto 0);
           Ib : out  STD_LOGIC_VECTOR (27 downto 0);
           Qb : out  STD_LOGIC_VECTOR (27 downto 0);
           clk : out  STD_LOGIC);
end acquisitionLast;

---

entity mixer_p is
    Port ( dA : in  STD_LOGIC_VECTOR (13 downto 0);
           dB : in  STD_LOGIC_VECTOR (13 downto 0);
           argin : in  STD_LOGIC_VECTOR (23 downto 0);
           ready : in  STD_LOGIC;
           dIA : out  STD_LOGIC_VECTOR (27 downto 0);
           dQA : out  STD_LOGIC_VECTOR (27 downto 0);
           dIB : out  STD_LOGIC_VECTOR (27 downto 0);
           dQB : out  STD_LOGIC_VECTOR (27 downto 0));
end mixer_p;

---

entity localOscPipeline is
    Port ( argin : in  STD_LOGIC_VECTOR (23 downto 0);
           ready : in  STD_LOGIC;
```

```
            Re : out  STD_LOGIC_VECTOR (13 downto 0);
            Im : out  STD_LOGIC_VECTOR (13 downto 0));
end localOscPipeline;

---

entity shifter is
    Port ( dI : in  STD_LOGIC_VECTOR (27 downto 0);
           dQ : in  STD_LOGIC_VECTOR (27 downto 0);
           tc : in  STD_LOGIC_VECTOR (6 downto 0);
           ready : in  STD_LOGIC;
           sI : out  STD_LOGIC_VECTOR (27 downto 0);
           sQ : out  STD_LOGIC_VECTOR (27 downto 0));
end shifter;

---

entity CampFiltre is
    Port ( I : in  STD_LOGIC_VECTOR (13 downto 0);
           Q : in  STD_LOGIC_VECTOR (13 downto 0);
           ready : in  STD_LOGIC;
           Iout : out  STD_LOGIC_VECTOR (27 downto 0);
           Qout : out  STD_LOGIC_VECTOR (27 downto 0);
           clk : out  STD_LOGIC);
end CampFiltre;

---

entity fir98 is
    Port ( clk: in std_logic;
           rfd: out std_logic;
           rdy: out std_logic;
           din: in std_logic_vector(13 downto 0);
           dout: out std_logic_vector(27 downto 0));
end fir98;

---

entity fir_filtre21 is
    Port ( din : in  STD_LOGIC_VECTOR (13 downto 0);
           ready : in  STD_LOGIC;
           dout : out  STD_LOGIC_VECTOR (27 downto 0));
end fir_filtre21;

---

entity camp20 is
```

```
    Port ( dI : in  STD_LOGIC_VECTOR (13 downto 0);
           dQ : in  STD_LOGIC_VECTOR (13 downto 0);
           ready : in  STD_LOGIC;
           cI : out  STD_LOGIC_VECTOR (13 downto 0);
           cQ : out  STD_LOGIC_VECTOR (13 downto 0);
           freq : out  STD_LOGIC);
end camp20;

---

entity shifter is
    Port ( dI : in  STD_LOGIC_VECTOR (27 downto 0);
           dQ : in  STD_LOGIC_VECTOR (27 downto 0);
           tc : in  STD_LOGIC_VECTOR (6 downto 0);
           ready : in  STD_LOGIC;
           sI : out  STD_LOGIC_VECTOR (27 downto 0);
           sQ : out  STD_LOGIC_VECTOR (27 downto 0));
end shifter;

---

entity correlation is
    Port ( Ia : in  STD_LOGIC_VECTOR (13 downto 0);
           Qa : in  STD_LOGIC_VECTOR (13 downto 0);
           Ib : in  STD_LOGIC_VECTOR (13 downto 0);
           Qb : in  STD_LOGIC_VECTOR (13 downto 0);
           clk : in  STD_LOGIC;
           MaSum : out  STD_LOGIC_VECTOR (27 downto 0));
end correlation;

---

entity gCorr is
    Port ( dI : in  STD_LOGIC_VECTOR (13 downto 0);
           dQ : in  STD_LOGIC_VECTOR (13 downto 0);
           freq : in  STD_LOGIC;
           Ia : out  STD_LOGIC_VECTOR (13 downto 0);
           Ib : out  STD_LOGIC_VECTOR (13 downto 0);
           Qa : out  STD_LOGIC_VECTOR (13 downto 0);
           Qb : out  STD_LOGIC_VECTOR (13 downto 0));
end gCorr;

