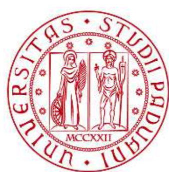


DIPARTIMENTO
DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

"PROGETTO E IMPLEMENTAZIONE VLSI DI UN FILTRO INTERPOLATORE IN TECNOLOGIA 22nm"

Relatore: Prof. Daniele Vogrig

Laureando: Stefano Brunello

ANNO ACCADEMICO 2022 – 2023

Data di laurea 03-04-2023

INTRODUZIONE

La progettazione di un chip è un processo molto complesso e richiede l'impegno di diverse figure professionali: ci sarà chi è specializzato nell'ambito digitale, chi nell'ambito analogico, chi nelle radiofrequenze e così via. Definire una buona divisione dei compiti è fondamentale per orientare il lavoro e per ottenere i risultati desiderati nei tempi previsti. A tal proposito l'architettura interna del chip deve essere suddivisa in aree tematiche in modo che, una volta definite le caratteristiche, possano essere progettate da uno o da un gruppo ristretto di persone. Una volta che tutti avranno ultimato la loro parte si potrà andare ad assemblare il prodotto finale. In questa tesi si tratterà appunto della progettazione di un filtro interpolatore che fornirà i dati in ingresso ad un trasmettitore.[1] Gli scopi principali del progetto sono due:

- **Ottimizzare il consumo di risorse:** come avremo modo di vedere l'utilizzo di un filtro interpolatore è molto più conveniente rispetto ad una memoria.
- **Semplificare la fase di test:** l'interfaccia con cui andremo ad inserire i dati sarà progettata in modo da garantire un maggior controllo in fase di test.

La necessità di utilizzare un filtro interpolatore nasce dalla volontà di alimentare il filtro ad una frequenza inferiore di quella a cui lui legge gli ingressi. Il filtro interpolatore leggerà i dati in ingresso ad una frequenza di 1.7 MHz mentre in uscita avranno una frequenza di 150 Mhz, con un sovracampionamento complessivo di 128. Nel primo capitolo andremo ad analizzare analiticamente vari tipi di filtri interpolatori così da scegliere quello più adatto ad essere implementato. Oltre ad una analisi puramente teorica saranno fatte anche delle simulazioni in Matlab® così da emulare le dinamiche interne dei dati nei circuiti digitali. Nel secondo capitolo procederemo con la progettazione in VHDL del circuito finale e delle sue componenti: il filtro interpolatore e la memoria. Per il filtro interpolatore svilupperemo due modelli e solo in seguito andremo a scegliere uno dei due. Sebbene servino solo per il test, le componenti saranno progettate con delle funzioni addizionali in modo che possano essere riutilizzate successivamente in altri progetti. Il capitolo successivo sarà dedicato all'analisi dei dati ottenuti dalle simulazioni delle componenti. Nello specifico sarà investigato in maniera approfondita il comportamento dei due filtri mentre per le altre il test sarà più rapido. Nel quarto capitolo andremo a trasporre i modelli teorici finora elaborati in qualcosa di più concreto tramite Design Vision di Synopsys.[5] In questa fase ottenendo delle stime circa i consumi di risorse dei due filtri opereremo la scelta del modello definitivo. Inoltre, in base ai report sulle caratteristiche della memoria decideremo di riprogettarla usando delle componenti di libreria. Quindi nel quinto capitolo analizzeremo le librerie messe a disposizione per la tecnologia a ventidue nanometri net tentativo di trovare una memoria alternativa. Vi sarà anche un primo paragrafo dedicato a descrivere brevemente le caratteristiche delle standard cell che andremo ad impiegare nella fase finale. Sebbene i risultati della ricerca siano stati buoni individuando una memoria idonea, la sintesi finale sarà fatta con quella sviluppata precedentemente in quanto non siamo riusciti ad ottenerla. Nel sesto capitolo si procederà a descrivere passo per passo il processo di Place and Route del in Innovus™.

L’utilizzo di una tecnologia piuttosto recente a 22 nm comporterà molteplici complicazioni visto il difficile settaggio del compilatore. Per offrire una soluzione la più valida possibile saranno riportati i comandi utilizzati in ogni fase così da poter essere replicati anche su progetti differenti. Con questo ultimo step potremo finire la nostra analisi essendo arrivati ad un progetto definitivo del chip pronto per essere unito al trasmettitore.

INDICE

1 Progettazione del filtro interpolare 7
1.1 Come funziona un filtro interpolatore 7
1.2 Scelta del tipo di filtro interpolatore 9
1.3 Simulazioni in Matlab® 13
1.3.1 I dati in ingresso 13
1.3.2 Simulazione del filtro interpolatore ideale 14
1.3.3 Simulazione del filtro interpolatore a dati quantizzati 16
2 Realizzazione in VHDL 19
2.1 Modello finale del Chip 19
2.2 Progettazione dei Filtri in VHDL 22
2.2.1 Modello filtro interpolatore 16-8 23
2.2.2 Modello filtro interpolatore 16-2-2-2 24
2.2.3 Formato e significato dei dati in ingresso 25
2.2.4 Design degli stadi 26
2.2.5 Modello del primo stadio 27
2.2.6 Gestione e generazione dei clock interni 28
2.2.7 Modello dei filtri del primo stadio 29
2.2.8 Modello dei filtri del secondo stadio 16-8 31
2.2.9 Modello del secondo, terzo e quarto stadio 16-2-2-2 32
2.2.10 Confronto tra i modelli 16-8 e 16-2-2-2 33
2.3 Progettazione Memoria in VHDL 34
2.3.1 Progettazione memoria ciclica 36
2.3.2 Progettazione registro 37
3 Simulazioni dei modelli VHDL 39
3.1 Simulazione del filtro interpolatore 40
3.1.1 Simulazione a Bassa Risoluzione 40
3.1.2 Simulazione a Media Risoluzione 41
3.1.3 Simulazione ad Alta Risoluzione 42
3.1.4 Confronto tra le varie Simulazioni 43
3.1.5 Test segnali di Reset ed Enable 45
3.2 Simulazione della memoria 47
3.3 Simulazione del design finale 48
4 Progettazione su Design Vision 49
4.1 Setup utilizzato per Design Vision 51
4.2 Calcolo preliminare dell'area e scelta del modello 52
4.3 Attribuzione dei vincoli 53

4.3.1 Vincoli utilizzati nel Filtro 16-2-2-2 55
4.3.2 Vincoli utilizzati nella memoria 56
4.3.3 Vincoli utilizzati nel design finale 57
4.4 Report 58
4.4.1 Report del Filtro interpolatore 60
4.4.2 Report della Memoria 63
4.4.3 Report del design finale 66
4.5 Conclusioni sui report 69
5 Memoria alternativa: scelta ed analisi 71
5.1 Librerie e kit disponibili 71
5.2 Memorie 74
5.3 Specifiche tecniche della memoria 77
5.4 Dual-Port SRAM (SPDV): Analisi del modello 79
5.5 Progettazione della logica di controllo 83
5.6 Conclusioni sulla memoria alternativa 86
6 Place and Route con Innovus™ 87
6.1 Setup di Innovus ed esportazione del Design 89
6.2 Floorplan 92
6.2 I/O Planning 93
6.4 Power Planning 94
6.5 Ottimizzazioni Pre-Placement 97
6.5.1 Ottimizzazione del compilatore 97
6.5.2 Celle di Welltap 98
6.5.3 Celle di Endcap 99
6.6 Placement 101
6.7 Clock Tree Synthesis 102
6.7 Ottimizzazioni Pre-Route 103
6.7.1 Celle di Filler 103
6.7.2 Ottimizzazioni NanoRoute 104
6.8 Route 108
7 Considerazioni finali 111
Bibliografia 113

1 Progettazione del filtro interpolare

Nella prima fase andremo a identificare uno o più possibili implementazioni del filtro interpolatore. Come sappiamo dall'introduzione uno dei compiti principali del circuito è quello di aumentare la frequenza di un dato segnale, compito che può essere fatto solo attraverso l'interpolazione dei dati mediate un filtro interpolatore. Inoltre, una accurata progettazione del filtro fatta matematicamente e tramite simulazioni in Matlab, è fondamentale per garantirne le buone prestazioni una volta implementato in VHDL. Sarebbe piuttosto spiacevole dover riprogettare la logica qualora il filtro non avesse il comportamento atteso. Quindi oltre all'identificazione dei modelli verranno anche fatti dei test circa il loro funzionamento così da certificare le prestazioni in vista della fase successiva.

1.1 Come funziona un filtro interpolatore

Nel filtro Interpolatore la ricostruzione del segnale ad una frequenza aumentata si raggiunge attraverso due operazioni in sequenza:

Zero-Stuffing: consiste nell'inserimento di un numero di zeri pari al fattore di sovra-campionamento(L) meno uno, tra due campioni successivi. Il segnale così ottenuto avrà una dimensione L volte superiore rispetto al segnale di partenza. Facendo un'analisi in frequenza si potrà notare come la frequenza massima raggiungibile sia di L volte maggiore, mentre lo spettro risulta duplicato a multipli della frequenza di ingresso. [3]

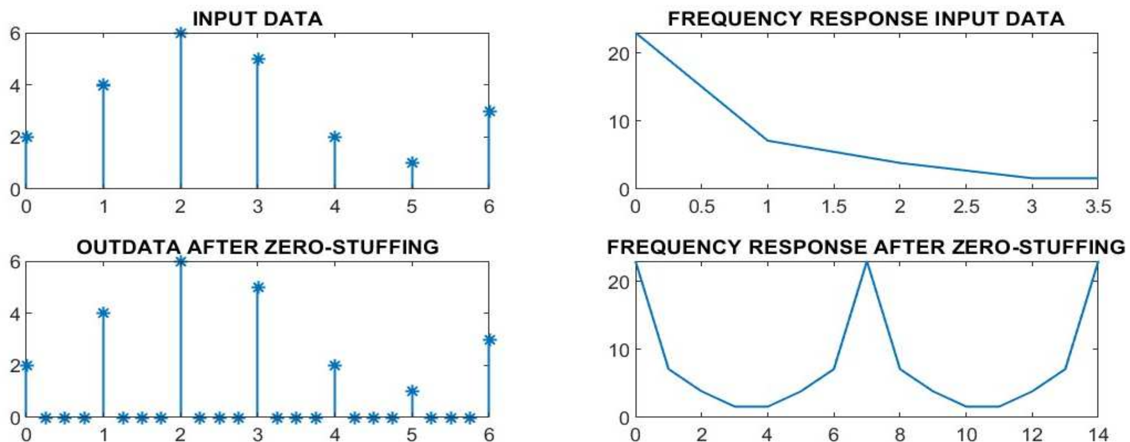


Figura 1: Effetti dello zero-stuffing in frequenza e nel tempo (frequenza di ingresso 7Hz)

Filtraggio: Per rimuovere i multipli dello spettro bisognerà necessariamente usare un filtro passa-basso cercando di filtrare i multipli e lasciare solo il contenuto in banda base. Sfortunatamente un filtro perfettamente rettangolare in frequenza è impossibile da realizzare, la scelta ricade dunque su un filtro a coseno rialzato che ha una risposta impulsiva nulla negli istanti Nt, annullando l'interferenza di intersimbolo (ISI). Ovviamente la complessità del filtro sarà correlata alle sue prestazioni, più noi vorremo avere un filtro prestazionale con una forte attenuazione delle copie maggiore sarà il numero di coefficienti del filtro. Anche la precisione dei dati con i quali andremo a rappresentare questi numeri può risultare importante, banalmente il coseno rialzato ha

un comportamento tendente a 0 per numeri che tendono ad infinito, quindi per usare molti fattori avrò bisogno di numeri in grado di rappresentarli efficacemente. Il suo comportamento in frequenza può essere descritto dalla seguente formula dove il fattore β detto roll-off (di rotolamento), al massimo può essere uguale a 1, permette di modificarne le caratteristiche: migliora la forma del filtro all'interno della banda ma diventa meno filtrante rispetto alle copie. [3]

$$H(f) = \begin{cases} T, & |f| \leq \frac{1-\beta}{2T} \\ \frac{T}{2}[1 + \cos\left(\frac{\pi T}{\beta}\left(|f| - \frac{1-\beta}{2T}\right)\right), & \frac{1-\beta}{2T} < |f| \leq \frac{1+\beta}{2T} \\ 0, & altrove \end{cases}$$

La sua risposta impulsiva è:

$$h(t) = \begin{cases} \frac{\pi}{4} sinc\left(\frac{t}{2\beta}\right), & t \pm \frac{T}{2\beta} \\ \frac{sinc\left(\frac{T}{t}\right) * \cos\left(\frac{\pi\beta t}{T}\right)}{1 - \left(\frac{2\beta t}{T}\right)^2} \quad , & altrove \end{cases}$$

Per meglio apprezzare gli effetti del fattore di rotolamento ecco un grafico per confrontare i due:

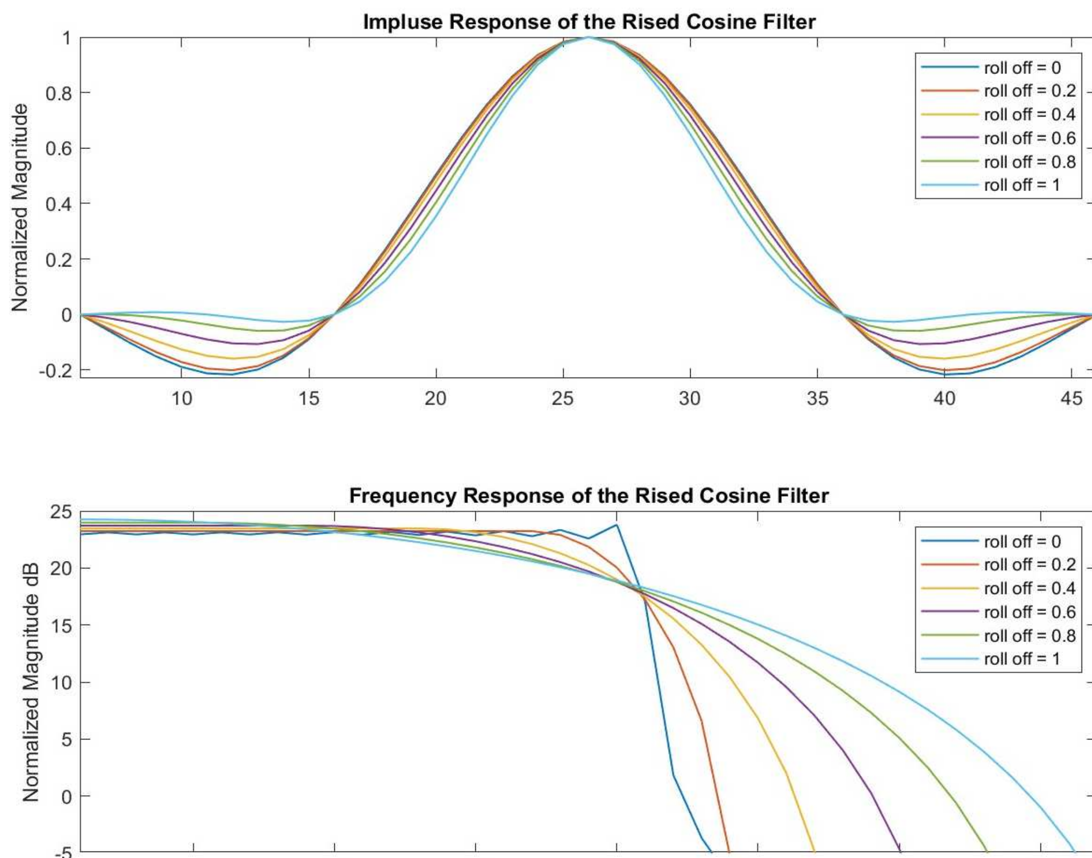


Figura 2: Risposta impulsiva e in frequenza del filtro a coseno rialzato con diversi roll-off

Come si può osservare dalla figura 2 il fattore di rotolamento ha come effetto principale quello di aumentare l'ampiezza del filtro fino ad un massimo di 2 volte la banda base mentre ne linearizza il comportamento nella stessa. Dalla risposta impulsiva si evince come il cambio del comportamento in frequenza si ottenga attraverso l'attenuazione dei lobi; nel caso ideale per avere un filtro rettangolare dovrei tener in considerazione un numero infinito di lobi mente nei casi concreti ne userò un numero inferiore. In conclusione, il fattore di rotolamento è un espediente che ci permette di utilizzare un numero minore di lobi e quindi filtri più piccoli mantenendo comunque le caratteristiche principali.

1.2 Scelta del tipo di filtro interpolatore

La realizzazione di un filtro interpolatore a stadio singolo, ovvero dove lo zero-stuffing viene fatto tutto in una singola volta risulta efficiente per fattori di sovra campionamento non troppo elevati in quanto la complessità del filtro a coseno rialzato associato può risultare eccessiva. Nel nostro caso ho fatto una simulazione con la funzione di Matlab *designMultirateFIR* ottenendo un filtro con ben 1016 coefficienti, numero ampliamente proibitivo per il nostro tipo di applicazione.[2] La migliore soluzione risulta dunque quella di creare un filtro interpolatore multistadio dove suddividendo appropriatamente lo zero-stuffing in due o più stadi si riesca ad ottenere singoli filtri meno complessi. La prima soluzione che mi è venuta in mente è quella di dividerlo in due stadi uno da 16 e uno da 8. Fortunatamente questo tipo di filtri sono già stati ampiamente studiati e Matlab mette a disposizione diversi tool per la loro realizzazione in maniera automatica. Nel mio caso ho usato la funzione *designMultistageInterpolator* che progetta direttamente l'intero filtro interpolatore multistadio impostando cinque semplici parametri che sono:

- Fattore di sovra campionamento complessivo: 128
- Frequenza di uscita: 150MHz
- Attenuazione di stopband: 65dB
- Lunghezza di banda unilaterale: 100kH
- Numero di stadi: 2

Il filtro così progettato aveva le seguenti caratteristiche:

- Coefficienti primo stadio: 73
- Coefficienti secondo stadio: 31
- Coefficienti complessivi: 104

Il numero di coefficienti si riferisce ai coefficienti totali in quanto diversi coefficienti in entrambi i filtri sono uguali a zero. Questa caratteristica permette di passare inalterato il valore in ingresso: gli zeri sono disposti rispetto all'uno centrale in modo che ogni dato non nullo in ingresso si troverà a passare inalterato dal filtro. La proprietà è garantita dal tipo di segnale che ho in ingresso, che ricordiamo è un segnale che ha

subito uno zero-stuffing. Da notare come il primo stadio a fattore 16 abbia bisogno di più del doppio dei coefficienti rispetto al secondo con un fattore 8. Ecco alcune figure per spiegare meglio la situazione:

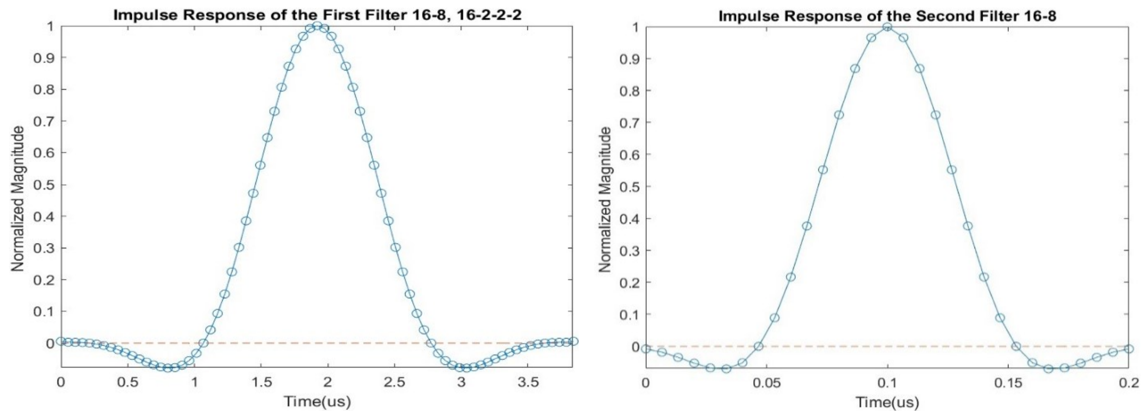


Figura 3: Risposta impulsiva primo e secondo stadio filtro interpolatore 16-8

La prima cosa evidente che si può notare è che nel primo caso a sovra campionamento maggiore i punti della figura sono più fitti rispetto alla seconda. Questo è dovuto al fatto che in ingresso tra un campione non nullo e il successivo passano ben 15 zeri; quindi, visto che il coseno rialzato ha degli zeri fissati bisogna prendere 15 coefficienti tra essi in modo che quanto un campione venga moltiplicato per uno (il valore centrale) gli altri vengano annullati dagli zeri. Nel secondo caso essendoci solo 7 zeri avrò bisogno di 7 coefficienti rendendo la figura visivamente più sfitta. Si può notare inoltre come la forma della risposta impulsiva sia pressoché la stessa ma sulla prima si mantiene un numero di coefficienti associati ai lobi laterali maggiore rispetto alla seconda, circa un lobo e mezzo nella prima e solo uno nella seconda. Questo è il motivo principale della discrepanza tra il numero di coefficienti nei due filtri: per filtri più grandi si mantiene una componente dei lobi laterali maggiore che porta ad avere più del doppio dei coefficienti. Di seguito si può apprezzare il suo comportamento in frequenza:

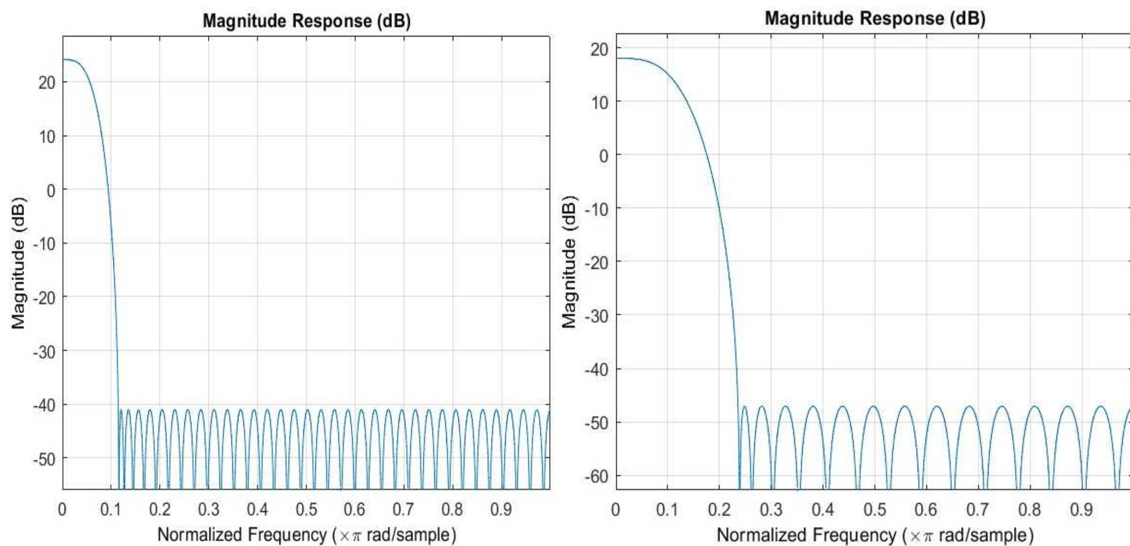


Figura 4: Risposta in frequenza primo e secondo stadio del filtro interpolatore 16-8

In queste figure si può apprezzare come l'attenuazione minima dei coefficienti sia minimo di 65 dB come voluto nelle specifiche del filtro. Il limite dei 65 dB è stato scelto perché per valori più alti il tool non riusciva più a trovare una soluzione soddisfacente nel modello a due stadi 16-8. Per avere attenuazioni maggiori si dovrà dunque scegliere un design differente.

Nel tentativo di trovare un nuovo modello più economico rispetto a quello 16-8 ho implementato un modello a quattro stadi 16-2-2-2 con i seguenti risultati:

- Coefficienti primo stadio: 73 *(invariato)*
- Coefficienti secondo stadio: 7
- Coefficienti terzo stadio: 7
- Coefficienti quarto stadio: 7
- Coefficienti complessivi: 94

In questo caso il primo filtro a fattore 16 risulta invariato mentre il secondo che era a fattore 8 viene suddiviso in tre stadi da 2. La scelta risulta vantaggiosa riducendo il numero di coefficienti complessivo di 10, il risparmio è addirittura maggiore se consideriamo solo i coefficienti non nulli. Per quanto riguarda la risposta impulsiva e in frequenza dei nuovi stadi è molto simile tra di loro quindi ne lascio solo una per confrontarla con quella precedente. [2]

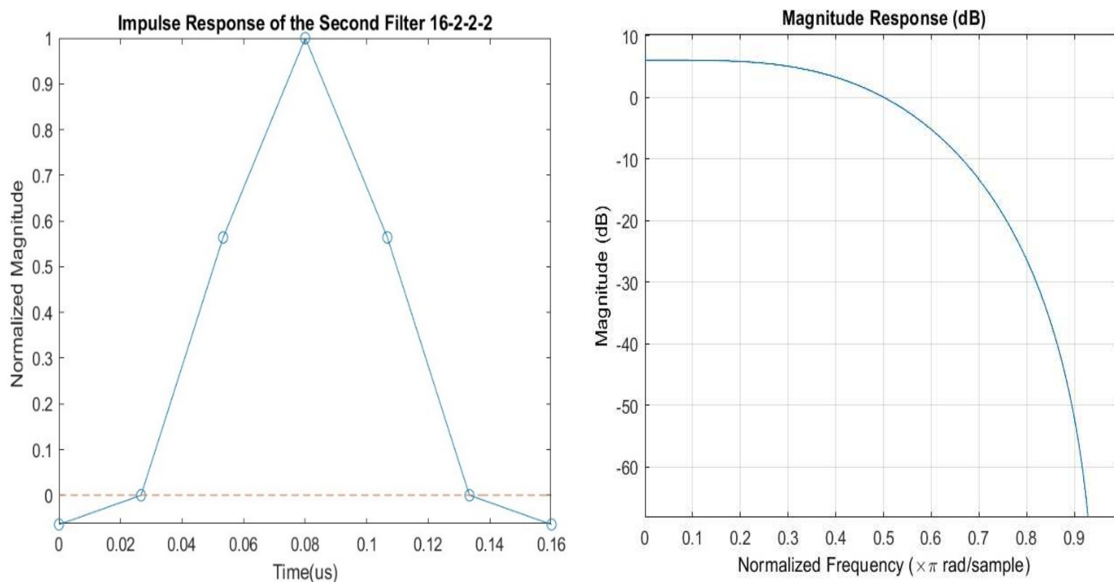


Figura 5: Risposta impulsiva e in frequenza del secondo stadio del filtro interpolatore 16-2-2-2

Mettendo ora a confronto le tre soluzioni ottenute si ottiene il seguente grafico:

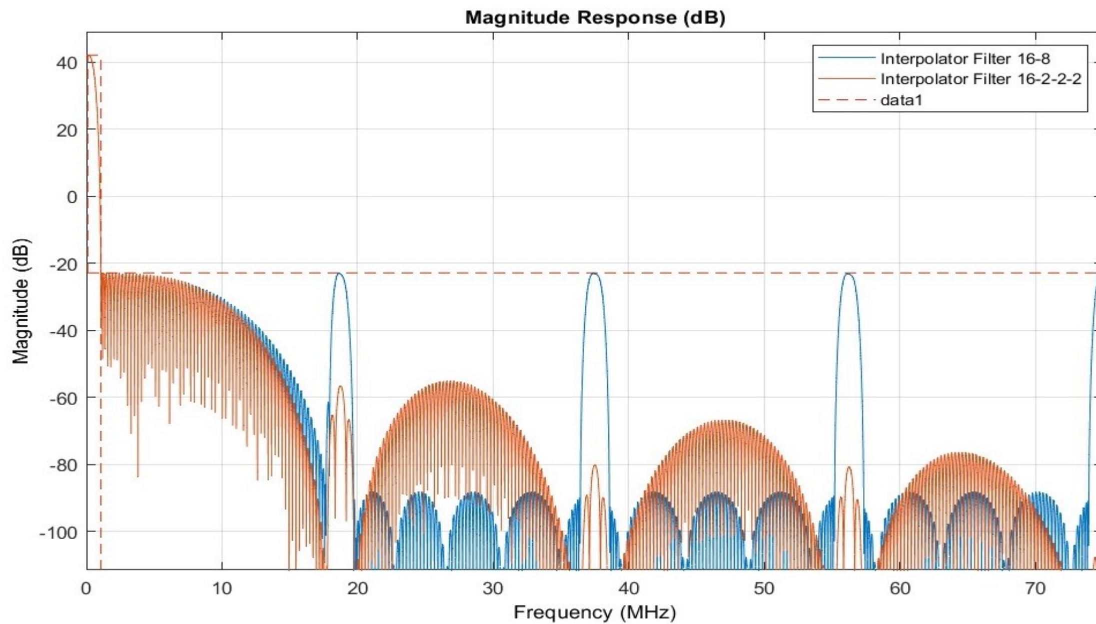


Figura 6: confronto tra filtro interpolatore 16-8 e 16-2-2-2

Complessivamente le due soluzioni soddisfano bene le specifiche, anche se il 16-8 attenuerà di meno in alta frequenza rispetto al 16-2-2-2 da come si può facilmente notare in figura. Dal punto di vista implementativo usare il 16-2-2-2 si tradurrà probabilmente in un risparmio di area per il minore utilizzo dei moltiplicatori ma potrebbe produrre un aumento di logica perché avrò bisogno di più clock per gestire tutti gli stadi. Inoltre, visto che il formato dei dati è a virgola fissa il fatto di avere più stadi può comportare delle problematiche in merito alla propagazione degli errori che possono amplificarsi nel passaggio da uno stadio al successivo.

La scelta migliore è dunque quella di continuare per entrambe le strade e aspettare di testare i comportamenti dei due filtri prima di prendere una decisione su quale filtro andare a implementare.

1.3 Simulazioni in Matlab®

Per simulare il comportamento dei due filtri ho semplicemente riprodotto il processo di zero-stuffing e filtraggio visto al punto precedente. Lo zero stuffing viene riprodotto creando un secondo array di zeri di lunghezza multipla rispetto all'array iniziale del fattore di sovra campionamento desiderato, andando poi a riportare i dati del primo nel secondo debitamente spaziati. Il codice che ho usato per compiere questa operazione è:

```
X1=zeros (1, Oversamplig_factor*length(X));
X1([1: Oversapling_factor: end]) = X;
```

L'array che si produce, X1 in questo caso verrà poi filtrato usando la funzione:

```
filter(Numerator,Denominator,X1);
```

Utilizzando un filtro a coseno rialzato il denominatore sarà uguale a 1 mentre i coefficienti visti precedentemente rappresenteranno il Numeratore. Questo processo può essere applicato a tutti gli stadi variando di volta in volta il fattore di sovra campionamento e i coefficienti del filtro. I dati prodotti da uno stadio finiranno dunque nel successivo dove verranno applicate le stesse operazioni fino a raggiungimento dello stadio finale dove i dati verranno analizzati.[2]

1.3.1 I dati in ingresso

I dati di ingresso devono rappresentare una situazione reale quindi per rappresentarli ho usato una funzione randomica creando dati nell'intervallo evidenziato successivamente per la 16QAM ovvero l'intervallo dei numeri dispari da -15 a 15. La funzione che genera questo codice è:

```
X=round(15*rand(1,1024))*2-15;
```

Il segnale creato sarà dunque a media nulla visto che anche nei casi reali tutti i simboli della costellazione hanno la stessa probabilità di essere chiamati. [2] Ecco un esempio dello spettro del segnale generato:

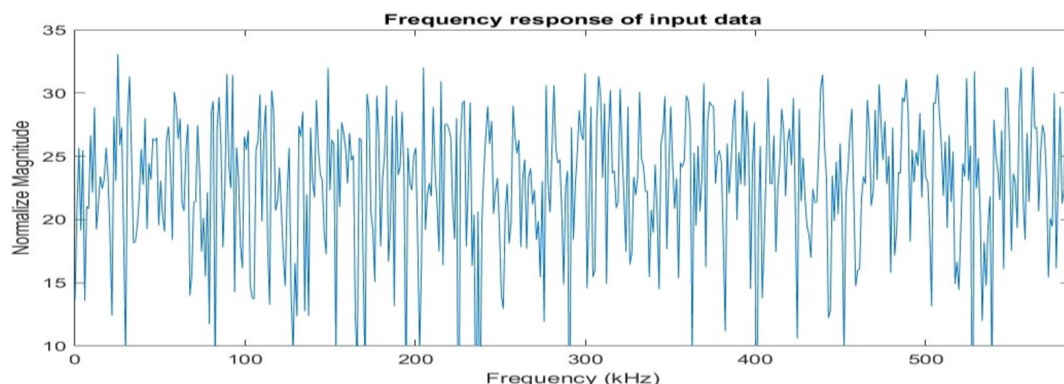


Figura 7: risposta in frequenza dei dati di ingresso

Come si vede in figura il segnale così generato sebbene randomico avrà comunque delle componenti più o meno alte in un raggio di circa 20 dB. Per valutare l'efficacia del nostro filtro sarà dunque necessario prendere come riferimento la componente a frequenza maggiore, se prendessimo come riferimento il valor medio non sarebbe corretto perché la minima attenuazione garantita dal filtro è di 65 dB quindi con molta probabilità alcune componenti non rispetterebbero tale limite. Lo spettro di uscita risulterà anche più alto visto che il filtro presenta un guadagno di circa 42 dB, il valore atteso massimo delle componenti in banda base sarà di circa 75 dB. [2]

1.3.2 Simulazione del filtro interpolatore ideale

Come prima implementazione di questi filtri ho usato il caso più ideale possibile con i coefficienti e i dati in doppia precisione, formato standard di Matlab. La scelta è stata voluta per testare facilmente le caratteristiche del filtro senza tenere in considerazione altri fattori di non idealità come la quantizzazione dei dati e dei coefficienti e i vari troncamenti interni che affronteremo in fasi successive.

La risposta in frequenza finale dei dati è stata:

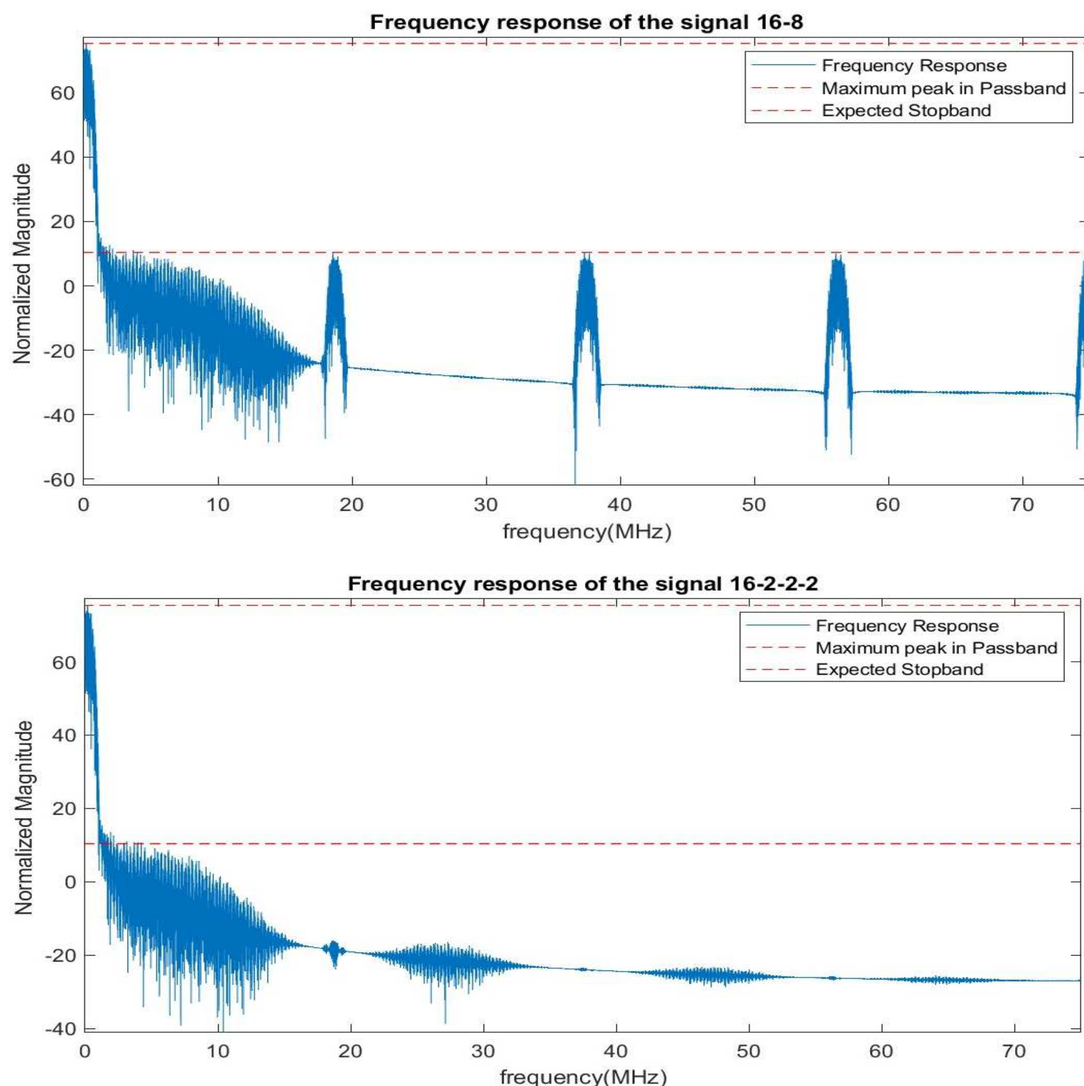


Figura 8: risposta in frequenza dei segnali di uscita filtri 16-8 e 16-2-2-2

Questi primi grafici dimostrano che l'implementazione dei filtri è corretta e che i dati prodotti in uscita mantengono un buon rapporto segnale rumore. L'attenuazione di 65 dB viene mantenuta per la maggior parte delle frequenze, anche se a frequenze basse, da 1 Mhz a 4 Mhz, il valore è leggermente minore segno che anche in un caso ad alta precisione ci sono comunque delle variazioni rispetto ai valori stimati. Nel modello 16-8 si può osservare come la presenza di immagini, in questo caso sono quelle prodotte dal secondo filtro con un sovra campionamento di 8, vengano attenuate quel tanto che basta per mantenere il limite dei 65 dB. Nel caso 16-2-2-2 le immagini vengono addirittura cancellate totalmente dal filtro andando ben oltre i limiti minimi. In conclusione, si può dire che i due modelli rispecchiano abbastanza bene le previsioni fatte a figura 6. [2]

Per ulteriore prova del comportamento del filtro andiamo a studiare i dati di uscita in relazione con i dati di ingresso:

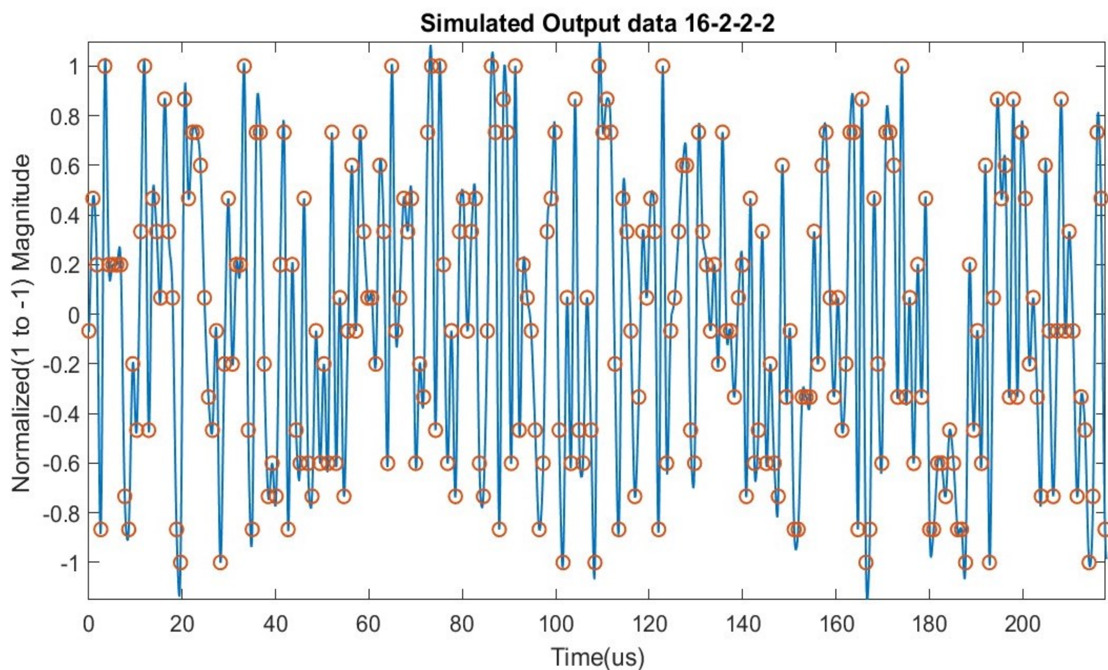


Figura 9: Confronto dati di ingresso vs dati di uscita

I puntini rossi rappresentano i dati in ingresso a frequenza bassa mentre i dati in blu uniti da linea continua rappresentano quelli in uscita dal filtro interpolatore. In questo caso ho usato solo i dati prodotti dal filtro 16-2-2-2 visto che le differenze con quello del 16-8 sono minimamente percettibili. I risultati sono buoni evidenziando come i dati in uscita vadano a sovrapporsi perfettamente a quelli d'ingresso. Si può inoltre osservare anche delle sovra elongazioni quando si hanno due dati di simile valore vicini tra di loro, comportamento congruo con la sovrapposizione delle risposte impulsive viste al punto precedente, in particolare con quella del primo filtro. Visti i risultati soddisfacenti si può procedere con una simulazione dove si prendano in considerazione anche fattori di non idealità.

1.3.3 Simulazione del filtro interpolatore a dati quantizzati

In questa simulazione si vuole prendere in considerazione quali siano gli effetti della quantizzazione dei coefficienti e dei dati interni sull'uscita del nostro filtro interpolatore. La dimensione dei dati è molto importante da considerare per poter stimare il consumo di area; per aggiungere un solo bit di risoluzione alla mia logica combinatoria si dovrà andare a riconsiderare tutti i registri, i sommatori e i moltiplicatori connessi a tali dati incrementando notevolmente il consumo di area. Da qui la necessità di:

- Simulare il comportamento usando diversi bit di precisione
- Creare un modello VHDL facilmente scalabile

La seconda verrà implementata successivamente in fase di progettazione mediante l'utilizzo di variabili e altre strategie. Nel nostro caso si è deciso di tenere il seguente modello per la precisione dei dati nei due filtri 16-8 e 16-2-2-2:

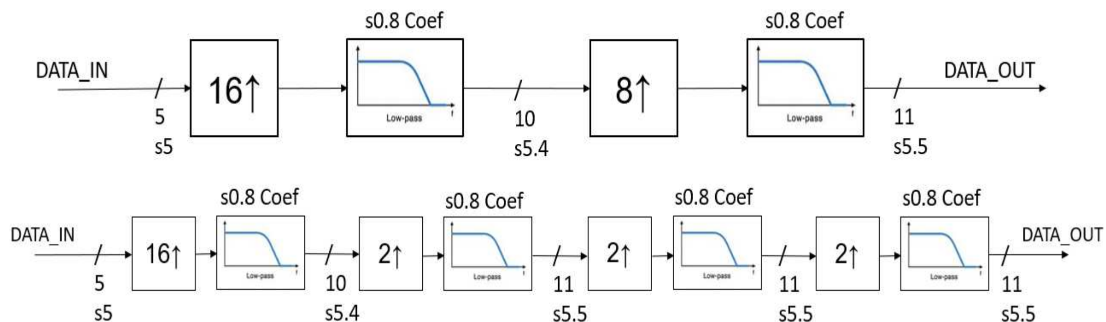


Figura 10: modello filtri 16-8 e 16-2-2-2 con precisione dei dati

I dati in ingresso ai filtri saranno in formato 5 bit con segno, essendo da -15 a 15, mentre quelli dei coefficienti dei filtri saranno a 9 bit nel formato s0.8. La precisione dei coefficienti è cruciale perché determina la precisione degli zeri nella risposta in frequenza: con una bassa precisione si rischia di avere bassi livelli di attenuazione. Gli altri dati in uscita dai filtri vengono poi scelti in relazione a tale parametro secondo la logica di garantire: un bit in più rispetto ai coefficienti dopo il primo filtro per gestire eventuali overflow e poi un ulteriore bit dopo il secondo stadio per cercare di limitare il rumore in alta frequenza. Per questo motivo si effettuerà un troncamento variabile di filtro in filtro, come dimostrato nel capitolo successivo.

Nel modello Matlab i sommatori e moltiplicatori interni sono a precisione massima in quanto è molto difficile agire internamente alla funzione *filter* descritta precedentemente e usata per compiere l'operazione di filtraggio.[2] Per avere un modello che tenga conto anche delle moltiplicazioni in virgola fissa bisognerebbe usare tool più sofisticati come Simulink ma a questo punto risulta più semplice passare direttamente alla implementazione su VHDL. I dati usati nel modello sono a precisione maggiore (16 bit), non esistono su Matlab variabili a cinque o dieci bit ma solo i formati canonici: otto, sedici, trentadue. La questione in questo caso può essere aggirata andando a troncare i bit in eccesso e utilizzando solo parte dei bit messi a disposizione.