---

entity MaS is
    Port ( IaA : in  STD_LOGIC_VECTOR (13 downto 0);
           QaA : in  STD_LOGIC_VECTOR (13 downto 0);
```

```
            IbA : in  STD_LOGIC_VECTOR (13 downto 0);
            QbA : in  STD_LOGIC_VECTOR (13 downto 0);
            IaB : in  STD_LOGIC_VECTOR (13 downto 0);
            QaB : in  STD_LOGIC_VECTOR (13 downto 0);
            IbB : in  STD_LOGIC_VECTOR (13 downto 0);
            QbB : in  STD_LOGIC_VECTOR (13 downto 0);
            MaSum : out  STD_LOGIC_VECTOR (27 downto 0));
end MaS;

---

entity max_3 is
    Port ( A : in  STD_LOGIC_VECTOR (27 downto 0);
           B : in  STD_LOGIC_VECTOR (27 downto 0);
           C : in  STD_LOGIC_VECTOR (27 downto 0);
           mOut : out  STD_LOGIC_VECTOR (27 downto 0));
end max_3;

---

entity PR is
    Port ( din : in  STD_LOGIC_VECTOR (27 downto 0);
           enable : out  STD_LOGIC;
           ready : in STD_LOGIC);
end PR;

---

entity TE is
    Port ( din : in  STD_LOGIC_VECTOR (27 downto 0);
           en : in  STD_LOGIC;
           tc : out  STD_LOGIC_VECTOR (6 downto 0));
end TE;

---

entity FE_low is
    Port ( din1I : in  STD_LOGIC_VECTOR (27 downto 0);
           din1Q : in  STD_LOGIC_VECTOR (27 downto 0);
           din2I : in  STD_LOGIC_VECTOR (27 downto 0);
           din2Q : in  STD_LOGIC_VECTOR (27 downto 0);
           clk : in STD_LOGIC;
           enable : in  STD_LOGIC;
           ready : in STD_LOGIC;
           Fout : out  STD_LOGIC_VECTOR (27 downto 0));
end FE_low;
```

```
---

entity cordic_vectoring_low is
    Port ( re : in  STD_LOGIC_VECTOR (27 downto 0);
           im : in  STD_LOGIC_VECTOR (27 downto 0);
           clk : in  STD_LOGIC;
           arg : out  STD_LOGIC_VECTOR (27 downto 0);
           convert : out STD_LOGIC);
end cordic_vectoring_low;

---

entity ROM_CORDIC is
    Port ( address : in  integer;
           data : out  STD_LOGIC_VECTOR (23 downto 0));
end ROM_CORDIC;

---

entity interpreter is
    Port ( diA : in  STD_LOGIC_VECTOR (27 downto 0);
           dqA : in  STD_LOGIC_VECTOR (27 downto 0);
           diB : in  STD_LOGIC_VECTOR (27 downto 0);
           dqB : in  STD_LOGIC_VECTOR (27 downto 0);
           enable : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           clk48 : in  STD_LOGIC;
           data: out  STD_LOGIC_VECTOR (10 downto 0);
           clk_out: out STD_LOGIC);
end interpreter;

---

entity CampF is
    Port ( diA : in  STD_LOGIC_VECTOR (27 downto 0);
           dqA : in  STD_LOGIC_VECTOR (27 downto 0);
           diB : in  STD_LOGIC_VECTOR (27 downto 0);
           dqB : in  STD_LOGIC_VECTOR (27 downto 0);
           enable : in  STD_LOGIC;
           clk : in STD_LOGIC;
           diAo : out  STD_LOGIC_VECTOR (27 downto 0);
           dqAo : out  STD_LOGIC_VECTOR (27 downto 0);
           diBo : out  STD_LOGIC_VECTOR (27 downto 0);
           dqBo : out  STD_LOGIC_VECTOR (27 downto 0);
           clk_out : out STD_LOGIC);
end CampF;
```