Per esempio, i dati uscenti dal primo filtro sono nel formato s5.9, per portarli al formato desiderato sarà sufficiente troncarli al formato s5.4 usando il seguente codice:

```
X_filter1=filter(b2,1,X)*2^-4;
X_filter1_8=int16(X_filter1);
```

La funzione *filter* restituisce valori in formato doppia precisione, da -2^13 a +2^13, il formato in virgola mobile è solo una nostra rappresentazione. [2] Basterà moltiplicare questi valori per il fattore 2^-4 e poi trasformarli in un formato intero a 16 bit per ritornare al formato desiderato. Nel caso dei filtri successivi usando questa stessa operazione si potrà troncare le cifre meno significative del fattore desiderato senza andare ad incrementare quelle più significative visto e considerato che i calcoli all'interno della funzione *filter* vengono effettuati in notazione decimale quindi il bit di segno non viene considerato nei calcoli. Il processo seppur un po' complicato nella sua logica ci permette di arrivare ai seguenti grafici in frequenza per i segnali in uscita:

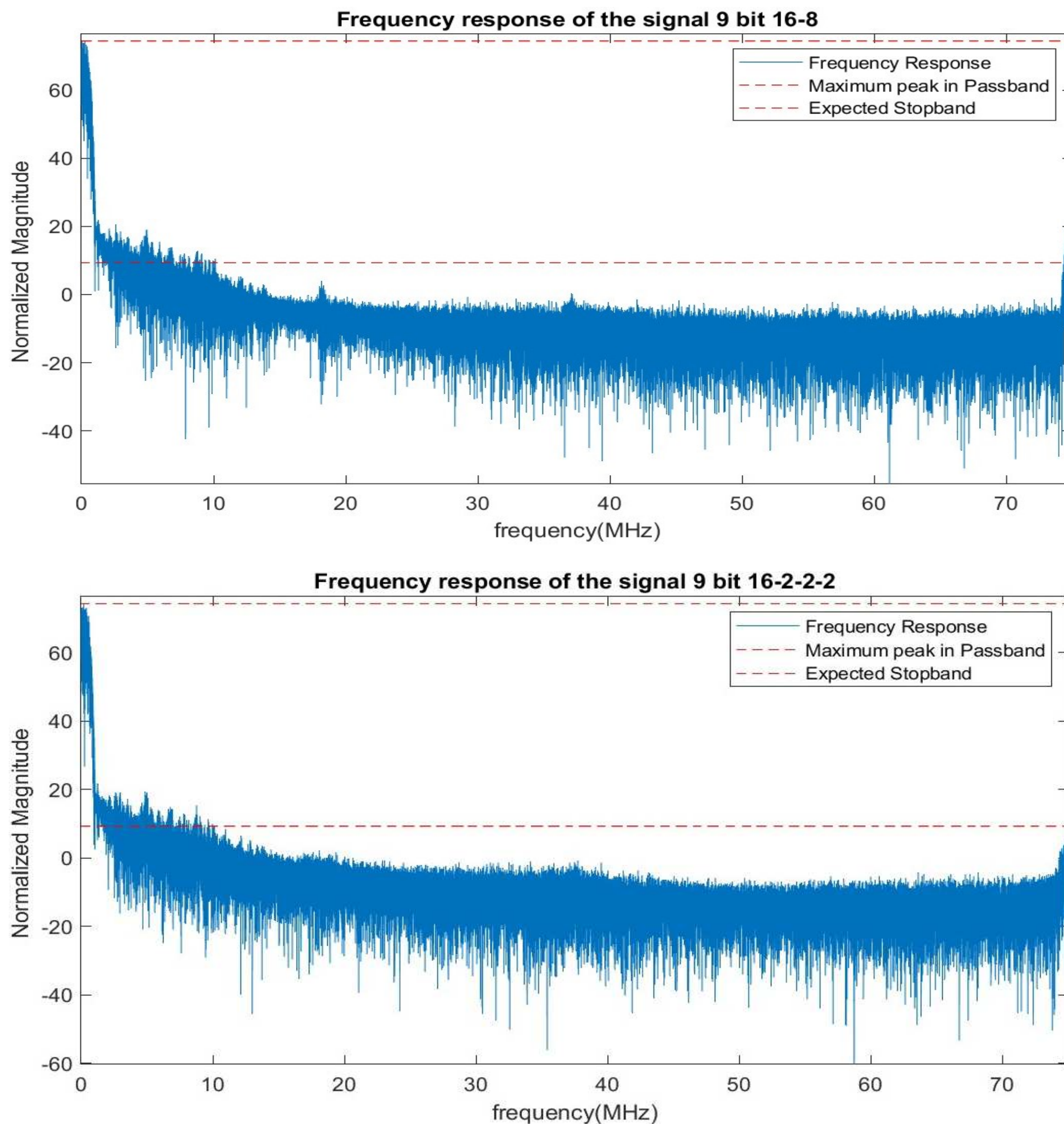


Figura 11: risposta in frequenza dei segnali in uscita dei filtri 16-8 e 16-2-2-2 con quantizzazione a 9 bit

Come si può osservare in figura 11 rispetto ai dati non quantizzati il livello del rumore in alta frequenza è aumentato notevolmente, prima era intorno ai -20 dB mentre ora più vicino agli 0 dB. La quantizzazione dei coefficienti ha inoltre portato a una minore attenuazione a frequenze basse: si può notare come ora l'attenuazione minima si raggiunga solo dopo i 10 MHz mentre prima la si poteva raggiungere a soli 4 MHz. Nel filtro 16-8 si osserva la scomparsa delle prime tre immagini con l'amplificazione della quarta che supera il livello minimo di attenuazione. Questo comportamento sebbene di difficile spiegazione non desta particolari preoccupazioni in quanto i dati in uscita verranno ulteriormente filtrati dal DAC per cui le componenti in alta frequenza verranno ulteriormente attenuate. Nel complesso il segnale mantiene una attenuazione minima di 55 dB che è ancora accettabile per il nostro impiego, passiamo ora dunque alla realizzazione di un modello VHDL per entrambi questi filtri.

2 Realizzazione in VHDL

L'obbiettivo di questa seconda fase è quello di progettare un modello VHDL del circuito finale ed in particolare delle sue componenti principali: il filtro interpolatore e la memoria. La progettazione di queste due componenti richiederà la maggior parte del lavoro mentre il design finale può essere trovato facilmente combinando queste componenti con altre più essenziali. L'analisi sarà svolta per step successivi partendo da una visione sommaria ad alto livello, del tipo ingressi e uscite, fino ad addentrarci nella logica interna. Una volta aver completamente sviluppato i modelli passeremo alla fase successiva in cui andremo a simulare il comportamento delle varie logiche.

2.1 Modello finale del Chip

Seguendo la logica descritta pocanzi partiamo analizzando il modello finale del chip ovvero quello che sarà poi implementato nel silicio con la tecnologia a 22 nm.

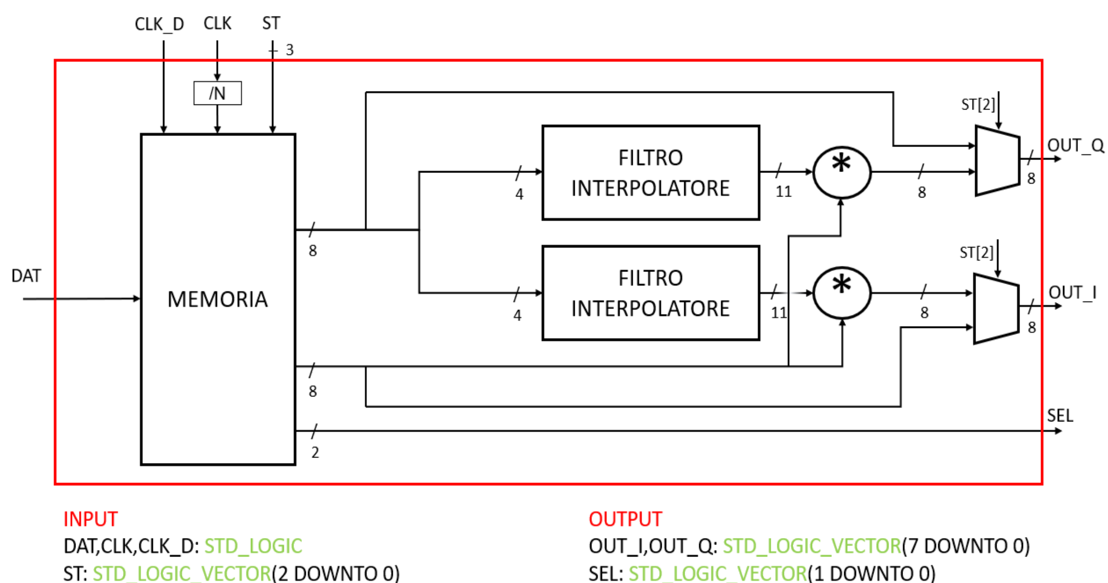


Figura 12: Modello finale del CHIP

Nel modello soprastante i segnali entranti o uscenti dal riquadro rosso rappresenteranno l' interfaccia attraverso cui il circuito interagirà col mondo esterno.[1]Ecco una breve descrizione:

- **DAT:** ingresso a singolo bit per la scrittura dei dati in memoria. Si è deciso di usare questo metodo d'inserimento molto semplice perché in fase di test in laboratorio possiamo inserire i dati in maniera semplice attraverso soli due segnali.
- **CLK_D:** è l'ingresso del segnale di clock usato durante la memorizzazione dei dati. Verrà fornito dalla stessa sorgente dei dati garantendo la corretta memorizzazione dei dati.
- **CLK:** è l'ingresso del segnale di clock principale che verrà fornito direttamente dalla sorgente interna al trasmettitore. Da questo verranno ricavati tutti gli altri segnali di clock a frequenza inferiore che verranno impiegati nel circuito. Verrà

fornito direttamente anche ai filtri anche se dal grafico non si vede per motivi di ingombro.

- **ST:** è il segnale che impone lo stato del circuito. In base a questo verranno decise alcune configurazioni interne quali lo stato dei multiplexer o se abilitare, disabilitare o resettare i filtri. Gli stati saranno cinque in totale e in uno solo di essi si andrà ad abilitare il filtro per l'interpolazione dei dati.
- **OUT_Q:** è l'uscita rappresentante il dato reale; può riportare i dati in uscita dal primo filtro oppure un dato costante preso dalla memoria.
- **OUT_I:** è l'uscita rappresentante il dato immaginario; può riportare i dati in uscita dal secondo filtro oppure un dato costante preso dalla memoria.
- **SEL:** sono due bit forniti dalla memoria al filtro per il settaggio della sorgente di clock interna al trasmettitore. Questo clock sarà quello indirizzato in ingresso nel segnale CLK. Per ottenere la frequenza desiderata di 150 MHz basterà fornire in ingresso il segnale 01.

Da come si può intuire la struttura degli ingressi e delle uscite sarà la stessa della memoria impiegata internamente. L'analogia si estenderà anche agli stati della macchina visto che la memoria è stata progettata specificatamente per questa applicazione. Di fatto la memoria vista nel modello è un contenitore che gestisce altre due memorie interne e la logica utilizzata per il controllo può essere vista come una estensione di quella utilizzata dal modello finale. Detto questo passiamo ad una rapida analisi degli stati, una ulteriore analisi sarà disponibile sul paragrafo dedicato alla memoria.

- **000/011/100/111 = Standby**: non fa nulla fino a nuovo ordine.
- **001 = Programming Cyclic**: memorizza nella memoria i dati da fornire al filtro serialmente tramite il segnale DAT con la frequenza di CLK_D.
- **010 = Running Filter**: vengono abilitati i filtri e la memoria fornisce su una uscita i dati precedentemente caricati a gruppi di otto che vengono poi suddivisi in due dati da quattro per i filtri e nell'altra una parola moltiplicativa per le uscite.
- **101 = Programming Register**: memorizza in un registro apposito della memoria la parola moltiplicativa , due parole a otto bit e due bit per il segnale SEL. Il segnale SEL da allora sarà sempre attivo.
- **110 = Running Constant**: impone le due parole a otto bit direttamente in uscita al trasmettitore per testare il funzionamento costante del Filtro.

Lo stand-by sarà attivato per vari valori del segnale di stato perché dei possibili otto stati che si possono usare con un segnale a tre bit ne sono stati usati solo cinque. Nel caso si volesse aumentare gli stati del chip basterà riattribuire uno di quelli usati per lo stand-by. Gli stati qui descritti sono abbastanza simili a quelli della memoria ma per maggiori informazioni saranno disponibili nel capitolo dedicato. Passando ad altro, dal modello possiamo osservare che sono presenti due filtri interpolatori: uno per la parte reale e uno per la parte immaginaria. Insieme i due dati uscenti dai filtri individueranno un punto nel piano complesso dove il trasmettitore posizionerà il proprio segnale wireless di uscita. Il modello dei filtri non viene ancora specificato in quanto a questo

livello avranno entrambi la stessa interfaccia o in altre parole la logica interna verrà mascherata. Ecco il modello generale dei filtri:

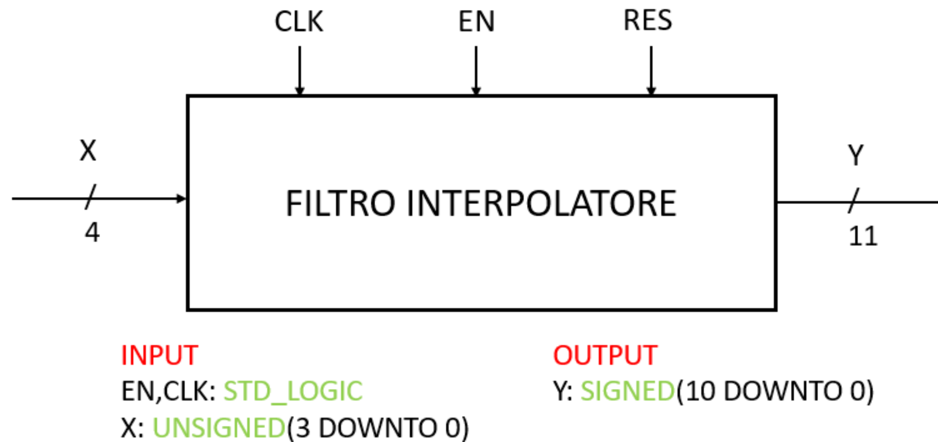


Figura 13: Modello mascherato di un filtro interpolatore

Da come si può vedere nel modello le interfacce sono standard: un ingresso, una uscita, un segnale per il reset e un segnale per l'abilitazione. L'unico segnale da specificare è il clock che sarà quello massimo, fornito dalla sorgente a centocinquanta megahertz. Per ottenere un clock più lento, sia internamente che esternamente, ci saranno dei divisori che otterranno tutti i vari sottomultipli garantendo il corretto flusso dei dati. Sul perché si hanno quattro bit d'ingresso rispetto ai cinque previsti precedentemente troverete un paragrafo dedicato nel capitolo successivo. La logica di controllo dovrà abilitare il funzionamento del filtro solo in uno degli stati operando attraverso i segnali di reset e set:

- **Running Filter:** EN = 1, RES=1
- **Altri stati:** EN = 0, RES=0

Il segnale di reset è inteso come attivo basso. In via generale si potrebbe usare solo il segnale di reset e lasciare l'enable alto in quanto il reset è prevalente. Uno dei problemi da tenere maggiormente in considerazione in questa logica è la gestione dei clock: nell'architettura l'uscita della memoria e l'ingresso dei filtri lavorano alla stessa frequenza ma con clock provenienti da due sorgenti diverse. Non ci sono quindi le garanzie che questi clock siano sincronizzati tra di loro. Per ovviare al problema la soluzione più semplice è quella di piazzare un registro nel filtro operante alla frequenza interna in modo da scandire l'ingresso dei dati mediante questa e garantire il corretto approvvigionamento dei dati. Questa soluzione sarà adottata anche internamente ai due filtri interpolatori. In ultima analisi andiamo a chiarire il significato dei moltiplicatori messi in coda ai filtri. Durante il test di un trasmettitore vi è spesso la necessità di inviare gli stessi segnali ma variandone l'ampiezza. Il modo per poter compiere questa operazione senza dover riscrivere in memoria tutti i dati è quello di moltiplicare le uscite per un dato fattore moltiplicativo. Nel nostro caso si è scelto di usare un fattore moltiplicativo nel formato s1.6 che è il secondo dato uscente dalla memoria. Il fattore moltiplicativo dovrà essere scelto in modo da non causare l'overflow dei dati; il troncamento avverrà in modo da preservare gli otto bit più significativi del segnale d'uscita dei filtri

2.2 Progettazione dei Filtri in VHDL

Per la progettazione dei filtri interpolatori in VHDL è utile, se non consigliato suddividere le varie componenti in altre più piccole e meno complesse così da semplificare l'analisi in fase di test. Infatti, testare unità poco complesse richiede minor sforzo sia nella creazione di testbench sia nella ricerca degli errori. Una volta che le unità avranno il comportamento desiderato si potranno combinare in altre più grandi e nel passare al test delle nuove. Per la creazione e la suddivisione della logica bisogna guardare innanzitutto al modello teorico del filtro per poi passare ad un diagramma VHDL più dettagliato e andando poi ad esplorare ogni entità fino alla più semplice. Con tale metodo vi lascio di seguito il modello teorico dei due filtri:

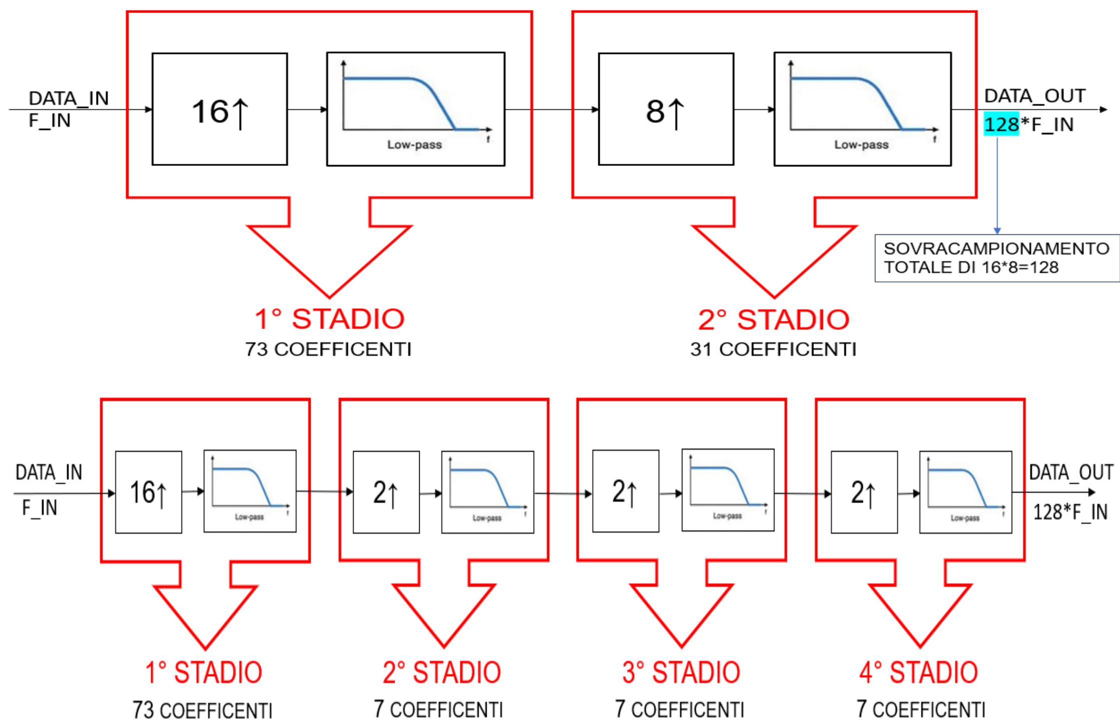


Figura 14: modello teorico dei filtri interpolatori 16-8 e 16-2-2-2

Dal modello teorico si può già intravedere una suddivisione a stadi: due stadi nel primo e quattro stadi nel secondo. Questo potrebbe far pensare che sia necessario realizzare ben sei stadi ma prestando attenzione a quanto detto nel precedente capitolo:

- Il primo stadio di entrambi i filtri è comune: stessi coefficienti
- Gli stadi con sovra campionamento due nel filtro 16-2-2-2 hanno lo stesso numero di coefficienti; quindi, possono essere modellizzati su una stessa entità andando a cambiare opportunamente i parametri.

Il numero totale di stadi da progettare sarà dunque di tre contro i sei evidenziati da questa logica. Tenendo questo a mente possiamo passare alla realizzazione di un modello VHDL più dettagliato dei due filtri.

2.2.1 Modello filtro interpolatore 16-8

Come si può vedere da figura 12 il filtro sarà suddiviso in due stadi rappresentanti i due filtri interpolatori: uno a sedici e uno a otto. La precisione dei dati sarà la stessa di quella testata nel capitolo precedente e visibile in figura 10:

- Ingresso primo stadio a 5 bit in formato s4
- Coefficienti a 9 bit in formato s0.8
- Maggiore precisione nei dati in uscita dagli stadi per arginare overflow e rumore.

Ecco il modello così realizzato:

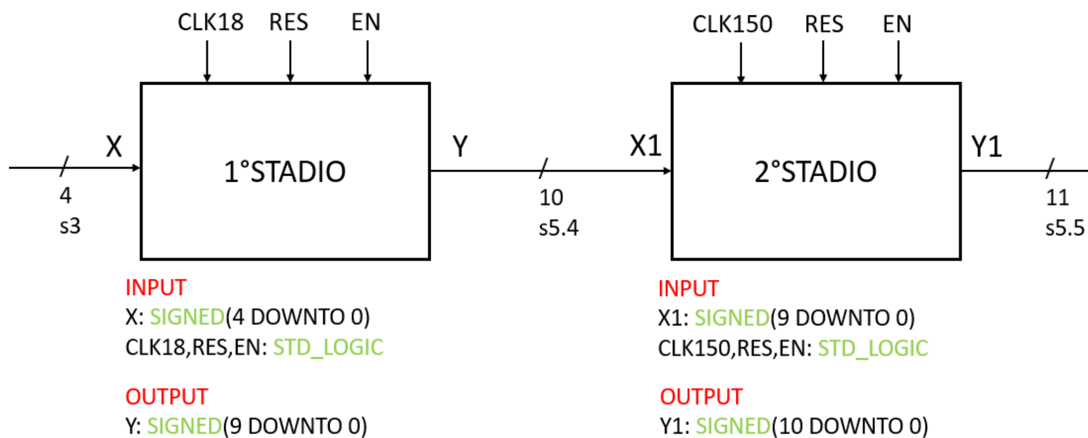


Figura 15: modello VHDL del filtro interpolatore 16-8

Come si può osservare in figura ogni stadio necessita di due clock perché deve acquisire dati a una data frequenza e produrre dati in uscita ad una frequenza maggiore. Sono inoltre presenti un segnale di reset se si vuole interrompere l'operazione di filtraggio e uno di enable se la si vuole stoppare per riprendere successivamente. I diversi segnali di clock hanno frequenza di:

- CLK18 = 18,75 MHz
- CLK150 = 150 MHz

Tra i dati in uscita dal primo stadio e quelli in ingresso nel secondo vi deve essere uguaglianza sia nel numero di bit sia nella rappresentazione. Internamente ai filtri vi dovrà anche essere qualche meccanismo per il troncamento dei dati: ricordiamo infatti che quando andiamo a moltiplicare due numeri in formato virgola fissa il risultato avrà bisogno almeno della somma di bit dei due addendi per essere rappresentato senza perdita di informazione. Nel primo filtro si avrà dunque:

$$s4 * s0.8 = s5.8 \rightarrow troncamento\ di\ 4\ \rightarrow s5.4 \qquad 1°stadio$$

$$s5.4 * s0.8 = s6.12 \rightarrow troncamento\ di\ 7\ \rightarrow s5.5 \qquad 2°stadio$$

In questo secondo caso dovrò andare a troncare anche il primo bit più significativo, quello dopo il bit di segno. Tale bit è sempre ininfluente nei nostri filtri visto che per le caratteristiche del filtro interpolatore non si avrà mai bisogno di utilizzarlo, basta il bit stanziato precedentemente a evitare problemi di overflow. La sua presenza sarebbe giustificata solo se in figura 9 i dati oscillassero in un range maggiore di ±2.

2.2.2 Modello filtro interpolatore 16-2-2-2

Come si può vedere da figura 12 il filtro sarà suddiviso in quattro stadi rappresentanti i quattro filtri interpolatori: uno a sedici e tre a due. La precisione dei dati sarà la stessa di quella testata nel capitolo precedente e visibile in figura 10:

- Ingresso primo stadio a 5 bit in formato s4
- Coefficienti a 9 bit in formato s0.8
- Maggiore precisione nei dati in uscita dagli stadi per arginare overflow e rumore.

Ecco il modello così realizzato:

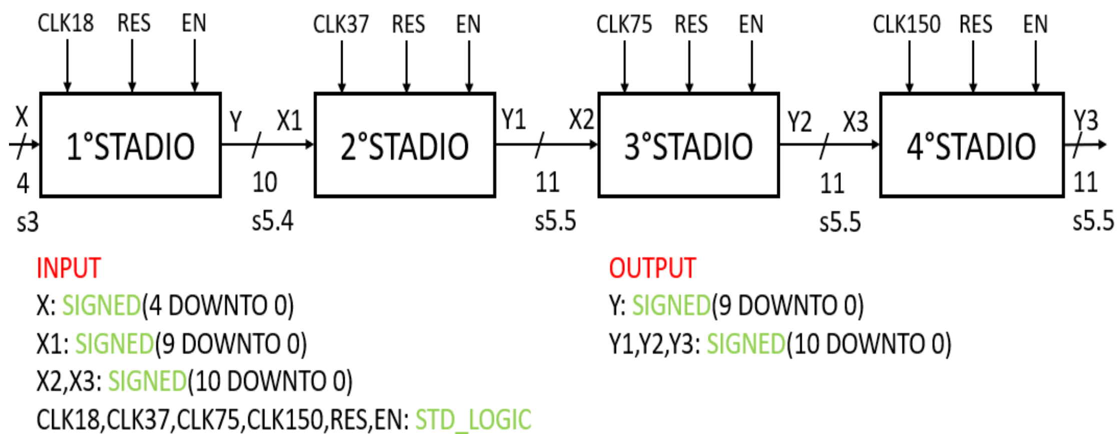


Figura 16: Modello VHDL del filtro interpolatore 16-2-2-2

Il numero di segnali di clock in questo secondo caso è maggiore del precedente in quanto vi sono ben quattro stadi, la frequenza di ciascuno di essi è

- CLK18 = 18,75 MHz
- CLK37 = 37,5 MHz
- CLK75 = 75 MHz
- CLK150 = 150 MHz

Sono presenti anche in questo caso i segnali di reset e di enable per resettare o fermare il filtro. Per quanto riguarda il troncamento dei dati il procedimento è lo stesso usato nel filtro 16-2-2-2.

$$s4 * s0.8 = s5.8 \rightarrow troncamento\ di\ 4\ \rightarrow s5.4 \qquad 1°stadio$$

$$s5.4 * s0.8 = s6.12 \rightarrow troncamento\ di\ 7\ \rightarrow s5.5 \qquad 2°stadio$$

$$s5.5 * s0.8 = s6.13 \rightarrow troncamento\ di\ 8\ \rightarrow s5.4 \qquad 3°\ e\ 4°stadio$$

Concettualmente è tutto molto simile a quello visto precedentemente, passiamo dunque alla prossima fase della progettazione andando a studiare i singoli stadi dei filtri.

2.2.3 Formato e significato dei dati in ingresso

Lo scopo ultimo del filtro interpolatore è quello di fornire dati ad un trasmettitore che rappresentano tutte le possibili costellazioni rappresentabili nelle modulazioni 16-QAM e 4-QAM. Da qui la necessità di trovare il minor numero di bit per cui si possano rappresentare le costellazioni delle due modulazioni:

- 16-QAM → necessari almeno 4 bit
- 4-QAM → necessari almeno 2 bit

Il minor numero di bit con cui si possono rappresentare entrambi i formati è dunque di quattro bit. Tuttavia, usare solo quattro bit con segno presenta dei problemi nel 16-QAM perché il segnale creato non è a media nulla per il fatto che i valori oscillano tra -8 è +7. Ogni valore della costellazione ha teoricamente la stessa probabilità di essere chiamato; quindi, il segnale generato dalla successione di questi valori dovrebbe essere a media nulla, cosa che non avviene se usassimo la rappresentazione a 4 bit con segno. I valori devono essere mappati in un modo diverso secondo la seguente tabella:

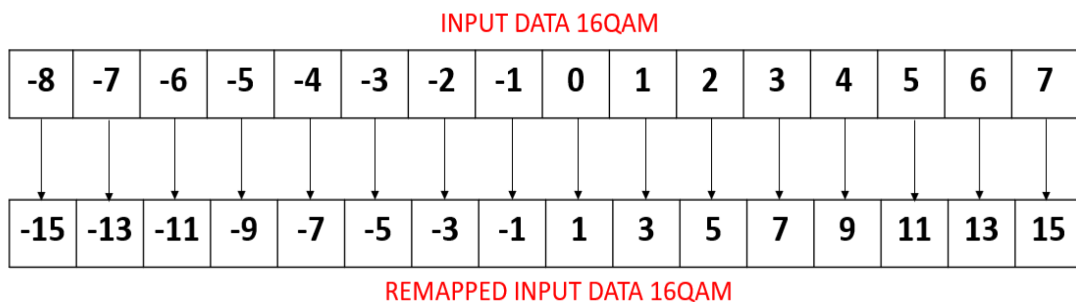


Figura 17: Mappa dei dati nella 16-QAM

Il range di valori ora è perfettamente simmetrico allo zero generando a sua volta segnali a media nulla come trovato teoricamente. I valori effettivamente forniti all'ingresso di ogni filtro interpolatore saranno a cinque bit con segno ma ne saranno usati solo metà di tutto il range. Questo escamotage permette di avere dati ingresso nei filtri interpolatori a quattro bit ma rappresentanti di valori a cinque bit. Da qui si può capire la discrepanza tra quanto visto nel capitolo precedente dove i dati ingresso erano a 5 bit e questo dove i dati sono a 4 bit. Sfruttando queste mappe si possono anche descrivere i valori della 4-QAM nel seguente modo:

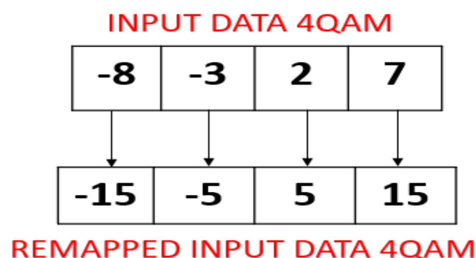


Figura 18: Mappa dei dati 4-QAM

Con questa mappa è possibile usare la stessa logica della 16-QAM per mappare la 4-QAM senza portare alcuna modifica circuitale. La semplicità dei dati in ingresso porterà più avanti ad altre scelte di progettazione in particolare nel primo stadio dei filtri interpolatori che analizzeremo successivamente.

2.2.4 Design degli stadi

Per la realizzazione del primo stadio e anche dei successivi sono state principalmente due le soluzioni realizzative trovate:

1. **Classica:** Creare un unico grande filtro dove far entrare il dato di ingresso seguito da quindici zeri. La soluzione in questo caso avrebbe necessitato di un numero di moltiplicatori almeno pari a trentotto, sfruttando la simmetria dei coefficienti e un numero di sommatori e di blocchi di ritardo circa uguali ai coefficienti. Design preposto principalmente al risparmio di area e soggetto ad alti consumi di potenza: il filtro funzionerebbe alla frequenza di clock maggiore consumando maggiore potenza per il maggior numero di switch subiti dalle componenti. Secondariamente guardando ai dati di ingresso il filtro si troverà ad analizzare zero per la maggior parte del tempo che di certo non rende giustificato il maggior consumo di potenza. In conclusione, la soluzione risulterebbe vantaggiosa solo se avessi dei forti limiti di area e dovessi ridurre la dimensione dei filtri.

2. **Scomposta:** Il problema dell'inefficienza di avere zeri in ingresso nella soluzione classica può essere gestito andando a notare che solo i coefficienti distanziati tra loro di sedici valori concorrono alla creazione di un valore di uscita. Andando a raccogliere tutti questi coefficienti ci si accorge che il nostro filtro può essere scomposto in sedici filtri più piccoli: nove da cinque coefficienti e sette da quattro. I filtri ridotti funzionerebbero alla frequenza di clock inferiore e basterebbe un selezionatore alla frequenza di clock maggiore che alterni i valori in uscita per ottenere il comportamento desiderato. In questo caso il consumo di area sarebbe maggiore perché non si riesce a dimezzare il numero di moltiplicatori come nel caso precedente ma si otterrebbe un notevole risparmio dei consumi. Il debugging sarebbe inoltre più facile perché basterebbe testare solo un filtro ridotto e usare quello come esempio per la creazione degli altri. L'utilizzo di filtri a frequenza minore di quella di uscita ripara anche da violazioni delle costanti di tempo che diventano meno probabili.

Vista la convenienza di questa seconda logica il modello verrà adottato per tutti gli altri stadi scegliendo, il numero dei filtri in base al fattore di sovra campionamento desiderato . Lo schema sarà lo stesso di quello in figura 19 ma con un numero di filtri ridotti variabile: otto per il secondo stadio del filtro 16-8 e due per gli ultimi tre stadi del filtro 16-2-2-2

2.2.5 Modello del primo stadio

Il primo stadio come abbiamo avuto modo di vedere sarà comune ad entrambe le logiche. I segnali di clock utilizzati saranno due: quello a 18.75 megahertz fornito esternamente e quello a 1,7 megahertz che viene ricavato da questo, ulteriori informazioni saranno fornite nel paragrafo successivo.

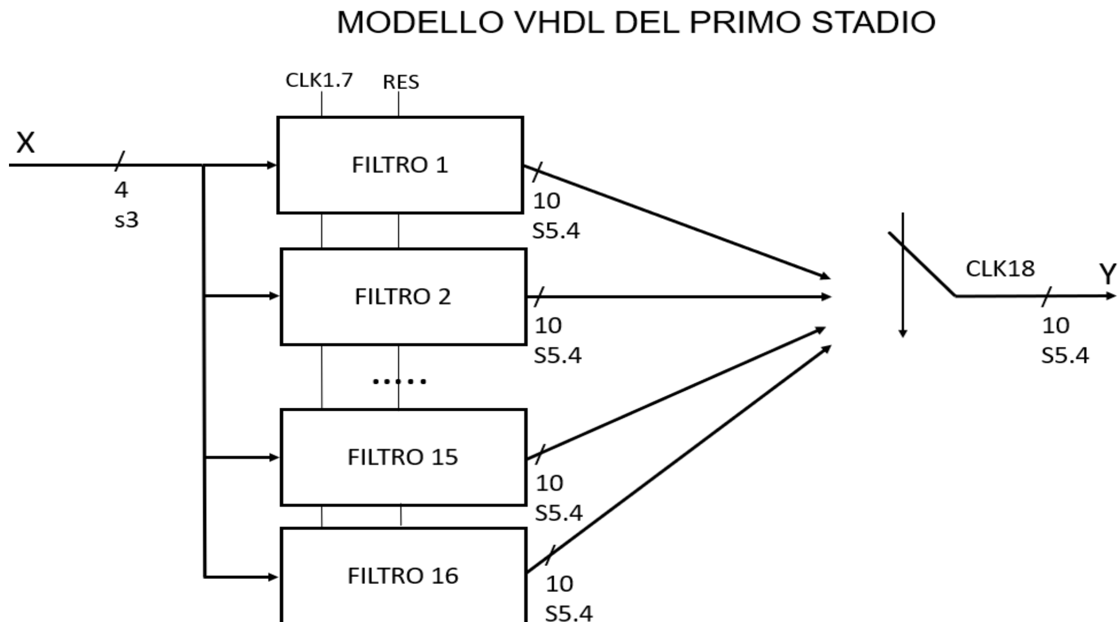


Figura 19: modello VHDL del primo filtro interpolatore

Nel modello esposto nella figura soprastante si può osservare la presenza di sedici filtri collegati al medesimo dato d'ingresso, le cui uscite sono collegate ad un selettore funzionante alla frequenza di clock maggiore per comandare in uscita tutti i dati nell'ordine corretto. In ingresso vi sarà un registro operante a un sedicesimo della frequenza di clock, ottenuto mediante divisione del primo, così da essere aggiornato solo al momento opportuno. Ogni filtro è molto semplice e necessità per il proprio funzionamento di due segnali oltre al dato in ingresso che sono: il clock, il reset. Il segnale di enable non comanda direttamente i filtri ma va invece a bloccare lo stato del clock lento che hanno in ingresso impedendo a questi ultimi di proseguire l'analisi. Nel paragrafo successivo si parlerà appunto della generazione interna dei clock. Il comportamento di tale circuito sarà quindi quello proposto nella soluzione scomposta presentata nel paragrafo precedente.

Tutti gli stadi dei due filtri saranno implementati con la logica della scomposizione; l'unica differenza sarà dunque la realizzazione dei filtri ridotti che dipenderà dalle caratteristiche dei filtri e dai dati analizzati.

2.2.6 Gestione e generazione dei clock interni

Nel nostro modello per semplicità andremo a derivare i vari clock da un singolo segnale di clock dato da un oscillatore al quarzo ad una frequenza di centocinquanta megahertz. La derivazione può essere fatta in modo molto semplice andando a impostare un counter in corsa libera: i bit dello stesso rappresenteranno dei clock a frequenza inferiori. Nella seguente figura si vede come questo avviene.

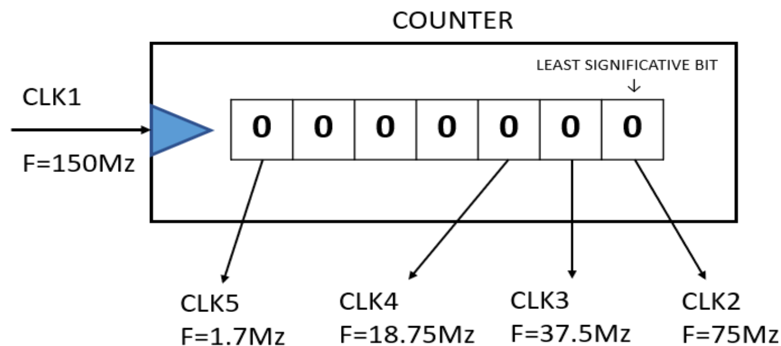


Figura 20: Schema di generazione clock con un counter

Come si può notare in figura sovrastante il bit meno significativo andrà a generare un clock ad una frequenza dimezzata, il secondo a un quarto della frequenza originale e così via secondo potenze di due. I principali benefici di questa scelta sono:

- Riduzione della complessità dell'albero di clock: non devo più portare i clock da una sorgente esterna ma vengo creati direttamente in loco.
- Semplificazioni dei modelli RTL: ho un minor numero di segnali d'ingresso per cui più facili da gestire.
- Semplificare l'attuazione del segnale di enable: per bloccare i filtri in modo efficace sarà necessario andare ad agire sul counter. Se blocco l'aggiornamento del counter i clock da esso generanti non innescheranno più i filtri che rimarranno bloccati.

Andiamo ora a vedere il modello dell'albero di clock che si andrà a generare con questa scelta:

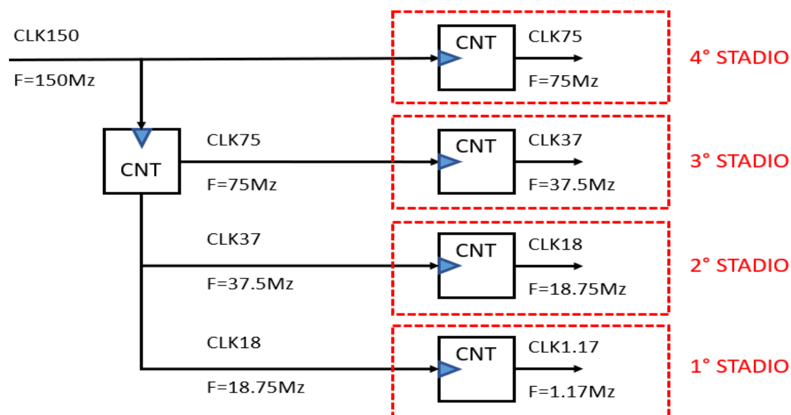


Figura 21: Albero di clock per il 16-2-2-2

La generazione dei primi clock viene fatta direttamente nella top entity propagandole poi ai vari stadi che genereranno a loro volta altri segnali di clock per i vari filtri interni.

Questa scelta è stata presa considerando che internamente agli stadi vi è già un counter che seleziona le varie uscite dei filtri, quindi, basta riportare il bit di interesse all'ingresso dei vari filtri senza bisogno di ulteriore logica. Nel caso del 16-8 l'approccio è stato simile con la creazione di un clock nella top entity che verrà propagato al primo stadio mentre per il secondo è sufficiente utilizzare quello d'ingresso.

Come visto precedentemente nel modello globale questa struttura ha una criticità nel fatto che è molto difficile garantire che tutti clock alla stessa frequenza siano identici gli uni agli altri. Ogni stadio ha un suo clock interno per comandare gli stadi mentre il dato da inserire viene fornito con il clock dello stadio precedente. Questo modello non è l'ideale per un corretto approvvigionamento degli stadi. La soluzione, come visto precedentemente sarà quella di porre dei registri in ingresso agli stadi che vadano a campionare l'ingresso garantendo le tempistiche corrette dei dati. Oltre a questo l'inserimento dei registri va a rompere il flusso diretto che si aveva tra gli stadi predisponendo una sorta di pipeline aumentando la velocità del circuito.

2.2.7 Modello dei filtri del primo stadio

Come anticipato precedentemente il fatto di avere sedici possibili valori per i dati in ingresso e dei coefficienti costanti ci permette di prendere in considerazione due soluzioni per la realizzazione dei filtri ridotti:

1. **Moltiplicatori:** soluzione consigliata se devo moltiplicare due numeri variabili di volta in volta o se il numero dei bit degli addendi è troppo elevato da rendere una LUT poco vantaggiosa.

2. **Look Up Table (LUT):** soluzione consigliata se i due numeri da moltiplicare hanno un basso numero di bit o se uno di essi è costante nel tempo.

Con soli sedici possibili valori in ingresso il primo stadio può servirsi delle LUT in sostituzione dei moltiplicatori, cosa non possibile per gli stadi a seguire. Le LUT in questione avranno la seguente struttura:

LOOK UP TABLE

1*H	3*H	5*H	7*H	9*H	11*H	13*H	15*H
-1*H	-3*H	-5*H	-7*H	-9*H	-11*H	-13*H	-15*H

Figura 22: modello generico delle Look Up Table

Il fattore H riportato nelle LUT rappresenta il generico coefficiente del filtro mentre i valori per cui viene moltiplicato sono tutti i possibili valori d'ingresso nel filtro. La profondità delle LUT, ovvero il numero di bit che compone ogni dato in esse contenuto dovrebbe essere reso variabile in modo che si possa aggiustare in base alle prestazioni che si vogliono ottenere:

- **N = 14** → massima profondità, maggiore accuratezza, maggiore consumo di area

- **N = 10** → minima profondità, minima accuratezza, minor consumo di area.

Facendo dei rapidi calcoli quello che si ottiene è

$$Num_{coeff} * Num_{ingresso} * Profondità = 73 * 16 * N = 1168 * N$$

I calcoli indicano che un solo bit nella risoluzione corrisponde a ben mille centosessantotto singoli registri. Per ulteriori modifiche in fase di test sarebbe prudente rendere variabile la profondità delle LUT così da testare il comportamento dei filtri con diversi livelli di risoluzione interna e scegliere quella più adeguata.

Passando ora alla realizzazione dei filtri questi sono stati realizzati in forma diretta in quanto funzionando ad una frequenza di clock relativamente bassa hanno ampi margini rispetto alle costanti di tempo. Ecco il modello VHDL elaborato per questi filtri:

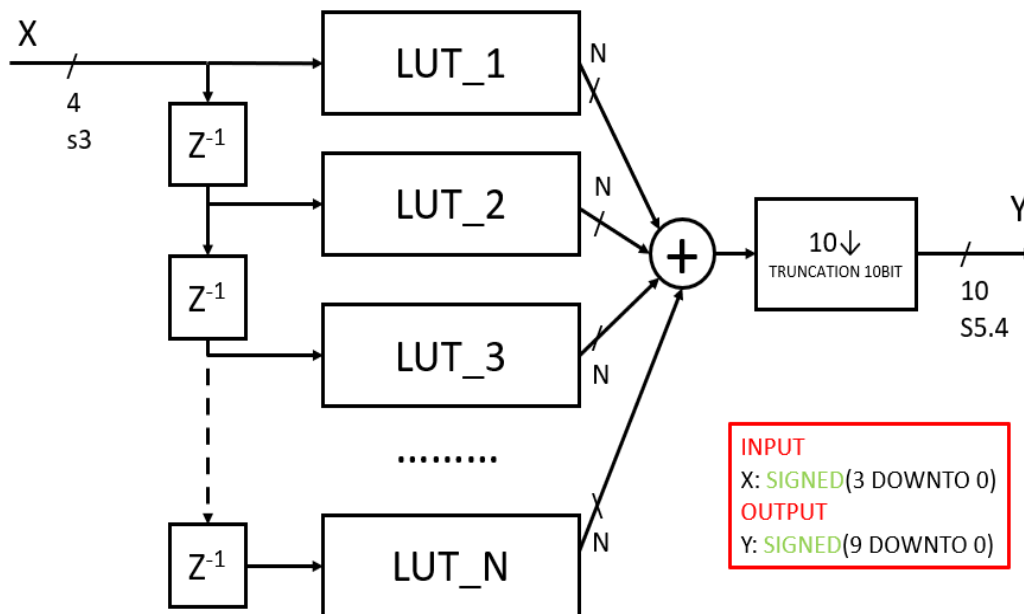


Figura 23: Modello VHDL di un filtro del primo stadio

Essendo la profondità delle LUT variabile è necessario mettere un blocco di troncamento prima dell'uscita così da troncare le cifre meno significative e tenere soltanto quelle desiderate. Un secondo vantaggio di utilizzare la forma diretta lo si può apprezzare in figura dove si può notare che posizionando i blocchi di ritardo sui dati d'ingresso invece che sui dati d'uscita dalle LUT vi è un notevole risparmio di area (4 registri per ogni blocco contro gli N registri se posizionati altrove). I blocchi di ritardo anche se non segnato hanno le seguenti funzioni:

- Sono sincronizzati con il clock più lento CLK1.
- Possono essere resettati dal segnale di reset.

Questa soluzione essendo molto pratica verrà riproposta anche più avanti, vedi figura 20.

2.2.8 Modello dei filtri del secondo stadio 16-8

Il secondo stadio avrà la stessa struttura del primo visto in figura 17 con la differenza che i filtri saranno otto invece che sedici. Per la realizzazione dei filtri invece non è più vantaggioso usare le LUT, infatti, facendo lo stesso calcolo di prima:

$$Num_{coeff} * Num_{ingresso} * Profondità = 31 * 2^{10} * N = 31744 * N$$

Il numero di singoli registri da allocare per un singolo bit di profondità diventa molto alto per non parlare del numero minimo che si dovrebbe avere per una profondità di undici bit. La soluzione più vantaggiosa è quella di sostituire alle LUT viste precedentemente con dei moltiplicatori come riportato nella figura sottostante.

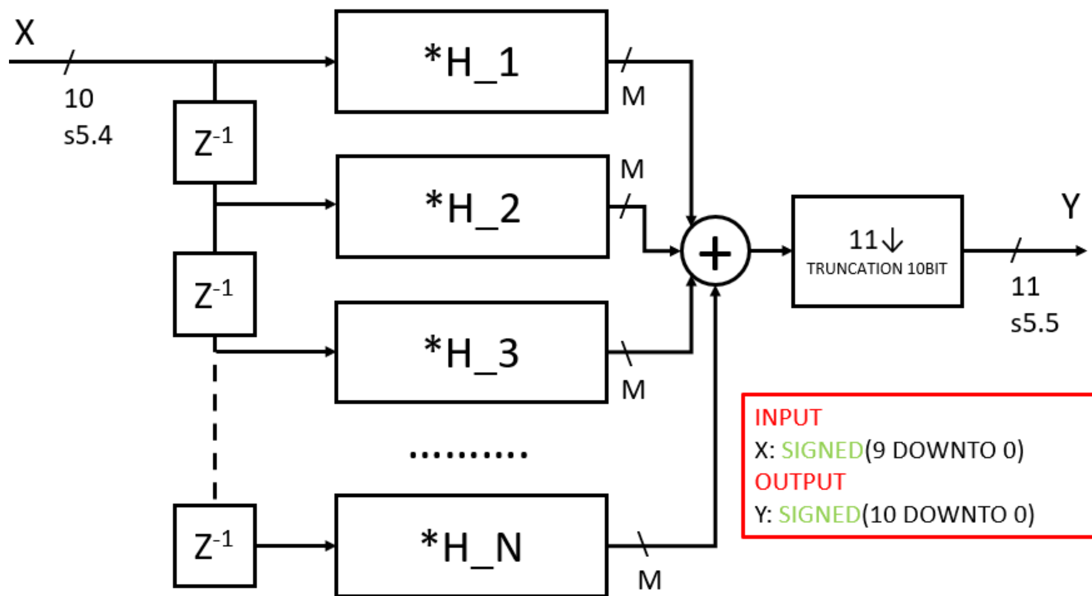


Figura 24: modello VHDL di un filtro del secondo stadio 16-8

Anche in questo caso ho usato la forma diretta in quanto di più semplice implementazione avendo già fatto i filtri del primo stadio ed essendo abbastanza sicuro che con la tecnologia che sto utilizzando (22 nanometri) dovrebbe comportarsi bene anche a queste frequenze. Le criticità in ambito temporale sono localizzate principalmente nei primi filtri che essendo selezionati per primi devono fornire i risultati in tempi più stretti rispetto agli ultimi. In ogni caso tengo pronto un modello in forma inversa se dovessero intervenire delle criticità nelle costanti di tempo. La precisione dei dati in uscita dai moltiplicatori viene resa variabile per poter essere aggiustata in fase di test.

Come nel caso dei filtri del primo stadio i blocchi di ritardo lavorano alla frequenza di clock minore: CLK2 e sono resettabili dal reset.

2.2.9 Modello del secondo, terzo e quarto stadio 16-2-2-2

Gli stadi con sovra campionamento due del filtro 16-2-2-2 sebbene piccoli verranno anch'essi scomposti secondo la procedura vista precedentemente per arrivare a una logica simile a quella espressa in figura 19 ma con soli due filtri. La realizzazione dei filtri ridotti in questo caso sarà diversa da quella usata precedentemente vista la particolare struttura di questi ultimi deducibile anche dai grafici in figura 5.