```
---

entity H is
    Port ( diA : in  STD_LOGIC_VECTOR (27 downto 0);
           dqA : in  STD_LOGIC_VECTOR (27 downto 0);
           diB : in  STD_LOGIC_VECTOR (27 downto 0);
           dqB : in  STD_LOGIC_VECTOR (27 downto 0);
           enable : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Hi11 : out  STD_LOGIC_VECTOR (27 downto 0);
           Hq11 : out  STD_LOGIC_VECTOR (27 downto 0);
           Hi12 : out  STD_LOGIC_VECTOR (27 downto 0);
           Hq12 : out  STD_LOGIC_VECTOR (27 downto 0);
           Hi21 : out  STD_LOGIC_VECTOR (27 downto 0);
           Hq21 : out  STD_LOGIC_VECTOR (27 downto 0);
           Hi22 : out  STD_LOGIC_VECTOR (27 downto 0);
           Hq22 : out  STD_LOGIC_VECTOR (27 downto 0);
           inibit : out STD_LOGIC);
end H;

---

entity MRC is
    Port ( diA : in  STD_LOGIC_VECTOR (27 downto 0);
           dqA : in  STD_LOGIC_VECTOR (27 downto 0);
           diB : in  STD_LOGIC_VECTOR (27 downto 0);
           dqB : in  STD_LOGIC_VECTOR (27 downto 0);
           clk : in  STD_LOGIC;
           Hi11 : in  STD_LOGIC_VECTOR (27 downto 0);
           Hq11 : in  STD_LOGIC_VECTOR (27 downto 0);
           Hi12 : in  STD_LOGIC_VECTOR (27 downto 0);
           Hq12 : in  STD_LOGIC_VECTOR (27 downto 0);
           Hi21 : in  STD_LOGIC_VECTOR (27 downto 0);
           Hq21 : in  STD_LOGIC_VECTOR (27 downto 0);
           Hi22 : in  STD_LOGIC_VECTOR (27 downto 0);
           Hq22 : in  STD_LOGIC_VECTOR (27 downto 0);
           inibit : in STD_LOGIC;
           Re : out  STD_LOGIC_VECTOR (55 downto 0);
           Im : out  STD_LOGIC_VECTOR (55 downto 0));
end MRC;

---

entity decoder is
    Port ( clk : in  STD_LOGIC;
           clk48 : in  STD_LOGIC;
           Re : in  STD_LOGIC_VECTOR (55 downto 0);
```

```
            Im : in  STD_LOGIC_VECTOR (55 downto 0);
            inibit : in STD_LOGIC;
            data: out  STD_LOGIC_VECTOR (10 downto 0));
end decoder;
```

Appendice C

Simulazioni

In questa appendice si riportano le simulazioni effettuate per la determinazione del tempo di ritardo introdotto dalla sintesi del codice VHDL.

Nelle figure seguenti vengono riportate le simulazioni relative al trasmettitore. Si mostrano in particolare il frame completo l'inserimento dei pilot nella trasmissione e il ritardo complessivo del Trasmettitore.

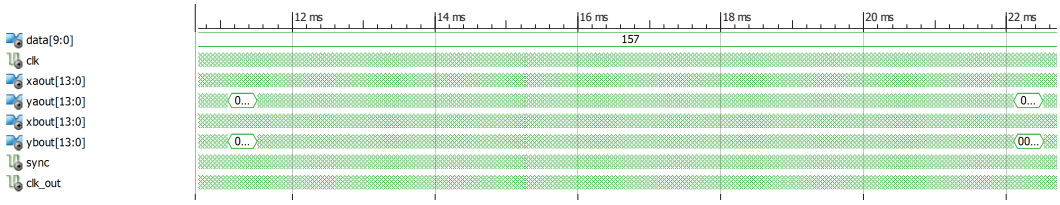


Figura C.1: Simulazione del blocco Bitmapping: Frame completo.

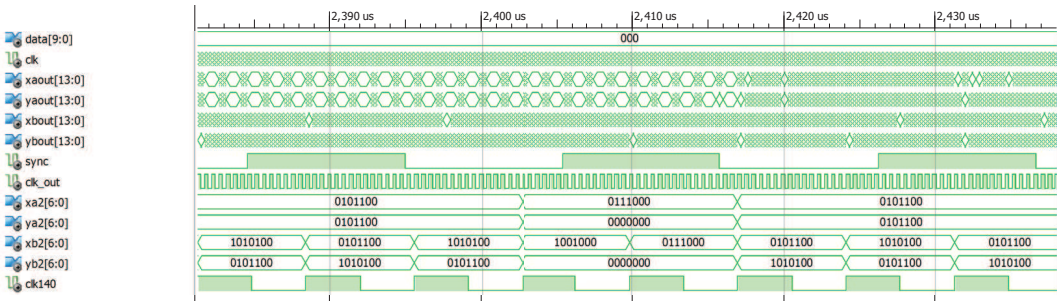


Figura C.2: Simulazione del blocco Bitmapping: Inserimento del Pilot.

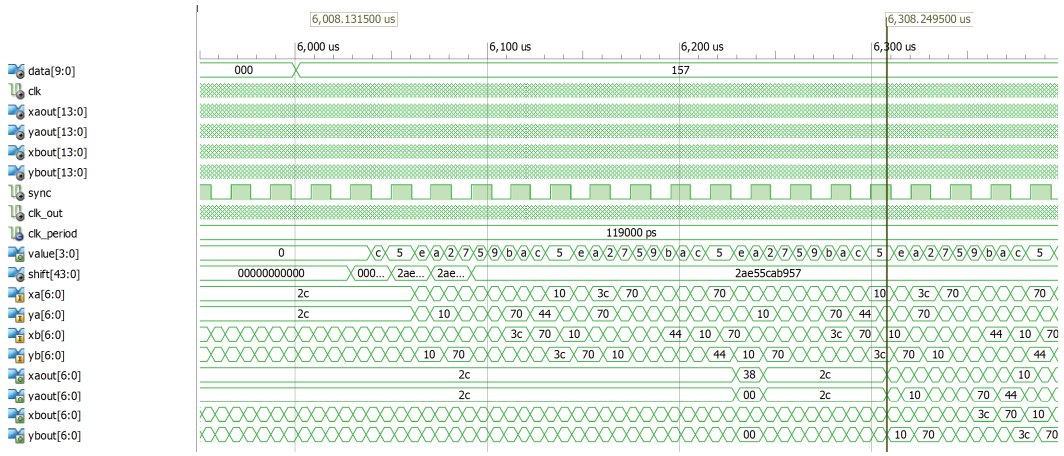


Figura C.3: Simulazione del blocco Trasmettitore: Ritardo di trasmissione.

Di seguito vengono riportati i ritardi relativi al ricevitore. In particolare si riportano i ritardi relativi al blocco di Acquisizione e di Interpretazione che sono gli unici che introducono ritardo ingresso/uscita al segnale, gli altri sono solo blocchi che permettono la sincronizzazione.

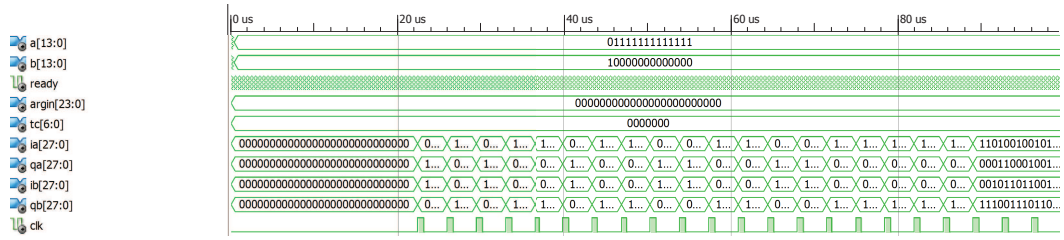


Figura C.4: Simulazione del blocco Acquisition: Risposta ad un impulso.

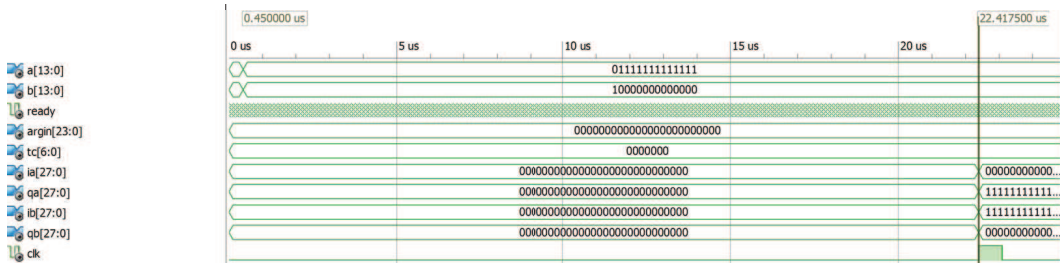


Figura C.5: Simulazione del blocco Acquisition: Ritardo di risposta al gradino.

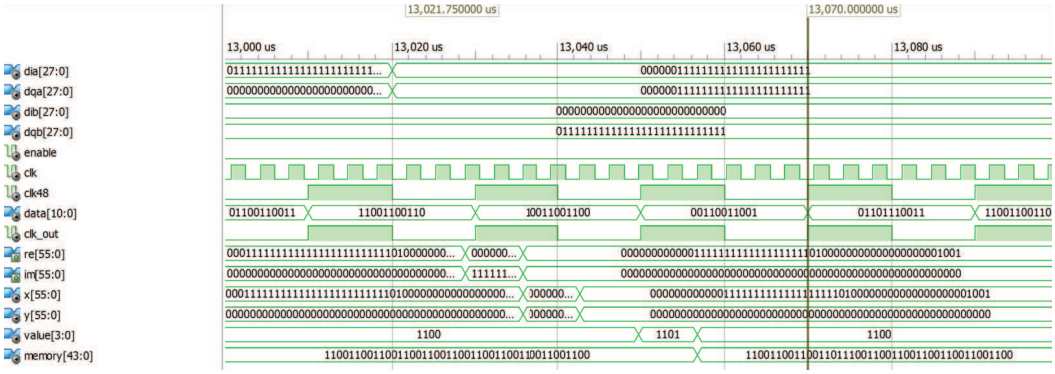


Figura C.6: Simulazione del blocco Interpreter: Ritardo di decodifica del gradino.

.

Bibliografia

[1] A.Vigato "Sistemi di Ricetrasmissione per Radiomicrofoni Professionali", Mar. 2011.