La struttura generale di questi filtri può essere espressa nella seguente tabella:

A	0	B	1	B	0	A

La scomposizione poi porta alla creazione dei seguenti filtri ridotti:

1°

A	B	B	A

2°

0	1	0

Il primo filtro essendo simmetrico può essere implementato usando la forma trasposta inversa permettendoci di dimezzare il numero dei moltiplicatori complessivi da quattro a due. La seguente figura ne mostra il modello VHDL definitivo:

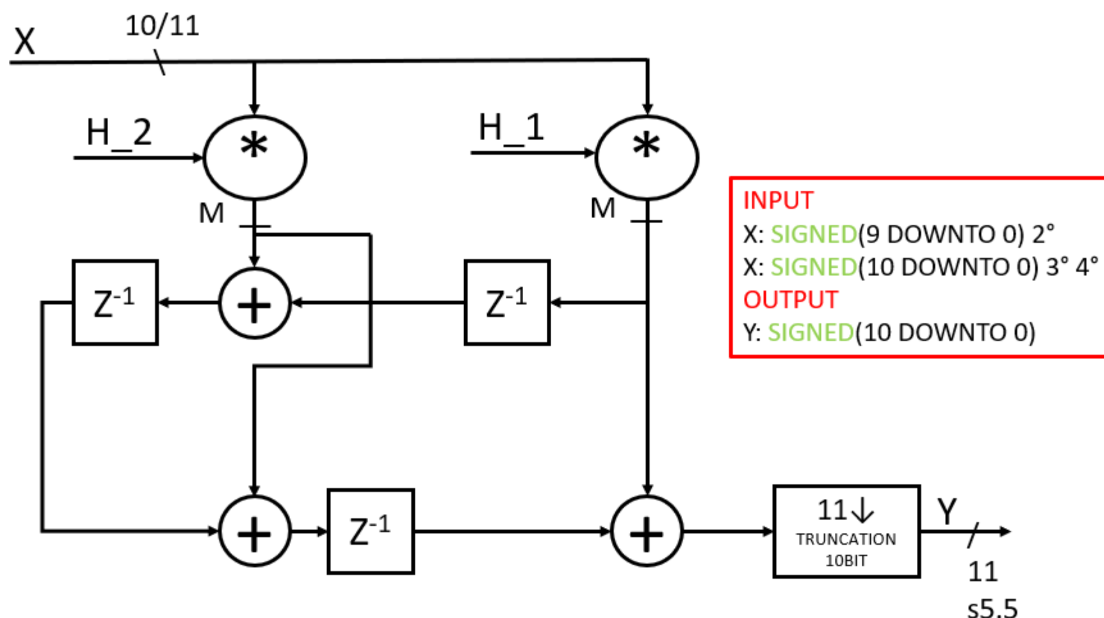


Figura 25:modello VHDL dei filtri a fattore 2

I dati d'ingresso presentano due valori in quanto il secondo filtro che prende dati dal primo avrà in ingresso dati a 10 bit e produrrà in uscita dati a 11 bit mentre il terzo e il quarto avranno in ingresso dati a 11 bit e produrranno in uscita dati ad 11 bit. Anche in questo caso come visto nei filtri del primo stadio ci riserviamo di decidere in fase di collaudo il numero di bit da tenere per la precisione interna, motivo per cui mettiamo il valore generico M nei dati uscenti dai moltiplicatori e il blocco di troncamento in uscita:

- M = 19 valore massimo

- M = 11 valore minimo

La precisione interna M avrà effetto principale sui sommatori e sui blocchi di ritardo ad essi collegati. La cosa si applica anche al paragrafo precedente.

Il secondo filtro può essere realizzato semplicemente attraverso un blocco di ritardo e sottoponendo il dato a uno shift a sinistra di otto bit: il valore di uno nella nostra notazione viene rappresentato come una moltiplicazione per due alla ottava. Questo ci permette di limitare il numero dei moltiplicatori a due per ogni stadio.

2.2.10 Confronto tra i modelli 16-8 e 16-2-2-2

Ora che abbiamo prodotto dei modelli VHDL per i nostri filtri interpolatori andiamo a stimare, almeno teoricamente, quale delle due implementazioni risulta più efficiente in base all'utilizzo delle componenti. Partiamo riassumendo le caratteristiche cruciali di ogni filtro:

Filtro Interpolatore 16-8:

- Modello a due stadi
- Numero di LUT necessarie per il primo stadio: 67
- Numero stimato di sommatori primo stadio: 54
- Numero di moltiplicatori necessari per il secondo stadio: 27
- Numero stimato di sommatori secondo stadio: 21

Filtro Interpolatore 16-2-2-2:

- Modello a quattro stadi
- Numero di LUT necessarie per il primo stadio: 67
- Numero stimato di sommatori primo stadio: 54
- Numero di moltiplicatori complessivo per gli ultimi stadi: 6
- Numero stimato di sommatori ultimi stadi: 12

Guardando ai dati degli ultimi stadi e considerando che il primo stadio è comune ad entrambe le soluzioni appare chiaro che la soluzione a 16-2-2-2 risulti più vantaggiosa. Il fatto di promettere le stesse prestazioni ma con un numero di moltiplicatori complessivi più di quattro volte inferiore alla prima la rende di gran lunga migliore. Prima però di prendere una decisione bisogna andare a simulare il comportamento di entrambe le soluzioni.

2.3 Progettazione Memoria in VHDL

Ora che si ha un modello abbastanza definitivo del filtro bisogna progettare una struttura capace di memorizzare e di somministrare successivamente i dati al filtro. Le soluzioni in questo caso possono essere diverse e dipendono dal tipo di utilizzo che se ne vuole fare. Nel nostro caso volendo semplicemente testare il funzionamento finale del chip e del trasmettitore abbiamo bisogno di una memoria non particolarmente complessa ma piuttosto versatile e facile da impostare. Le caratteristiche che sono state scelte per questa memoria saranno:

- Presenza di una memoria ciclica di 1024 registri di 8 bit
- Presenza di un registro di 26 bit di cui:
  1) 8 bit per la parola del moltiplicatore finale del filtro
  2) 8+8 bit per parole costanti per simulare un segnale costante
  3) 2 bit per il settaggio del trasmettitore
- Il caricamento dei dati avverrà in maniera seriale mediante due segnali esterni a 1 bit: uno per i dati e uno per il clock.
- La memoria dovrà fornire nel caso di utilizzo ciclico, la sequenza dei dati d'ingresso e la parola moltiplicativa al filtro.
- Presenza di un segnale in uscita per il settaggio del trasmettitore.

L'utilizzo di questa struttura a doppio clock permette di evitare il ricorso a protocolli di comunicazione seriali piuttosto complessi quali I2C e I3C; tali protocolli seppure molto più prestanti sono difficili da usare, rendendoli vantaggiosi a progetto definitivo e non per un semplice testing. Passando ora a tradurre le specifiche in un modello VHDL quello che si ottiene è:

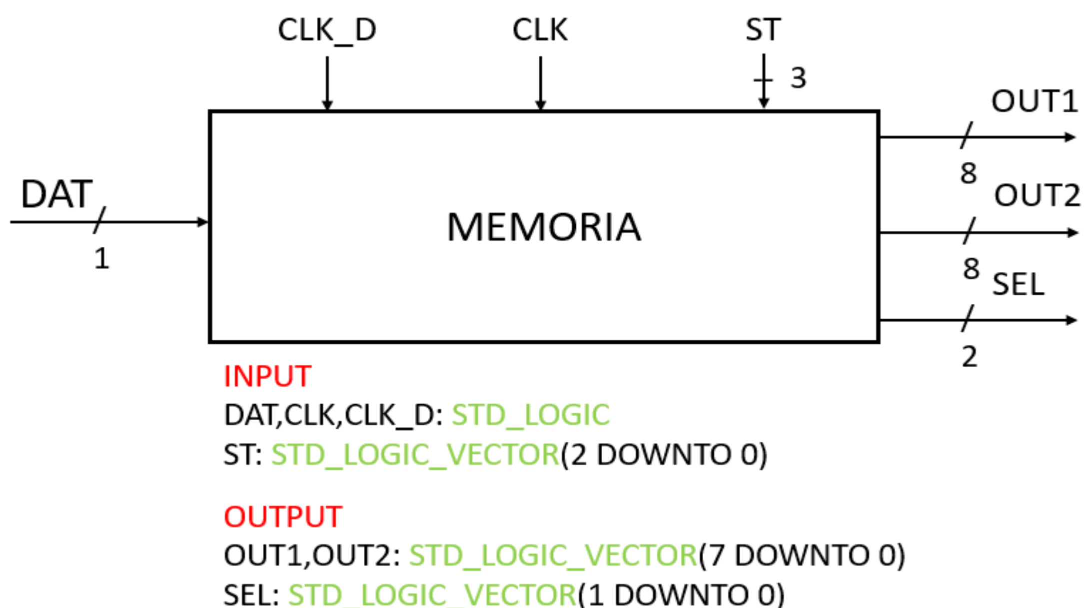


Figura 26: modello VHDL memoria per il filtro interpolatore

Il comportamento della memoria vista qui sopra dipende dai bit di stato rappresentati dal segnale ST, in particolare il primo bit seleziona quale delle due memorie si andrà ad utilizzare mente i successivi selezioneranno il tipo di operazione da eseguire. L'elenco dei possibili comandi per il settaggio sarà dunque:

- **000/011/100/111 = Standby**: non fa nulla fino a nuovo ordine.
- **001 = Programming Cyclic**: memorizza nella memoria ciclica i dati provenienti dal segnale d'ingresso seriale DAT, con la frequenza fornita dal segnale di clock esterno CLK_D. Nel dato a otto bit i primi quattro rappresenteranno la parte reale mentre gli altri la parte immaginaria.
- **010 = Running Filter**: legge la memoria ciclica fornendo i dati in OUT1 con la frequenza del clock esterno CLK. Fornisce inoltre il coefficiente moltiplicativo del filtro in OUT2 e i due bit di settaggio per il trasmettitore in SEL.
- **101 = Programming Register**: memorizza nel registro i dati provenienti dal segnale d'ingresso seriale DAT, con la frequenza fornita dal segnale di clock esterno CLK_D.
- **110 = Running Constant**: indirizza i dati presenti sul registro direttamente al DAC bypassando il filtro. Di fatto l'uscita OUT1 fornirà il la parte reale del dato mentre OUT2 quella immaginaria. Fornisce inoltre i due bit per il settaggio del trasmettitore.

La presenza di due segnali di clock uno per la memorizzazione e uno per l'inserimento dei dati nel filtro è necessaria per semplificare l'utilizzo del chip da parte di utilizzatori esterni. Avere un segnale di clock apposito per la memorizzazione dei dati, permette di usare una sorgente qualsiasi senza doversi mettere a usare quella interna. Una volta caricata la memoria ciclica potrà essere riprodotta un numero indefinito di volte e riscritta in base al test da effettuare. Il registro, una volta caricato con i valori prestabiliti, comanderà direttamente il segnale SET indifferentemente dallo stato in cui si trova la memoria. L'assenza di un segnale di reset è dovuta al fatto che durante la fase di programmazione i nuovi dati vanno a cancellare quelli vecchi che quindi non necessitano di essere resettati. L'inserimento dei dati dovrà essere fatto secondo lo standard del Big Ending ovvero dal bit più significativo fino al meno significativo. Questo standard è stato scelto per la facilità implementativa e di lettura: da computer basta generare una serie di 1024 numeri, tradurli in binario a 4 bit e unirli a formare un'unica sequenza di dati. Per esempio:

$$4 + 3i\,, -3 + i\,, -2 \;\rightarrow 0100\ 0011\ 1101\ 0001\ 1110$$

2.3.1 Progettazione memoria ciclica

Passando ora alla fase implementativa una possibile struttura di memoria atta a rappresentare il comportamento della memoria ciclica sopraindicato può essere espressa nel seguente modello.

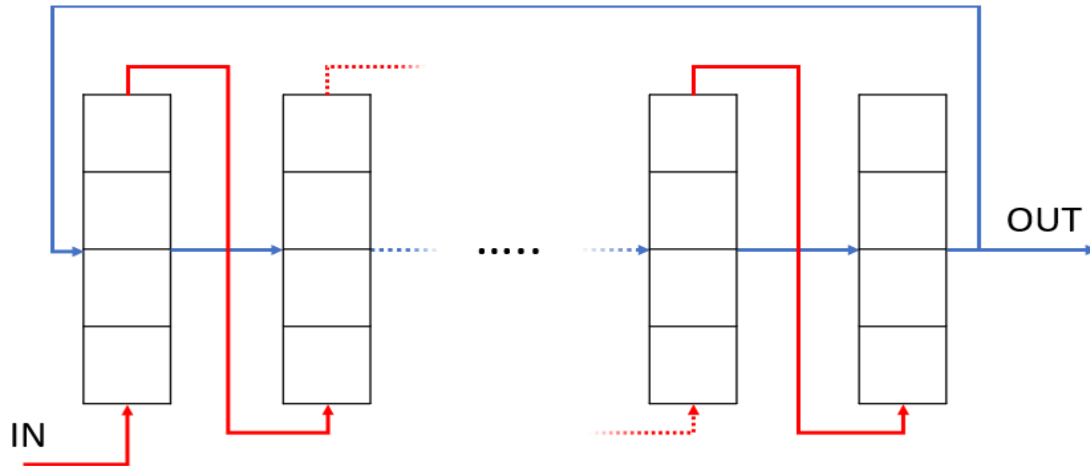


Figura 27: Modello strutturale della memoria ciclica

Nello schematico si può capire come la struttura della memoria sia basata sulla concatenazione di una singola struttura. Questa entità è a sua volta composta da otto registri singoli che a seconda dello stato di operatività della macchina fanno scorrere i bit in uno dei due percorsi evidenziati:

- **Rosso:** si attiva nella fase di Programming Cyclic e permette di caricare in memoria i dati d'ingresso tramite lo scorrimento di un bit per volta. Il primo dato inserito dovrà quindi attraversare verticalmente, registro per registro, tutte le celle prima di giungere nella posizione desiderata. Per il corretto funzionamento della memoria saranno necessari un numero di colpi di clock uguale al numero totale dei bit memorizzati.

- **Blu:** si attiva nella fase di Running Filter e permette di far scorrere i dati parallelamente di cella in cella quattro bit alla volta. La ciclicità è garantita dal fatto che nell'ultima cella, oltre a indirizzare il dato in uscita, viene riportato indietro nella prima cella reinserendolo nel ciclo. Per ottenere un giro completo in questo caso, traslando quattro bit per volta, saranno necessari un numero di colpi di clock pari al numero di registri istanziati.

Per quanto riguarda la realizzazione di questa memoria si proceduto per step andando a realizzare per prima la cella elementare a 8 bit e successivamente l'intera memoria mediante la generazione ricorsiva della stessa. In particolare, oltre agli ingressi/uscite dovute ai percorsi sopraindicati saranno necessari anche un segnale di clock e un bit per il settaggio i percorsi. Con questo modello si può modificare agilmente sia il numero di registri che la profondità degli stessi anche in caso di applicazioni future. Secondariamente, in sede di sintesi, usando una cella elementare poco complessa assicura una sintesi rapida ed efficace.

2.3.2 Progettazione registro

Una volta creata la memoria ciclica andiamo a progettare anche il registro a 26 bit partendo dal definire il significato dei dati contenuti al suo interno, essenziale per il suo corretto utilizzo. La seguente figura mostra pertanto la suddivisione interna del registro con evidenziate le varie aree e il loro significato.

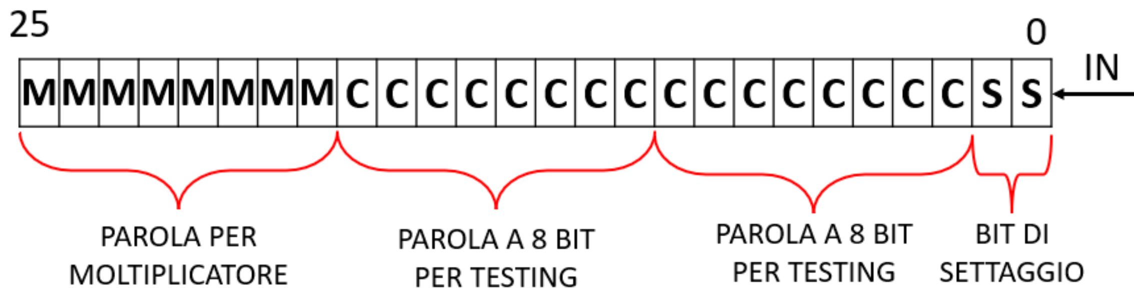


Figura 28: suddivisione interna del registro di memoria

I dati entrando dal meno significativo e finendo poi su quello maggiormente significativo seguiranno una logica Big Ending in continuità con quello visto precedentemente. A loro volta, anche i dati internamente contenuti seguiranno la stessa logica; l'inserimento partirà dunque, col dato più significativo della parola di moltiplicazione, proseguendo fino al meno significativo e ripartendo dal più significativo della seconda parola. Riportando i codici utilizzati si avrà:

- REG(25 downto 18): parola di moltiplicazione.
- REG(17 downto 10): prima parola rappresentante il dato reale.
- REG(9 downto 2): seconda parola rappresentante il dato immaginario.
- REG(1 downto 0): bit di settaggio.

La realizzazione è fatta semplicemente concatenando una serie di registri e di fatto ricalca quella usata per la creazione delle celle nella memoria ciclica.

3 Simulazioni dei modelli VHDL

La simulazione dei modelli vhdl ha il compito principale di evidenziare e correggere tutti i malfunzionamenti, assicurando il buon comportamento del circuito. Il processo di simulazione è molto semplice e ha bisogno di sole due componenti:

- **Testbench:** consiste nella creazione di un design apposito per testare il funzionamento del circuito e non oggetto di sintesi successiva. In pratica fornisce degli stimoli in ingresso e ne esamina gli effetti sul circuito.

- **Tool di elaborazione Grafica:** consiste in un compilatore che mediante il testbench propaga i segnali internamente, secondo la logica descritta nei modelli fino alle uscite. Nel suo utilizzo principale produce l'intera evoluzione temporale di uno o più segnali. I tool utilizzati sono stati DVE e Verdi.

La figura seguente mostra una rielaborazione grafica di quanto detto.

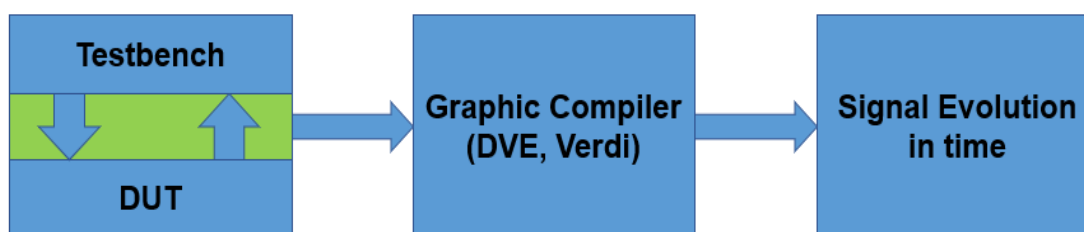


Figura 29: modello operativo delle simulazioni

Questo procedimento dovrebbe essere riportato per ogni entità che viene istanziata ma per motivi di brevità saranno riportate solo le simulazioni delle entità principali, tralasciando le varie simulazioni intermedie. L'obbiettivo delle simulazioni dei filtri 16-8 e 16-2-2-2, non è solo quello di testarne il corretto funzionamento ma anche quello di trovare la configurazione ottimale che permetta di avere un buon rapporto tra costi e prestazioni. Per trovare questa intesa si andranno a produrre tre simulazioni distinte variando la risoluzione interna dei dati, che dal capitolo precedentemente è stata fatta appositamente scalabile.

1. **Simulazione a Bassa Risoluzione: N = 11 M = 12**

2. **Simulazione a Media Risoluzione: N = 13 M = 15**

3. **Simulazione ad Alta Risoluzione: N = 14   M = 18**

I valori di N e M sono riportati nei paragrafi precedenti e rappresentano uno la risoluzione dei dati contenuti nelle LUT e l'altro la risoluzione dei dati post moltiplicazione e prima dei sommatori. I risultati verranno analizzati nei seguenti paragrafi e serviranno al fine di scegliere quale modello andare ad implementare definitivamente. Una volta concluso il test dei filtri si procederà con il test della memoria e del modello finale; bisognerà assicurarsi solo che abbiano il comportamento corretto.

3.1 Simulazione del filtro interpolatore

Per testare il corretto funzionamento dei filtri andremo a somministrare una serie di dati in ingresso e ad analizzarne lo spettro dei dati in uscita. La simulazione verrà effettuata a tutte le risoluzioni e i risultati comparati tra di loro. Infine, verranno testate anche i segnali di Enable e Reset in un paragrafo dedicato.

3.1.1 Simulazione a Bassa Risoluzione

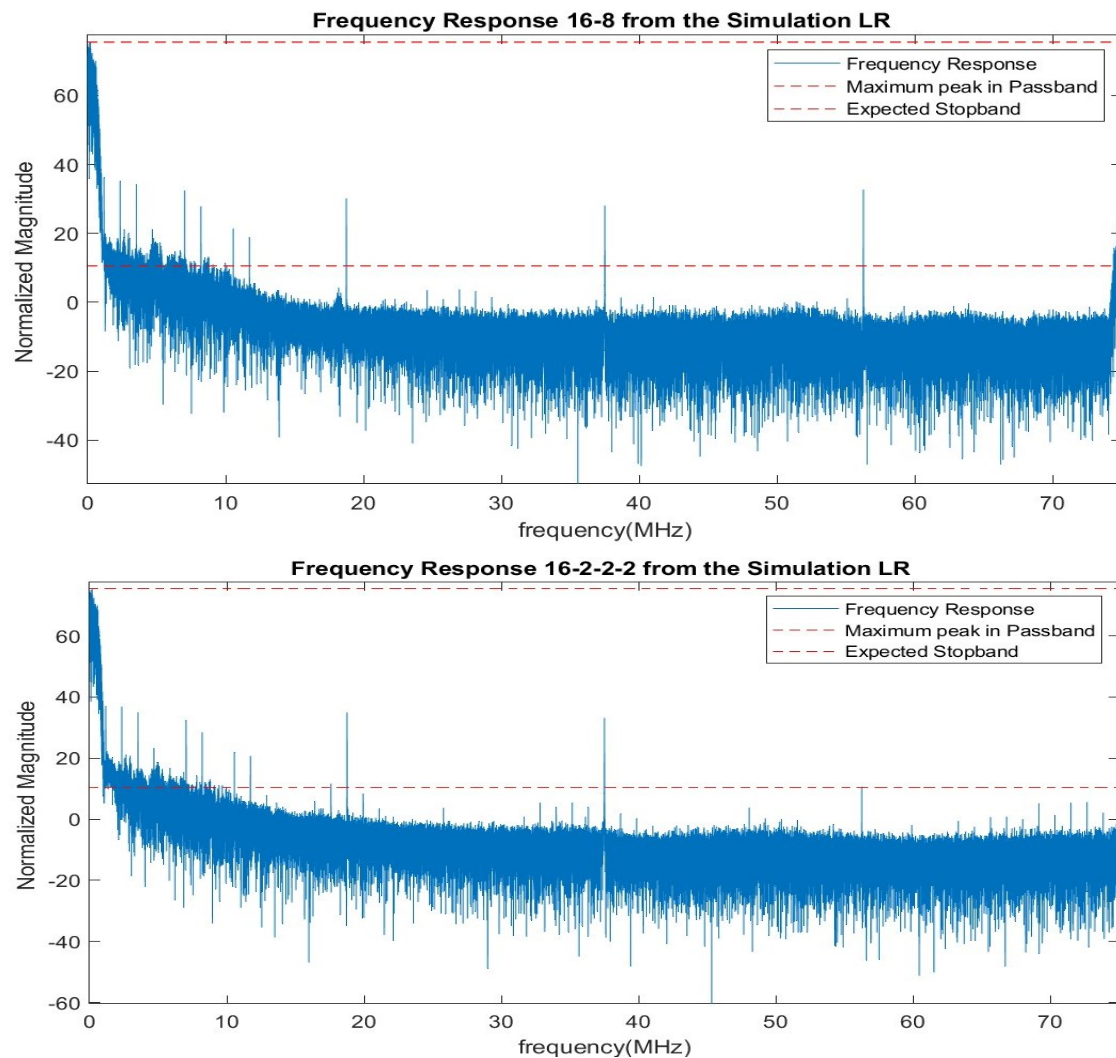


Figura 30: Risposte in frequenza dei segnali di uscita 16-8 e 16-2-2-2 a bassa risoluzione

Come si può osservare nella figura sovrastante il fatto di tenere per le somme interne solo un bit in più del necessario non fa lavorare correttamente il filtro creando varie armoniche in entrambe le configurazioni; in particolare la prima, la seconda e la terza sono le più alte con valori massimi attorno ai 36 dB. Queste prime armoniche sono le più importanti perché sono le più difficili da rimuovere anche considerando il successivo filtro passa basso che si troverà all'ingresso del DAC. La presenza di armoniche così elevate a bassa frequenza sta ad indicare che il filtro del primo stadio non sta funzionando correttamente. Le armoniche in alta frequenza, diciamo dai dieci megahertz in su vengono attenuate più facilmente anche se in questo caso sono abbastanza alte intorno ai 34/35 dB. La minima attenuazione sarà inferiore ai 40 dB molto meno rispetto ai 65 dB impostati all'inizio.