[2] Fernando J. Alvareza, Alvaro Hernandezb, Jesus Urenab, Manuel Mazob, J. Jesus Garcyab, J. Antonio Jimenezb, Ana Jimenezb, "Real-time implementation of an efficient correlator for complementary sets of four sequences applied to ultrasonic pulse compression systems", Microprocessors and Microsystems 30 (2006) pp. 43—51, May 2005.

[3] ETSI, "Electromagnetic compatibility and Radio spectrum Matters (ERM); Wireless microphones in the 25 MHz to 3 GHz frequency range; Part 1: Technical characteristics and methods of measurement" ETSI EN 300 422-1 V1.3.2 (2008–03), Mar. 2008.

[4] ETSI, "Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications", ETSI EN 302 307 V1.1.2 (2006–06), June 2006.

[5] S. M. Alamouti, "A simple transmit diversity technique for wireless communications," *IEEE Trans. on Commun.*, vol. 16, pp. 1451–1458, Oct. 1998.

[6] J.–C. Belfiore, G. Rekaya, and E.Viterbo, "The golden code: a 2×2 full–rate space–time code with non–vanishing determinants," *IEEE Trans. Info. Theory*, vol. 51, pp. 1432–1436, Apr. 2005.

[7] M.J.E. Golay, "Complementary series," *IRE Trans. Inf. Theory*, IT-7, pp. 82–87, 1961.

[8] B.M. Popovic, "Efficient Golay correlator," *Electron. Lett.*, vol. 35, no. 17, pp. 1427–1428, Aug. 1999.

[9] V. Lomi, G. Pierobon, D. Tonetto, and L. Vangelista, "Improved initial synchronisation in the presence of frequency offset in UMTS FDD mode," *Lecture Notes in Computer Science*, vol. 2345, pp. 1165–1171, May 2002.

[10] N. Benvenuto and G. Cherubini, *Algorithms for Communications Systems and their Applications,* Chichester: Wiley, 2002.

[11] S. Wang and A. Abdi, "Aperiodic complementary sets of sequences–based MIMO frequency selective channel estimation," *IEEE Commun. Lett.*, vol. 9, no. 10, pp. 891–893, Oct. 2005.