3.1.2 Simulazione a Media Risoluzione

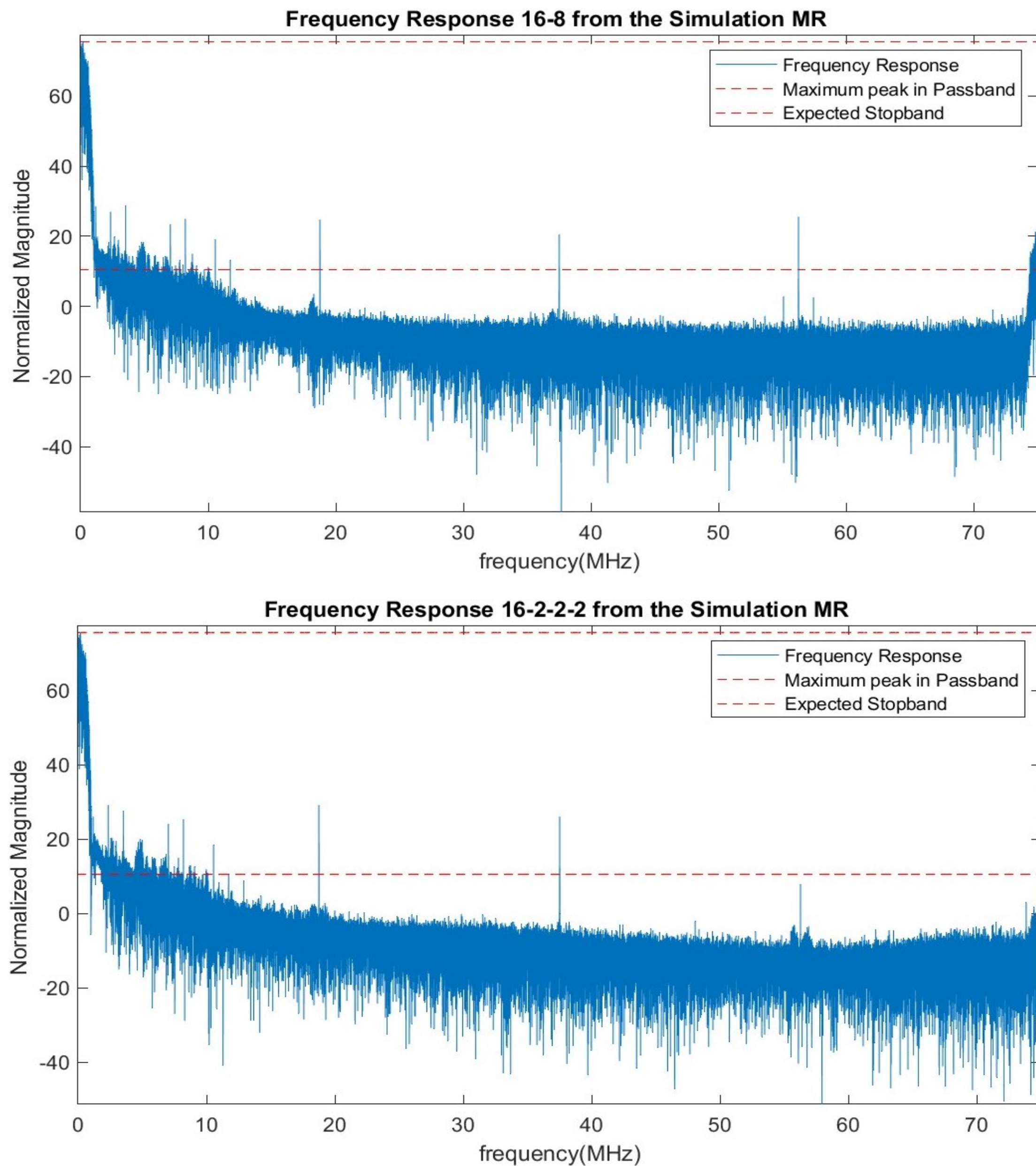


Figura 31: Risposte in frequenza dei segnali di uscita 16-8 e 16-2-2-2 a media risoluzione

L'utilizzo di una maggiore risoluzione dei dati interni comporta notevoli miglioramenti nella risposta in frequenza dei due segnali con una significativa attenuazione delle armoniche: diminuzione media attorno ai 8-9 dB. Con riferimento alla figura 11 si può notare la grande somiglianza, tolti i picchi dovuti alle armoniche, con il modello stimato in precedenza andando a vedere che anche qui la soglia minima dei sessantacinque dB di attenuazione non si raggiunge subito ma intorno ai 10 MHz. Per osservare più chiaramente le differenze tra le armoniche nelle varie simulazioni fare riferimento alla tabella conclusiva.

3.1.3 Simulazione ad Alta Risoluzione

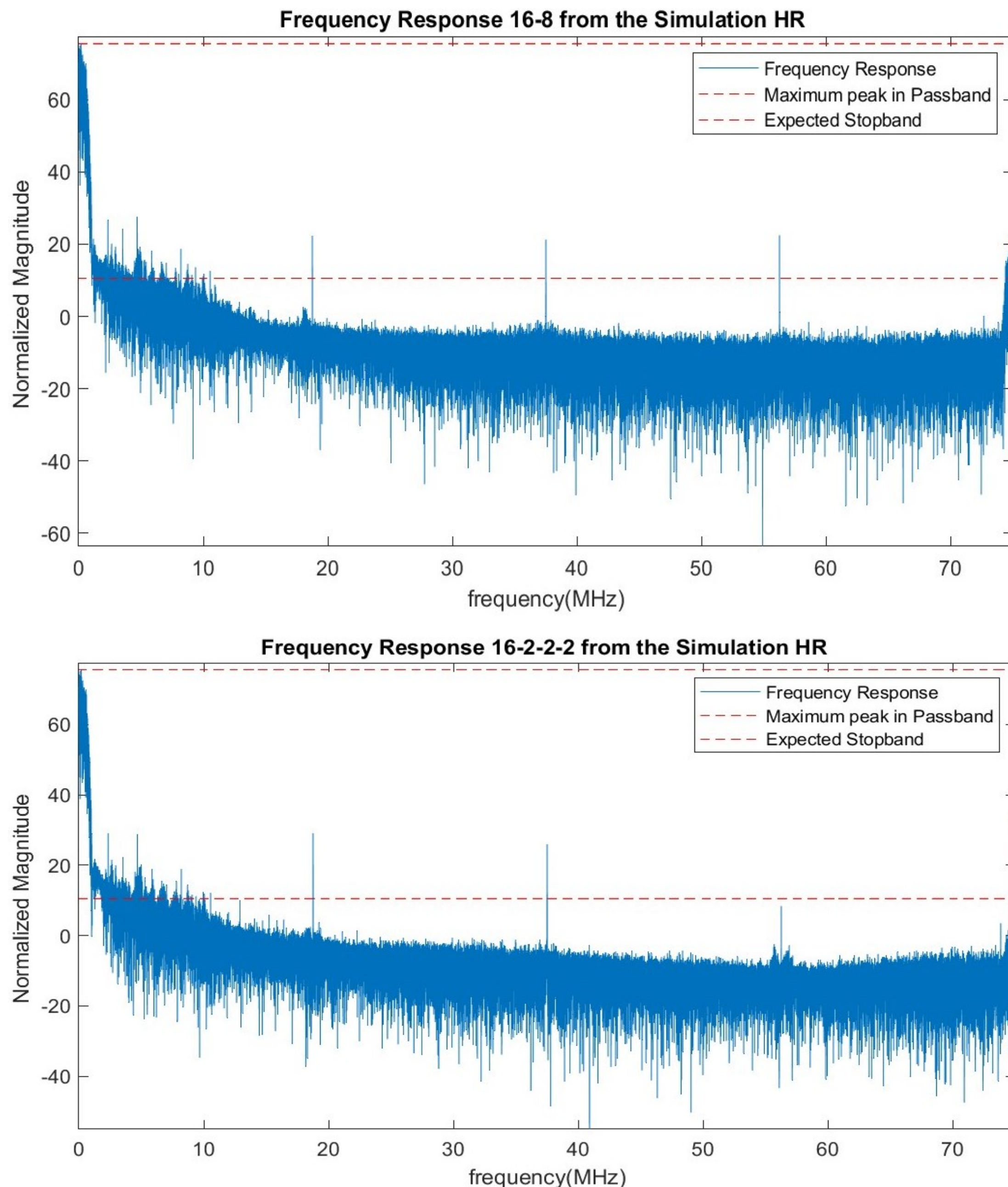


Figura 32: Risposte in frequenza dei segnali di uscita 16-8 e 16-2-2-2 ad alta risoluzione

Rispetto al caso precedente l'attenuazione delle armoniche è meno evidente anche se si può notare la cancellazione della prima nel 16-2-2-2 e in una lieve attenuazione di quelle più alte. Globalmente vale quello detto per i precedenti filtri: la somiglianza rispetto a quanto stimato precedentemente e il miglior funzionamento dei filtri rispetto al primo caso. Questo risultato è il massimo risultato ottenibile con le nostre condizioni visto che una risoluzione ulteriore non sarebbe realizzabile. Per ottenere risultati ancora più precisi sarebbe necessario aumentare la risoluzione dei coefficienti fino a dieci bit ma anche ora le prestazioni sono comunque soddisfacenti.

3.1.4 Confronto tra le varie Simulazioni

In questa parte andremo ad analizzare le caratteristiche dei segnali di uscita così da meglio inquadrare gli effetti della perdita di risoluzione dei dati. Prenderemo in considerazione solo le prime armoniche, quelle a bassa frequenza, ovvero quelle dovute al primo filtro, visto che quelle in alta frequenza vengono attenuate efficacemente dal filtro del DAC. La seguente tabella riporta i valori associati alle varie componenti armoniche rilevate dalle varie logiche.

Tabella riassuntiva delle armoniche							
Armonica	Frequenza(MHz)	16-8 (dB)			16-2-2-2 (dB)		
		HR	MR	LR	HR	MR	LR
1°	1.1719	21.5	28.6	36.2	----	26.1	37.2
2°	2.3438	26.7	27	35.2	29.1	29.1	36.9
3°	3.5156	24	28.7	34.1	22.2	27.4	34.8
4°	4.6875	27.5	16.5	21.1	28.6	19.3	23.3
5°	5.8594	18.1	14.4	----	18.8	14.4	16.8
6°	7.0312	14.8	23.5	32.5	14.4	24.1	32.4
7°	8.2031	18.7	25	27.8	18.9	25.4	28.4
8°	9.375	13.6	----	----	----	----	----
StopBand Minima		48	46.8	39.3	46.4	46.4	38.3

La StopBand si riferisce sempre al massimo trovato in banda base: in questo caso a 75.5 dB. Analizzando la tabella si può concludere che:

- **Basse risoluzioni:** la performance dei filtri risulta alquanto compromessa con armoniche decisamente superiori agli altri due casi presi in esame.

- **Medie risoluzioni:** riduzione notevole del contenuto armonico rispetto al caso precedente ma rimane abbastanza alto sulle prime.

- **Alte Risoluzioni:** buone attenuazioni globalmente anche se alcune armoniche risultano più alte, come nel caso della quarta armonica. Tale comportamento è di difficile spiegazione ma sembra una redistribuzione del carico armonico, visto che globalmente il contenuto armonico decresce. La forte attenuazione della prima è comunque molto positiva.

Sul piano comportamentale le due soluzioni sono quasi perfettamente sovrapponibili, basti guardare alla seguente figura che sovrappone i dati ottenuti dai due filtri a media risoluzione con quelli d'ingresso. In questo caso faccio riferimento alla figura che si otterrebbe se si troncassero gli ultimi tre bit, ovvero i dati effettivamente entranti nel trasmettitore. La scelta è stata voluta perché in questo caso si riesce ad apprezzare meglio le differenze tra i due segnali.

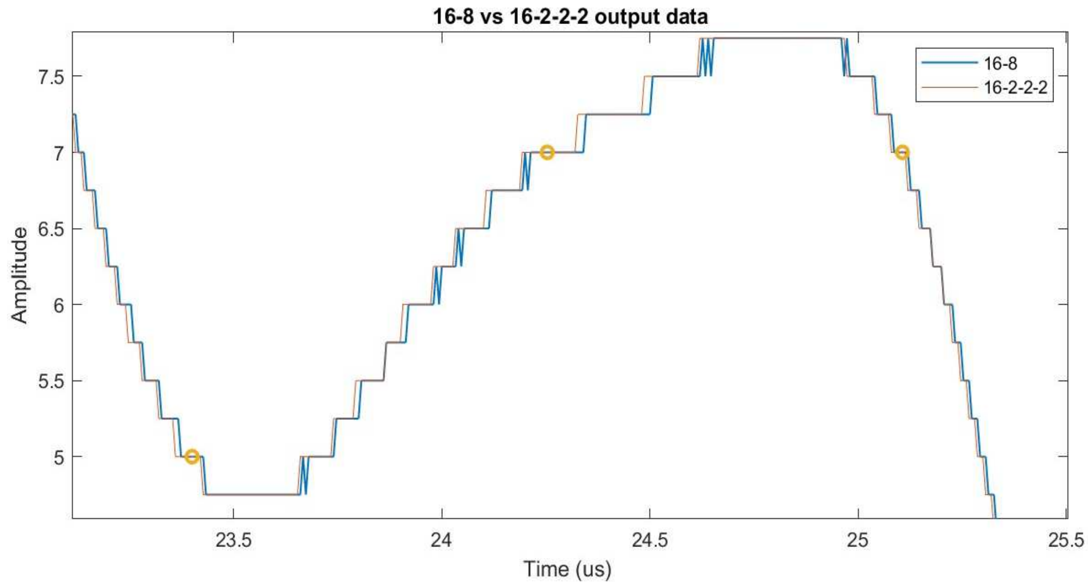


Figura 33: confronto tra i dati di uscita dei vari filtri

La situazione è la stessa a tutte le risoluzioni e dimostra il buon comportamento rispetto alle simulazioni fatte in precedenza. Anche analizzando il grafico su un maggior numero di punti il risultato rimane invariato; talvolta un segnale è più pulito mentre l'altro è un po' più sporco ma il comportamento dei due rimane coerente. Per analizzare meglio le divergenze tra i due segnali ho elaborato un grafico che mostra l'evoluzione temporale del segnale di errore, ottenuto come valore assoluto della differenza tra i due. In questo caso riporto quello ottenuto con la risoluzione massima a undici bit, se lo facessi a otto bit non sarebbe corretto perché verrebbero ridotti a un solo bit.

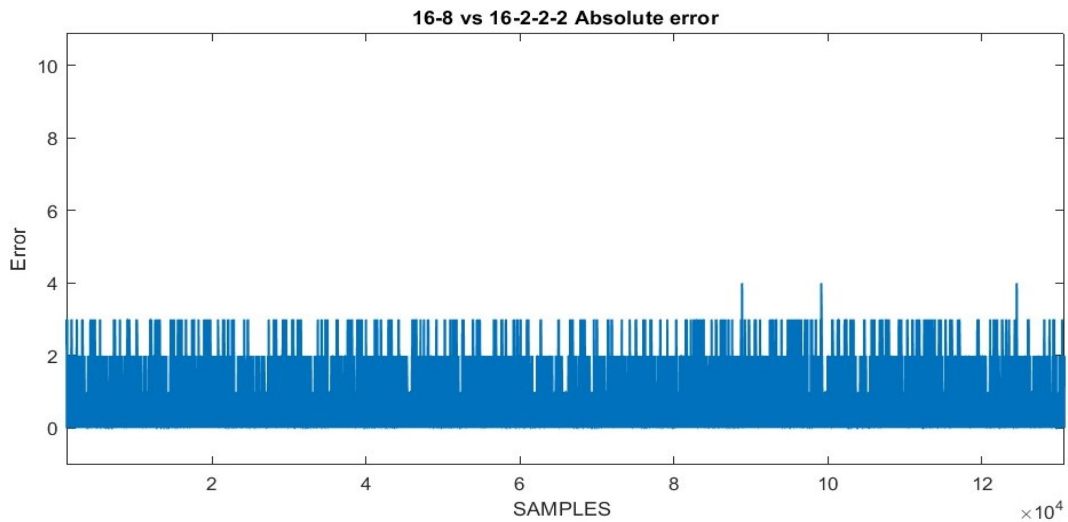


Figura 34: evoluzione temporale dell'errore in bit nel formato a 11 bit

Come si può vedere dal grafico il massimo errore raggiunto tra i due segnali è di quattro; corrispondente a due bit nel formato a undici e ridotto a uno solo nel formato a otto bit. Dal punto di vista comportamentale possiamo concludere che entrambe le soluzioni si comportano bene. Per decidere quale dei due filtri utilizzare dovremmo andare nella fase successiva studiando l'occupazione di area dei due circuiti.

3.1.5 Test segnali di Reset ed Enable

Per testare i segnali di reset ed enable si simulerà il comportamento con alcuni dati in ingresso, per esempio quelli usati precedentemente, andando ad alzare ed abbassare i suddetti segnali per vederne gli effetti sull'uscita del filtro. La simulazione prevede di testare in particolare tre casi:

1. **EN=0, RES=1** → test segnale di enable
2. **EN=1, RES=0** → test segnale di reset
3. **EN=0, RES=0** → test di accavallamento segnale reset ed enable

I tre casi permetteranno di testare tutte le principali casistiche che possono incorrere nel funzionamento ordinario del circuito. Il test di accavallamento serve per vedere quale delle due dinamiche prevale nel caso entrambi i segnali si attivino, in questo caso logica prevalente sarà il reset. Andiamo ora a simulare il comportamento così ottenuto nei primi due casi riportati.

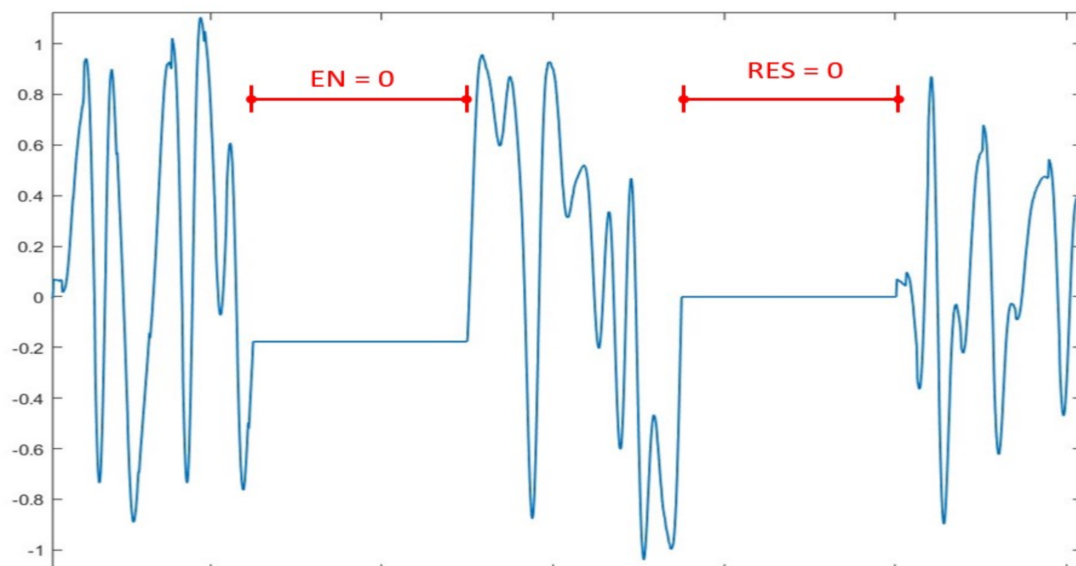


Figura 35: Risultati simulazione dei segnali di reset ed enable singoli.

Come si può vedere in figura:

- **Enable OFF:** durante la disabilitazione del segnale di enable l'uscita viene tenuta costante ed alla riabilitazione la curva prosegue regolarmente per il valore successivo.
- **Reset ON:** durante la fase di reset il filtro porta subito l'uscita a zero ed alla riabilitazione vi è un breve transitorio dovuto al caricamento del filtro simile a quello che si vede nella fase iniziale.

Questo comprova il buon comportamento di entrambi i segnali, il prossimo passo sarà quello di andare a vedere quando questi segnali vengono accavallati. Nel test di accavallamento si andrà a disabilitare prima il segnale di enable e poi ad attivare quello di reset: il reset ha una importanza maggiore rispetto all'enable e facendo il contrario non si vedrebbe differenza tra il prima e il dopo.

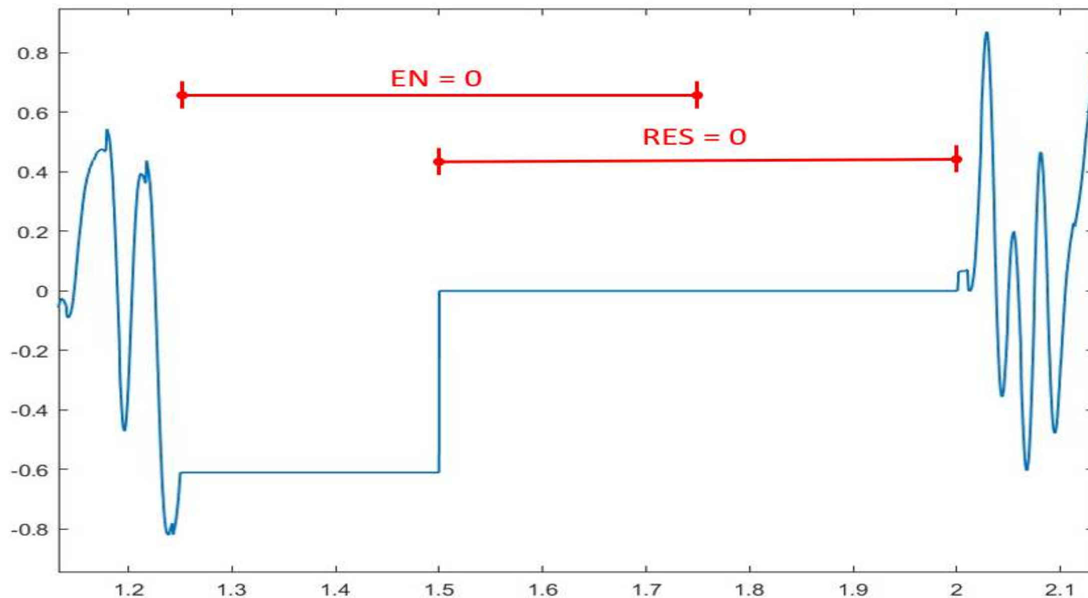


Figura 36: Risultati simulazione dei segnali di reset ed enable in accavallamento

Inizialmente con la disattivazione dell'enable l'uscita rimane costante fino all'attivazione del reset che la resetta a zero; anche in questo caso alla fine del reset si può osservare un transitorio prima del ritorno al funzionamento ordinario. Nel complesso le due simulazioni hanno mostrato il comportamento atteso ma per essere maggiormente sicuri ed evitare l'insorgere di malfunzionamenti bisognerà andare a vedere anche lo stato dei registri e dei counter interni che devono essere consistenti con lo stato del filtro:

- **Test Enable:** quando l'enable è basso il filtro si mette in uno stato di inattività che si propaga a tutti i vari sotto filtri. Di fatto bisognerà controllare che tutti i valori dei vari filtri e counter interni agli stadi vengano tenuti costanti fino alla risalita del segnale di enable.
- **Test Reset:** quando questo segnale è attivo (0 perché è un segnale del tipo attivo basso) il filtro si predispone nel modo più veloce possibile per l'analisi di nuovi dati. In questo caso si dovrà controllare che ogni registro e counter interno vengano resettati correttamente.

Per brevità si riportano solo delle annotazioni perse dall'analisi dei dati con il compilatore grafico senza riportare immagini dei segnali.

- **Enable:** il comportamento risulta nella norma con tutti i registri che mantengono costante il loro valore fino alla nuova abilitazione.
- **Reset:** lo stato dei registri e dei counter viene correttamente resettato anche se il valore d'uscita nel primo filtro non è esattamente zero ma riporta il valore del primo coefficiente del filtro. Questo comportamento si deve al fatto che l'uscita del primo filtro ridotto dipende direttamente dal valore zero in entrata, quindi uno per la nostra rappresentazione (vedi par 2.4), riportando un valore diverso da zero. Fortunatamente questo valore essendo molto basso viene annullato dai filtri successivi riportando il valore in uscita dal filtro interpolatore totale a zero.

Con questo possiamo ritenere concluso il test del filtro e possiamo passare al prossimo test sulla memoria.

3.2 Simulazione della memoria

Il funzionamento della memoria verrà testato in maniera rapida andando a memorizzare una serie di dati e a leggerli successivamente. Nell'attraversamento di una memoria i dati non subiranno modifiche, cosa che invece avviene nei filtri, risparmiando l'analisi spettrale vista precedentemente. L'unica cosa che si controllerà sarà la corretta memorizzazione dei dati andando a verificare gli ingressi e le uscite. I dati che verranno inseriti nella memoria ciclica saranno gli stessi di quelli usati per la simulazione del filtro, ma qualunque serie di milleventiquattro andrebbe bene. Nel registro invece andremo a memorizzare i seguenti dati :

- **01000000** per la parola moltiplicativa (**40** in esadecimale)
- **11110000** per la prima parola (**f0** in esadecimale)
- **00001111** per la seconda parola (**0f** in esadecimale)
- **10** per i due bit di settaggio

Di seguito potrete vedere alcune immagini estratte dalle simulazioni effettuate.

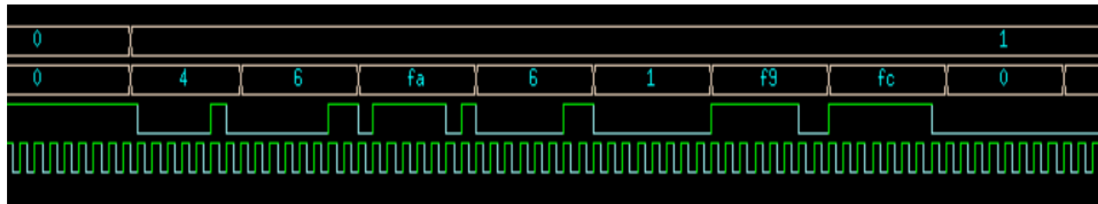


Figura 37: estratto simulazione di scrittura nella memoria ciclica

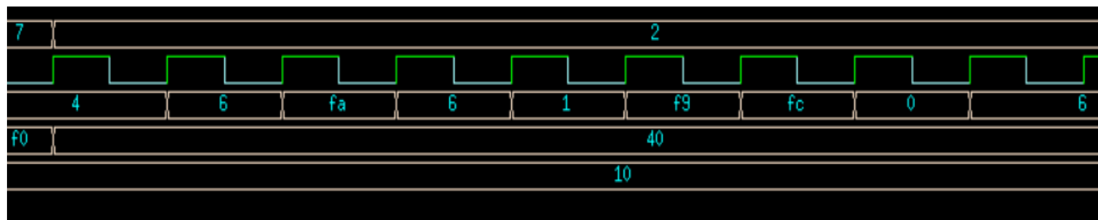


Figura 38: estratto simulazione lettura ciclica

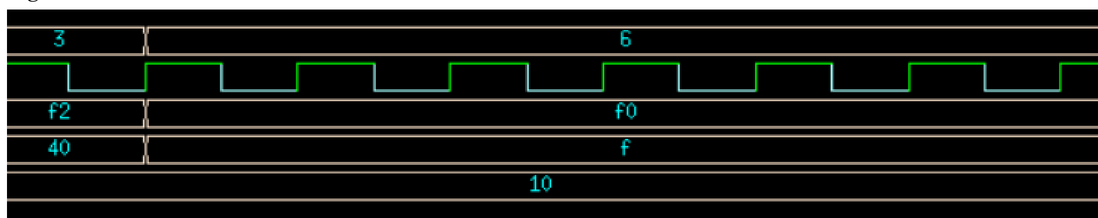


Figura 39: estratto simulazione di lettura registro

Nelle figure qui elencate il primo segnale riportato è sempre il segnale di stato da cui possiamo dedurre lo stato della memoria. Nella prima potete osservare come la serie di dati venga scomposta nel segnale d'ingresso DAT sottostante che sarà memorizzato in memoria ad ogni fronte di salita del segnale di clock. Per verificare la correttezza di queste operazioni basta fare riferimento alla seconda figura dove si vede la medesima serie di dati uscire dalla memoria. La memorizzazione del registro è stata fatta a parte seguendo la medesima procedura usata per la memoria ciclica. Anche in questo caso possiamo vedere che tutti i dati sono stati memorizzati correttamente considerando che gli ultimi tre segnali sono nell'ordine OUT1, OUT2 e SEL i cui valori abbiamo fissato precedentemente. Con questo possiamo ritenere concluso il test della memoria e passare al successivo.

3.3 Simulazione del design finale

Una volta aver appurato il buon comportamento del filtro interpolatore e della memoria possiamo passare al test del design finale. Il test sarà volto a certificare il buon comportamento collettivo delle componenti, dando già per buone le performance delle singole unità. Come nel caso della memoria si andrà prima a memorizzare una serie di dati e poi ad abilitare le uscite, attivando i filtri o il percorso diretto a seconda dello stato. I dati utilizzati saranno gli stessi usati nelle due simulazioni precedenti, per mantenere la continuità. La memorizzazione dei dati è già stata testata nella simulazione della memoria e non richiede un ulteriore test: gli ingressi della memoria sono collegati direttamente agli ingressi del design finale rendendo le due situazioni del tutto analoghe. Ai fini di semplificare l'analisi dei due filtri sarà inviata la stessa sequenza di dati ad entrambi così da ottenere lo stesso comportamento in uscita. La seguente figura mostra i segnali prodotti dalle due uscite durante la simulazione, :

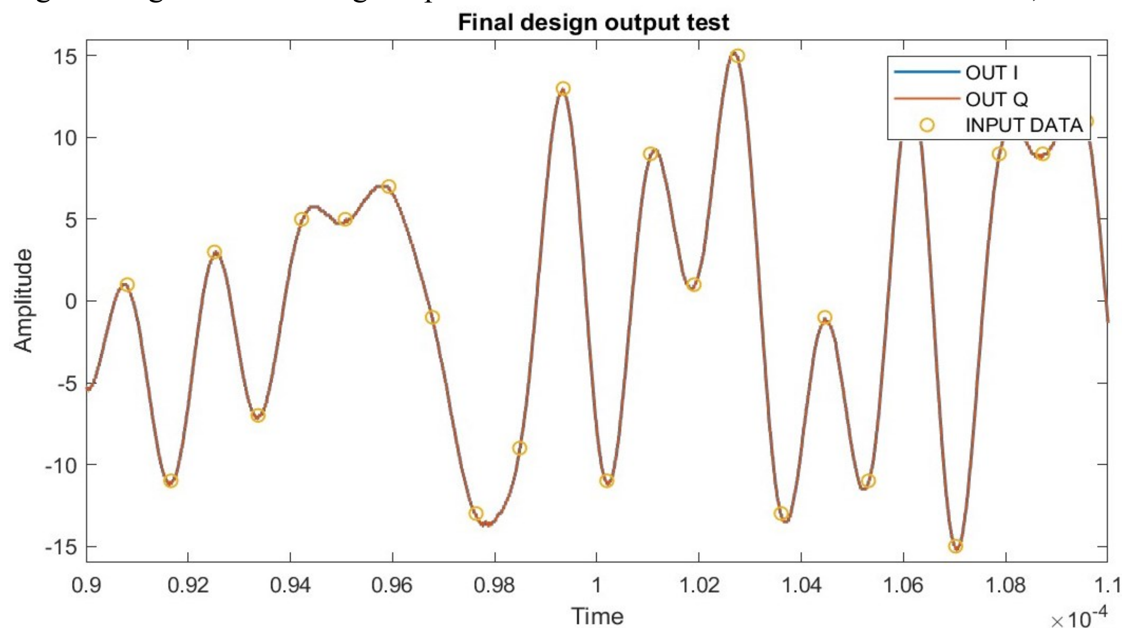


Figura 40: Test segnali di uscita OUT_I e OUT_Q con filtro interpolatore 16-2-2-2

Da come si può osservare le due uscite sono perfettamente sovrapposte indicando come i due filtri abbiano il medesimo ritardo complessivo. L'andamento è inoltre coerente con i dati ingresso dimostrando ancora una volta il corretto comportamento della memoria. La sovrapposizione con i dati in ingresso è resa possibile dal coefficiente moltiplicativo applicato (del valore di x40) che nel formato s1.6 rappresenta il valore uno e quindi non apportando modifiche alla forma d'onda finale. La simulazione a segnali costanti ha dato anch'essa i risultati desiderati ma non verrà riporta in quanto sarebbe solo un ripetersi di quanto visto nella memoria. Nel complesso le simulazioni hanno riportato il comportamento desiderato del circuito per cui possiamo concluderle e passare alla prossima fase.

4 Progettazione su Design Vision

Ora che abbiamo dei modelli vhdl funzionanti il prossimo passo sarà quello di trasporre questi modelli in una logica a standard cell chiamata anche gate-netlist, ovvero andare a identificare tutte le celle e le interconnessioni che realizzano la logica da noi ideata. Nel fare questo Synopsys ha creato un tool di sintesi logica chiamato Design Vision che prenderà i modelli VHDL e li sintetizzerà andando a creare una lista di gate. Il tool di fatto rappresenta solo una interfaccia grafica di un altro tool denominato Design Compiler. Le modalità di inserimento dei comandi sono due:

- **Command Line:** digitare i comandi direttamente dal terminale, metodo più veloce ma meno intuitivo in quanto ogni comando presenta una serie di attributi più o meno complessa difficile da ricordare.

- **Guidato:** usare le finestre di inserimento proposte da design Vision che permettono di generare dei comandi in modo interattivo con un maggiore controllo sugli attributi senza dover consultare i datasheet.

Di fatto il metodo guidato produce un comando che viene poi inserito sulla Command line. Le sintesi prodotte possono essere create con dei gates generici o attraverso delle librerie specifiche come nel nostro caso. Il diagramma seguente riporta il flusso di progettazione da seguire per Design Vision. [4] [5]

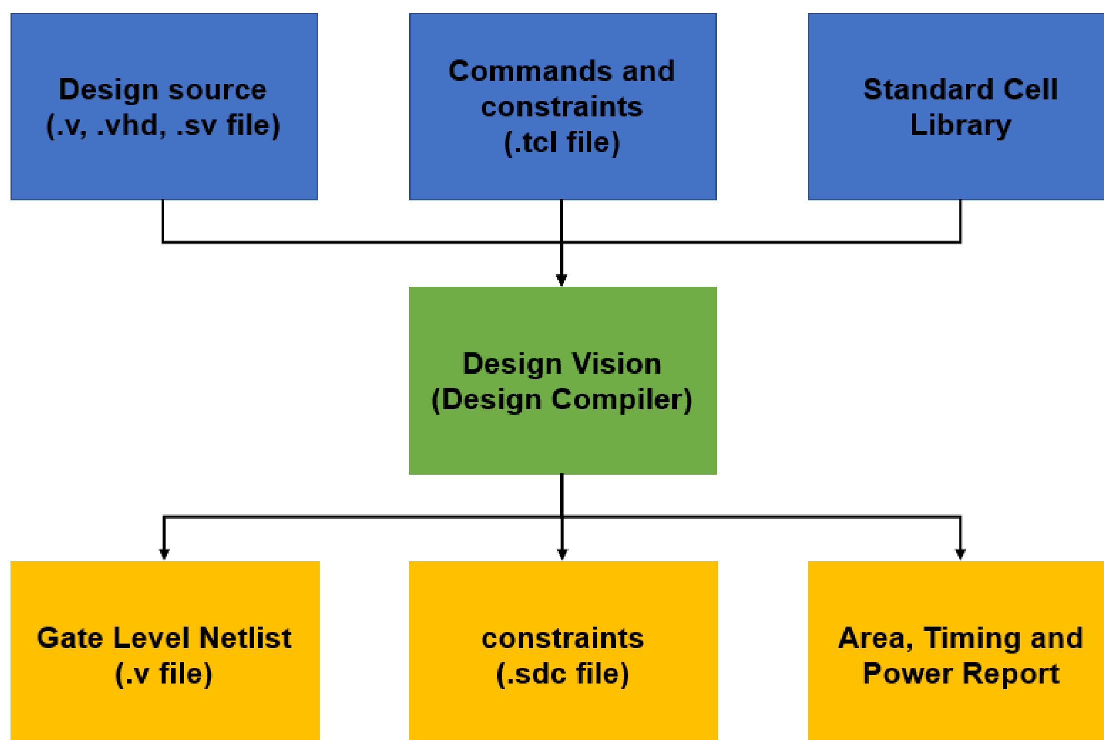


Figura 41 : modello riassuntivo di Design Vision.

Come si può vedere in questo diagramma a seguito dell'inserimento di alcuni input Design Vision permette di creare dei file (.v e .sdc) che saranno poi passati ad altri tool dove avverrà il place and route, nel nostro caso passeremo questi file a Innovus dove procederemo con la messa in posa effettiva delle componenti.

Per il completamento di questo flusso sono necessarie determinate operazioni:

1. **Analyze:** in questo passo vengono controllati i file per vedere se sono idonei per la sintesi. La sintassi per questo comando è:
   -analyze -library work -format vhdl {Lista di file HDL}
   Oppure andando su **File→Analyze→Add** e selezione i file prescelti.

2. **Elaborate:** in questo passo si crea un design dai file precedentemente analizzati ma ancora sottoponibile ad aggiustanti tipo clock, ritardi che possono essere inseriti anche successivamente. Le librerie a questo devono essere inserite altrimenti la sintesi non può cominciare. La sintassi in questo caso è: *elaborate Entity_name -architecture arch_name -library work*
   Oppure andando su **File→Elaborate→Add**

3. **Compile:** in questa fase avviene la vera compilazione del design che viene mappato in una libreria di gate o di celle. Vi sono varie opzioni a seconda delle ottimizzazioni: si può scegliere se concentrarsi sull'area, sulle prestazioni o sul consumo di potenza. Dopo questa fase si può procedere con la generazione di differenti rapporti; molto importanti quelli sui ritardi e sul consumo di area. Può essere attivato andando su **Complile→Complie_design**

4. **Save:** Dopo la compilazione il design può essere salvato in un file HDL (**File→Save as→folder_name→Format→File_name**) o di altro tipo per essere passato ad altri tool dove avverrà la messa in posa definitiva delle celle. Servirà inoltre creare un secondo file dove verranno salvate tutte le personalizzazioni riguardanti l'albero di clock e quant'altro attraverso il comando: **Write_sdc folder/file_name**.

Seguendo questa procedura andremo ad analizzare non solo il modello definitivo del chip ma anche i filtri e la memoria. In questo modo otterremo una migliore comprensione del circuito finale identificando quali sono le parti più critiche. Per esempio, avendo una stima dell'area delle componenti in caso di problemi sapremo dove andare ad intervenire per ridurne il consumo. L'analisi sarà quindi svolta prima sulle singole componenti e poi successivamente sul modello completo. Oltre a questo, sarà presente anche una piccola sezione dove andremo a simulare il consumo di area dei vari filtri alle diverse risoluzioni così da scegliere la versione definitiva. Prima di tutto questo però bisognerà andare a controllare il setup di Design Vision così da specificare nel modo più preciso possibile l'ambiente di lavoro in cui andremo ad operare

4.1 Setup utilizzato per Design Vision

Prima di procedere con le varie fasi è necessario fare un piccolo passo indietro e andare a vedere come è stato settato Design Vision. Le cose da controllare in questo caso sono le unità di misura, le librerie, la versione del prodotto utilizzato così da essere sicuri su quello che si va a scrivere evitando errori banali per una errata comprensione dei dati. Quando si utilizza Design Vision è sempre utile controllare le unità di misura preimpostate attraverso il comando **Report_units**. Di solito le unità di misura dipendono dalle librerie utilizzate quindi usare questo comando a tool appena avviato produce solo un messaggio di errore. Nella versione che sto utilizzando dopo aver elaborato il design le unità di misura sono:

- Tempo: 10e-12 s (ps)
- Capacità: 10e-15 Farad (fF)
- Resistenza: 1000 Ohm (KOhm)
- Voltaggio: 1 volt (V)
- Corrente: 10e-3 Ampere(mA)

La libreria utilizzata può essere trovata andando su **File→Setup**, in questo caso si ha:

GF22FDX_SC8T_104CPP_BASE_CSC20R_TT_0P80V_0P00V_0P00V_0P00V_25C.db

Altre cose che si possono controllare a questo punto possono essere:

- I nomi dei valori logici alto e basso:
  valore logico 0: VSS
  valore logico 1: VDD
- La versione del prodotto utilizzato: R-2020.09-SP2
- L’utilizzo di packages: IEEE.std_logic_1164

Dalla finestra di setup è inoltre possibile andare a modificare tutte le varie specifiche come le librerie, le cartelle di salvataggio dei file e molte altre variabili interne. Prestando attenzione poi all’avvio del tool vengono riportati anche gli altri prodotti Synopsis sempre associati a Design Compiler tra cui:

- Design Compiler Graphical
- DC Ultra
- DC Expert
- DFTMAX
- HDL Compiler
- VHDL Compiler
- Power Compiler

Per maggiori informazioni su questi prodotti si consiglia di consultare la guida utente di Design Vision.

4.2 Calcolo preliminare dell'area e scelta del modello

Avendo ora assodato il buon comportamento dei due filtri possiamo passare a stimare il consumo di area delle due logiche. La somiglianza delle prestazioni viste nei paragrafi precedenti rende questa fase cruciale in quanto quella con un minor consumo di area sarà anche quella che verrà usata per la messa in posa definitiva. Per semplificare l'analisi andrò a creare due file per ricalcolare l'area automaticamente una volta cambiato il valore delle variabili sui file. L'attuazione ricorsiva degli script può essere fatta andando su **File→Execute_Script** e selezionando l'apposito file. Nello svolgere queste operazioni ho però notato dei problemi quando andavo a sovrascrivere il vecchio design del filtro 16-2-2-2. Da una prima analisi sembrava che sebbene fosse riuscito a sintetizzare correttamente il filtro la prima volta, alla seconda, anche non cambiando i parametri, non riuscisse a sintetizzare tutti gli elementi lasciando di fatto dei vuoi e compromettendo il calcolo dell'area. Per sistemare ho predisposto negli script un comando apposito per cancellare i design precedentemente creati prima di procedere con una nuova analisi. Il comando utilizzato è stato:

RemoveDesign -all_design

Grazie a questo stratagemma sono riuscito a completare l'analisi riportando i seguenti dati:

Tabella riassuntiva calcolo dell'area						
Filtro Interpolatore	16-8			16-2-2-2		
Risoluzione	HR	MR	LR	HR	MR	LR
Consumo Area	4290	4235	4020	3379	3325	3113

Come era prevedibile anche dal capitolo precedente il filtro interpolatore 16-2-2-2 è quello con il minor consumo di area. Il taglio del numero dei moltiplicatori ha comportato un risparmio d'area medio piuttosto elevato, oltre 900 µm^2. Facendo ulteriori analisi ho scoperto che la dimensione minima di un moltiplicatore con in ingresso due dati a 9 e 11 bit è intorno ai 170 µm^2. Facendo il calcolo con il numero di moltiplicatori stimati le cose non tornano: il modello 16-8 dovrebbe occupare molta più area di quanta riportata nella tabella soprastante. La spiegazione di questa discrepanza risiede nei coefficienti del filtro interpolatore che essendo in molti casi composti da soli zeri possono essere risolti in logiche più semplici riducendo il consumo di area. Se avessimo dei coefficienti più complessi la discrepanza tra i due modelli sarebbe ancora più accentuata dimostrando la convenienza del modello 16-2-2-2. D'ora in poi l'analisi procederà solo per il modello 16-2-2-2.

4.3 Attribuzione dei vincoli

Ora che siamo riusciti a scegliere una delle due versioni del filtro dobbiamo riportate il set di comandi che andremo ad utilizzare per l'attribuzione dei vincoli quali Clock, ritardi e altri . La cosa più complessa da fare nel nostro caso è l'attribuzione dei clock perché molti sono derivati da altri segnali e hanno bisogno di una procedura dedicata per il loro riconoscimento. Il compilatore non è di fatto in grado di riconoscerli automaticamente, di solito se viene attribuito un clock a un determinato pin d'ingresso e questo si propaga direttamente a una sotto entità, anche l'attributo di clock viene mantenuto ma non è questo il caso. Il procedimento prevede dunque di istanziare prima i clock principali, detti Master Clock e di passare successivamente a definire i clock generati da essi. Per assegnare l'attributo di Master clock al pin d'ingresso prescelto il comando che ho utilizzato è stato:

create_clock -name "CLK150" -period 6666.666 -waveform { 0 3333.333 } { CLK }	
-name "CLK150"	Nome scelto dall'utente in questo caso si riferisce ad un clock a 150 megahertz.
-period 6666.666	Indica il periodo del clock in picosecondi. $\frac{1}{150\ 000\ 000} = 6666.6\overline{6}\ ps$
-waveform {0 3333.333 }	Serve a definire la forma dell'onda di clock, lo zero serve per posizionare il fronte di salita mente l'altro quello di discesa. In questo caso si è preso metà del tempo totale di clock per creare un clock con un duty cycle del 50%.
{ CLK }	Indica il nome del pin dove si va ad attribuire il clock appena creato.

Una volta creato il master Clock del nostro circuito si può passare alla definizione dei clock generati tramite il comando:

create_generated_clock -name CLK18 -source CLK -master_clock CLK150 -divide_by 8 FIR1/CLK	
-name "CLK18"	Nome scelto dall'utente in questo caso si riferisce ad un clock a 18.75 megahertz.
-source CLK	Indica il pin sorgente del clock che nel nostro caso si riferisce a quello definito precedentemente.
master_clock CLK150	Indica il nome del clock generante da cui si ricava frequenza e duty cycle. Nel nostro caso e quello sopracitato.
-divide_by 8	Indica il fattore con cui vado a dividere il clock generante per ottenere il clock generato

FIR1/CLK	Indica il nome del pin dove si attribuisce il clock, ora essendo una sotto entità va chiamata in questo modo un po' come le cartelle all'interno del nostro pc.

Questo comando sarà usato in più occasioni per dichiarare al compilatore la struttura dell'albero di clock, motivo per cui di seguito vi saranno due sezioni dedicate sia per il filtro che per la memoria. Altri comandi utilizzati per la modellizzazione del segnale di clock sono stati:

-set_input_delay	Indica il tempo massimo di stabilizzazione del segnale d'ingresso. Nel nostro caso il tempo per cui il dato si propaghi dalla memoria all'ingresso del filtro.
-set_output_delay	Indica il tempo minimo in cui le uscite devono rimanere stabili per essere memorizzate correttamente dal trasmettitore.
-ste_clock_uncertainty	Indica l'incertezza del segnale di clock espressa in femto secondi.

Un secondo vincolo importante da includere è la cella motrice che in questo caso sarà una cella di libreria, di solito un buffer tra i vari disponibili. Una volta scelto il buffer il comando per formalizzare l'assegnazione è:

set_driving_cell -lib_cell SC8T_BUFX1_CSC20R -library Lib_Name { {X[3]} {X[2]} {X[1]} {X[0]} RES EN }	
-lib_cell SC8T_BUFX1_CSC20R	Indica il nome della cella da usare per ricavare gli attributi delle porte d'ingresso.
-library Lib_Name	Indica il nome della libreria di riferimento, è la stessa specificata durante il setup.
{{X[3]} {X[2]} {X[1]} {X[0]} RES EN }	Indica i nomi dei pin d'ingresso dove impostare i vincoli.

L'utilizzo di questo comando permette di associare un pin della libreria con una porta d'ingresso in modo che il compilatore possa modellizzare accuratamente la capacità di pilotaggio del driver esterno. Passando alle uscite bisogna attribuire le capacità di carico che dipendono dalle porte a cui verranno connesse. Dalle specifiche fornite la capacità attesa del DAC sarà intorno ai 20 femto Farad. Il comando utilizzato in questo caso sarà:

set_load 20 [all_outputs]

Il comando si applicherà solo ai segnali di tipo dinamico e non a quelli costanti come il segnale utilizzato per il settaggio dei bit che rimane stabile per la maggior parte del tempo.[1][4][5]

4.3.1 Vincoli utilizzati nel Filtro 16-2-2-2

Nel definire i vincoli del filtro bisogna ricordarsi che nel nostro design finale gli ingressi e le uscite saranno collegate a componenti interne. I comandi per settare i carichi delle uscite o la cella motrice sono superflui. Tuttavia, manterremo questi vincoli per evitare l'insorgere di errori nel compilatore visto che non dovrebbero produrre variazioni significative. Come anticipato precedentemente la dichiarazione dell'albero di clock è piuttosto complessa per il filtro considerando l'albero di clock che potete vedere nel paragrafo 2.2.6. Per verificare che l'attribuzione dei clock fatta mediante i comandi sia andata a buon fine ho lanciato un *report_clock* :

REPORT CLOCKS FILTRO 16-2-2-2					
CLOCK	**Period(fs)**	**Waveform**	**Attr.**	**Master Clock**	**Waveform modification**
CLK1.7	853333.25	{0 426667}	G	CLK18	Diveded_by(16)
CLK18	53333.33	{0 26666.66}	G	CLK150	Diveded_by(8)
CLK18_1	53333.33	{0 26666.66}	G	CLK37	Diveded_by(2)
CLK37	26666.66	{0 13333.33}	G	CLK150	Diveded_by(4)
CLK37_1	26666.66	{0 13333.33}	G	CLK75	Diveded_by(2)
CLK75	13333.33	{0 6666.67}	G	CLK150	Diveded_by(2)
CLK75_1	13333.33	{0 6666.67}	G	CLK150	Diveded_by(2)
CLK150	6666.67	{0 3333.33}	//		

Con riferimento al comando *create_generated_clock* visto precedentemente nella tabella soprastante mancano le informazioni riguardanti la sorgente a cui andrà attribuito il clock; questo perché dipende dai nomi dati ai vari filtri in fase di progettazione e non avrebbe senso riportarli senza riportare anche il codice stesso. Con riferimento alla figura 21 do una breve descrizione dei clock visti qui sopra:

- **CLK1.7** : prodotto dal primo stadio e propagato ai sedici filtri ridotti.
- **CLK18** : prodotto nella top entity e propagato al primo stadio
- **CLK18_1** : prodotto nel secondo stadio e propagato ai due filtri ridotti
- **CLK37** : prodotto nella top entity e propagato al secondo stadio.
- **CLK37_1** : prodotto nel terzo stadio e propagato ai due filtri ridotti.
- **CLK75** : Prodotto nella top entity e propagato nel terzo stadio.
- **CLK75_1** : Prodotto nel quarto stadio e propagato ai due filtri ridotti.
- **CLK150** : fornito dal trasmettitore è il master da cui si ricavano tutti gli altri segnali di clock, viene fornito direttamente al quarto stadio.

Oltre a questo, verrà attribuita una incertezza di cento pico secondi al clock principale a simulare una situazione reale.

4.3.2 Vincoli utilizzati nella memoria

Come nel caso precedente manterremmo i vincoli per il carico e la cella motrice anche se non necessari. Nel settaggio dei clock bisogna prestare molta attenzione in quanto nello stesso circuito vengono impiegati due clock di sorgenti diverse. La soluzione più immediata sarebbe quella di testare tutto il circuito usando come riferimento il clock più veloce. Nel caso generale quando ogni clock gestisce un percorso diverso usare su tutti quello più veloce non sarebbe l'ideale: il clock veloce potrebbe non funzionare sul percorso del clock lento. Tuttavia, riconoscendo la semplicità della logica possiamo escludere questa eventualità e testare l'intero circuito con la frequenza massima di 1.7 megahertz. Una frequenza così bassa però non permette al compilatore di esprimere le massime potenzialità del circuito, motivo per cui si faranno due analisi con frequenze di clock diverse: una a 1.7 e una a 150 megahertz. La seconda è stata scelta perché è la frequenza massima del design finale ma si potrebbe prendere una anche maggiore. Da queste simulazioni potremo ricavare molte informazioni circa il comportamento del circuito nei diversi ambienti. Andiamo ora a vedere l'attribuzione finale dei clock ottenuta mediante il comando *report_clock:*

REPORT CLOCKS MEMORIA					
CLOCK	**Period(fs)**	**Waveform**	**Attr.**	**Master Clock**	**Waveform modification**
CLK_INT	6666.67	{0 3333.33}	//		
CLK_CYC	6666.67	{0 3333.33}	G	CLK_INT	Diveded_by(1)
CLK_EXT	666666.69	{0 333333}	//		
CLK_REG	6666.67	{0 3333.33}	G	CLK_INT	Diveded_by(1)

Il risultato sarà lo stesso anche a bassa frequenza cambieranno soltanto i valori relativi al periodo e alla forma d'onda del clock interno. In questo caso i clock generati mantengono la frequenza e la forma originaria essendo divisi per uno. Lascio di seguito un breve commento circa i clock elencati in tabella:

- **CLK_INT** : segnale di clock in ingresso derivante internamente dal trasmettitore, nel circuito finale sarà quello ad alta frequenza(1,7 MHz).
- **CLK_CYC** : segnale di clock della memoria ciclica viene alternato tra i due clock in ingresso a seconda dello stato imposto.
- **CLK_EXT** : segnale di clock in ingresso fornito da una sorgente esterna al chip e a frequenza minore di quella interna.
- **CLK_REG** : segnale di clock del registro verrà usato principalmente per caricare dati nel registro.

Anche in questo caso attribuiremo una incertezza di cento picosecondi ai clock principali.

4.3.3 Vincoli utilizzati nel design finale

Nel design finale verranno usati tutti i vincoli visti precedentemente: cella motrice sui segnali d'ingresso (DAT, ST) e settaggio di un carico di venti femto Farad sulle uscite. La definizione dell'albero di clock si farà ancora più complessa per l'aumento sostanziale dei clock da generare avendo ben due filtri e una memoria. La tabella seguente riporta tutti i clock utilizzati nella logica:

REPORT CLOCKS DESIGN FINALE					
CLOCK	**Period(fs)**	**Waveform**	**Attr.**	**Master Clock**	**Waveform modification**
CLK1_7I	853333.25	{0 426667}	G	CLK18I	Diveded_by(16)
CLK1_7Q	853333.25	{0 426667}	G	CLK18Q	Diveded_by(16)
CLK1_7M	853333.25	{0 426667}	G	CLK150	Diveded_by(128)
CLK18I	53333.33	{0 26666.66}	G	CLK150	Diveded_by(8)
CLK18Q	53333.33	{0 26666.66}	G	CLK150	Diveded_by(8)
CLK18_1I	53333.33	{0 26666.66}	G	CLK37I	Diveded_by(2)
CLK18_1Q	53333.33	{0 26666.66}	G	CLK37Q	Diveded_by(2)
CLK37I	26666.66	{0 13333.33}	G	CLK150	Diveded_by(4)
CLK37Q	26666.66	{0 13333.33}	G	CLK150	Diveded_by(4)
CLK37_1I	26666.66	{0 13333.33}	G	CLK75I	Diveded_by(2)
CLK37_1Q	26666.66	{0 13333.33}	G	CLK75Q	Diveded_by(2)
CLK75I	13333.33	{0 6666.67}	G	CLK150	Diveded_by(2)
CLK75Q	13333.33	{0 6666.67}	G	CLK150	Diveded_by(2)
CLK75_1I	13333.33	{0 6666.67}	G	CLK150	Diveded_by(2)
CLK75_1Q	13333.33	{0 6666.67}	G	CLK150	Diveded_by(2)
CLK_CYC	853333.25	{0 426667}	G	CLK1_7M	Diveded_by(1)
CLK_REG	853333.25	{0 426667}	G	CLK1_7M	Diveded_by(1)
CLK_EXT	6666666	{0 3333333}	//		
CLK150	6666.67	{0 3333.33}	//		

Questi non sono altro che la somma dei vari clock visti precedentemente con l'aggiunta di un clock a 1.7 megahertz (CLK1_7M) che rappresenta il CLK_INT del caso precedente. Molti di questi hanno una lettera per identificare a quale filtro appartengano: I per immaginario e Q per reale. Oltre a questo, per simulare delle non linearità nei segnali di clock verrà attribuita al segnale di clock principale una incertezza di cento picosecondi.

4.4 Report

Una volta che si avranno applicato tutti i vincoli ritenuti necessari possiamo lanciare la compilazione della logica e passare all'analisi dei report. La compilazione in questo caso è stata eseguita con le opzioni di default: non avendo esigenze particolari ho lasciato i livelli di sforzo sui valori preimpostati presenti nella finestra di configurazione. In caso di problemi si potrà sempre rilanciare una nuova compilazione aumento lo sforzo nelle parti più critiche. L'analisi si focalizzerà sull'esito di questi tre report:

- **Report Area:** uno dei report più importanti e maggiormente utilizzati, permette di ottenere una stima approssimativa sul consumo d'area del circuito. Nei sistemi eterogenei, ovvero con diversi circuiti logici presenti nello stesso chip di silicio, gli spazi sono spesso risicati dando ai progettisti poca libertà circa le dimensioni finali del design. Un circuito che già in questa fase non si mostra conforme con le specifiche dovrà essere riprogettato in modo da diminuirne le dimensioni. Se l'eccedenza non è molta si può anche provare a rilanciare la compilazione aumentando lo sforzo del compilatore per rientrare nelle specifiche. Il report Area era già stato usato precedentemente per determinare il consumo d'area dei due filtri alle varie risoluzioni, in quel caso era stato riportato solo il dato riguardante l'occupazione complessiva di area dovuta alle standard cell. Nelle tabelle seguenti invece saranno riportati tutti i dati generati dal report. Il report fornirà il numero di celle impiegate suddiviso per tipologia e il loro consumo d'area. Il risultato che si ottiene dipende fortemente da come il compilatore interpreterà il nostro codice; per esempio, una parte che noi ipotizziamo puramente combinatoria potrebbe essere tradotta con una memoria occupando un'area maggiore. Controllare questi dati è fondamentale per capire se il compilatore ha interpretato il codice in modo corretto e se non ci fa capire dove andare ad intervenire.

- **Report Power:** il consumo di potenza sta diventando un aspetto sempre più cruciale nei nuovi dispositivi mobili dove la fonte di alimentazione principale è fornita da una batteria. Nel tentativo di estendere al massimo il tempo di utilizzo si sta cercando di ridurre sempre più il consumo di potenza e laddove le tecnologie non riescono ancora a spingersi bisogna porvi rimedio andando a modificare la logica alla base. Per esempio, nel filtro che abbiamo sviluppato gli stadi lavorano a frequenze inferiori rispetto all'uscita minimizzando l'utilizzo dei registri e quindi anche il consumo di potenza. Se avessimo fatto un singolo stadio operante alla frequenza massima la potenza dissipata sarebbe di gran lunga maggiore. Lanciando il report Power si otterranno delle stime sul consumo di potenza delle diverse tipologie di celle. Questo servirà in caso di modiche a indirizzare l'operato del progettista nelle parti più critiche. Di solito quando si cerca di ridurre il consumo di potenza le soluzioni non sono semplici e portando a inevitabili compromessi tra costi e prestazioni; in via generale alte prestazioni significa spesso alti consumi di potenza e viceversa.

- **Report Timing:** permette di andare a stimare il tempo necessario per la propagazione di un dato da un punto del circuito ad un altro. Quando in seguito ad un fronte di clock le entrate di una data porzione di logica vengono cambiate, ci vorrà del tempo affinché queste si propaghino e producano il corretto segnale di uscita. Per stimare questi tempi viene usata una simulazione di tipo Montecarlo che va a prendere in esame diversi percorsi e ne calcola lo slack. Lo slack è definito come la differenza tra il tempo massimo stabilito dai vincoli e quello di propagazione calcolato sul circuito; essenzialmente indica l'avanzo di tempo utile. Nel nostro caso il tempo che deve rispettare dipende dal clock utilizzato e gli altri eventuali vincoli come ritardi o incertezze. L'esito di queste analisi può dipendere anche da un singolo percorso che non rispetta la condizione portando l'intero circuito a lavorare a frequenze minori. Fortunatamente questo non è il nostro caso visto che stiamo lavorando a frequenze basse rispetto a quelle che potremmo ambire con questa tecnologia. Nel report timing vengono riportati i percorsi con un maggiore ritardo elencando l'intera lista di celle attraversate. Nel report di ogni percorso si potranno distinguere tre fasi: calcolo del tempo di propagazione nel circuito, calcolo del tempo massimo di propagazione e calcolo dello slack.Per brevità riporterò il report timing solo del percorso più lento tralasciando gli altri che verranno generati.

Come fatto anche per i vincoli ci saranno tre sezioni dedicate a ognuna delle tre logiche prese in esame dove verranno riportati i report seguiti da un breve commento.

4.4.1 Report del Filtro interpolatore

Report Area

Report Area Filtro interpolatore 16-2-2-2	
Number of ports:	3064
Number of nets	8044
Number of cells:	4393
Number of combinational cells:	3861
Number of sequential cells:	456
Number of macros/black boxes:	0
Number of buf/inv:	639
Number of references:	7
Combinational area:	2548.7155
Buf/Inv area:	91.653
Non-combinational area:	754.0582
Macro/Black Box area:	0.0000
Net Interconnect area:	undefined
Total cell area:	3302.773765
Total area:	undefined

Dai dati in tabella possiamo notare che il consumo totale dovuto alle standard cell è minore di quanto riportato nel calcolo preliminare. La motivazione può essere dovuta al fatto che nel primo caso non erano stati settati i segnali di clock. La frequenza dei clock è relativamente bassa per la tecnologia utilizzata dando la possibilità di scegliere implementazioni più lente ma con un minor consumo degli spazi. L'area totale è indefinita perché sarà decisa definitivamente solo nella fase di place and route.

Report Power

La seguente tabella riporta i dati ottenuti lanciando un report power dopo aver compilato il filtro interpolatore 16-2-2-2.

Report Power Filtro interpolatore 16-2-2-2				
Power Group	Internal Power	Switching Power	Leakage Power	Total Power
IO_pad	0.000	0.000	0.000	0.000 (0.0%)
Memory	0.000	0.000	0.000	0.000 (0.0%)
Black Block	0.000	0.000	0.000	0.000 (0.0%)
Clock Netwotk	0.000	0.000	0.000	0.000 (0.0%)
Register	25.031	7.8683e-3	5.2578	31.076 (38.04%)
Sequential	4.1802	1.8470e-3	0.6689	5.0338 (6.16%)
Combinational	6.4702	13.850	25.2711	45.591 (55.8%)
Total	35.681 µW	14.821 µW	31.1978 µW	81.700 µW
Total Dynamic Power			50.5026 µW	
Total Leakage Power			31.1983 µW	

Dalla tabella si capisce come il consumo di potenza sia dovuto principalmente alla logica combinatoria. Complessivamente un consumo dinamico di cinquanta micro-Watt è più che ragionevole considerando che il consumo del trasmettitore è stimato in qualche decina di milliwatt. In una precedente analisi fatta senza l'utilizzo dei registri tra uno stadio e l'altro era emerso un consumo di potenza addirittura maggiore, di circa settantuno microwatt, dimostrando ancora la convenienza nell'utilizzo dei registri

Report Timing

Report Timing Filtro interpolatore 16-2-2-2				
POINT	CELL	INCR	PATH	
clock CLK75_1 (rise edge)		0.00	0.00	
clock network delay (ideal)		0.00	0.00	
FIR2/FIR3/FIR1/COM2/Y2_reg[3]/CLK	(SC8T_DFFQX1_CSC20R)	0.00	0.00	r
FIR2/FIR3/FIR1/COM2/Y2_reg[3]/Q	(SC8T_DFFQX1_CSC20R)	42.47	42.47	f
FIR2/FIR3/FIR1/COM2/Y2[3]	(C2_2S_I_B1_1)	0.00	42.47	f
FIR2/FIR3/FIR1/COM1/X2[3]	(C1_2S_I_B1_1)	0.00	42.47	f
FIR2/FIR3/FIR1/COM1/add_23/A[3]	(C1_2S_I_B1_1_DW01_add_0_DW01_add_2)	0.00	42.47	f
FIR2/FIR3/FIR1/COM1/add_23/U8/Z	(SC8T_OAI21X0P5_CSC20R)	9.99	52.46	r
FIR2/FIR3/FIR1/COM1/add_23/U7/Z	(SC8T_AO21IAX0P5_CSC20R)	12.31	64.78	f
FIR2/FIR3/FIR1/COM1/add_23/U6/Z	(SC8T_OAI21X0P5_CSC20R)	12.10	76.87	r
FIR2/FIR3/FIR1/COM1/add_23/U5/Z	(SC8T_AO21IAX0P5_CSC20R)	12.47	89.34	f
FIR2/FIR3/FIR1/COM1/add_23/U4/Z	(SC8T_OAI21X0P5_CSC20R)	12.12	101.46	r
FIR2/FIR3/FIR1/COM1/add_23/U3/Z	(SC8T_AO21IAX0P5_CSC20R)	12.47	113.93	f
FIR2/FIR3/FIR1/COM1/add_23/U2/Z	(SC8T_OAI21X0P5_CSC20R)	12.12	126.06	r
FIR2/FIR3/FIR1/COM1/add_23/U1/Z	(SC8T_AO21IAX0P5_CSC20R)	15.61	141.67	f
FIR2/FIR3/FIR1/COM1/add_23/U1_7/CO	(SC8T_FAX1_A_MR_CSC20R)	29.90	171.57	f
FIR2/FIR3/FIR1/COM1/add_23/U1_8/CO	(SC8T_FAX1_A_MR_CSC20R)	27.91	199.49	f
FIR2/FIR3/FIR1/COM1/add_23/U1_9/CO	(SC8T_FAX1_A_MR_CSC20R)	27.90	227.39	f
FIR2/FIR3/FIR1/COM1/add_23/U1_10/CO	(SC8T_FAX1_A_MR_CSC20R)	27.90	255.29	f
FIR2/FIR3/FIR1/COM1/add_23/U1_11/CO	(SC8T_FAX1_A_MR_CSC20R)	27.90	283.19	f
FIR2/FIR3/FIR1/COM1/add_23/U1_12/CO	(SC8T_FAX1_A_MR_CSC20R)	27.90	311.09	f
FIR2/FIR3/FIR1/COM1/add_23/U1_13/CO	(SC8T_FAX1_A_MR_CSC20R)	27.90	338.99	f
FIR2/FIR3/FIR1/COM1/add_23/U1_14/CO	(SC8T_FAX1_A_MR_CSC20R)	27.90	366.89	f
FIR2/FIR3/FIR1/COM1/add_23/U1_15/CO	(SC8T_FAX1_A_MR_CSC20R)	27.90	349.79	f
FIR2/FIR3/FIR1/COM1/add_23/U1_16/CO	(SC8T_FAX1_A_MR_CSC20R)	24.26	419.04	f
FIR2/FIR3/FIR1/COM1/add_23/U1_17/Z	(SC8T_XOR3X1_MR_CSC20R)	51.24	470.29	r
FIR2/FIR3/FIR1/COM1/add_23/SUM[17]	(C1_2S_I_B1_1_DW01_add_0_DW01_add_2)	0.00	470.29	f
FIR2/FIR3/FIR1/COM1/Y2[10]	(C1_2S_I_B1_1)	0.00	470.29	r
FIR2/FIR3/FIR1/Y[10]	(F41_I_B1)	0.00	470.29	r
FIR2/FIR3/U5/Z	(SC8T_AOI22X1_CSC20R)	10.29	480.58	f
FIR2/FIR3/U4/Z	(SC8T_INVX1_MR_CSC20R)	75.21	555.78	r
FIR2/FIR3/Y[10]	(F4_top_I_B1)	0.00	555.78	r
FIR2/Y[10]	(F2over_TOP)	0.00	555.78	r
Y[10] (out)		0.00	555.78	r
data arrival time			555.78	
clock CLK150 (rise edge)		6666.67	6666.67	
clock network delay (ideal)		0.00	6666.67	
clock uncertainty		-100.00	6566.67	
data required time			6566.67	
Data required time			6566.67	
data arrival time			-555.78	
SLACK (MET)			6010.88	

Come era previsto il circuito non ha grossi problemi con le tempistiche riportando uno slack piuttosto alto, già così potrebbe funzionare ad una frequenza superiore al gigahertz. Il risultato è stato possibile soprattutto grazie all'utilizzo dei registri tra uno stadio e l'altro, interrompendo il percorso diretto che andava dall'ingresso all'uscita. In una precedente analisi svolta senza l'utilizzo dei registri il risultato era stato di 4126.4, di molto inferiore a quello trovato in questo caso.

4.4.2 Report della Memoria

Report Area

I risultati del report sono stati gli stessi anche variando il segnale di clock dimostrandosi invarianti rispetto alla frequenza del circuito.

Report Area Filtro interpolatore 16-2-2-2	
Number of ports:	20537
Number of nets	46411
Number of cells:	26919
Number of combinational cells:	17675
Number of sequential cells:	8218
Number of macros/black boxes:	0
Number of buf/inv:	9422
Number of references:	13
Combinational area:	4560.558092
Buf/Inv area:	1264.706563
Non-combinational area:	12580.771629
Macro/Black Box area:	0.0000
Net Interconnect area:	undefined
Total cell area:	17141.329722
Total area:	undefined

Dai dati in tabella possiamo vedere come le celle sequenziali occupino l'area maggiore rispetto alle altre tipologie seppur presenti in numero minore. Per enfatizzare questa caratteristica andiamo a calcolare l'occupazione media di una cella:

$$A_{M1comb} = \frac{4560.56}{17675} = 0.25\ um^2$$

$$A_{M1buf} = \frac{1264.7}{9422} = 0.134\ um^2$$

$$A_{M1seq} = \frac{12580.77}{8218} = 1.53\ um^2$$

Il risultato è che una cella sequenziale occuperà mediamente un'area sei volte maggiore di una cella combinatoria e più di dieci volte rispetto a un buffer/inverter. La relazione era simile anche nel report del filtro. Inoltre, il numero di celle sequenziali è pari al numero totale di singoli registri presenti nella memoria evidenziando un rapporto diretto tra i due. L'occupazione media di una cella sequenziale potrebbe dunque essere persa come l'area di un singolo registro. Questo mostra come l'utilizzo dei registri comporti un maggiore consumo d'area rispetto ad altre tipologie di strutture.

Report Power

Come era prevedibile i risultati dipendono molto dalla frequenza del clock che andiamo ad utilizzare. Nelle tabelle seguenti sono riportati i dati ottenuti dalle simulazioni.

Report Power Memoria CLK 1.7 MHz				
Power Group	Internal Power	Switching Power	Leakage Power	Total Power
IO_pad	0.000	0.000	0.000	0.000 (0.0%)
Memory	0.000	0.000	0.000	0.000 (0.0%)
Black Block	0.000	0.000	0.000	0.000 (0.0%)
Clock Netwotk	-7.2611e-02	0.75	3.3236e-2	0.7493(80.7%)
Register	2.4742e-02	3.6390e-3	109.0078	0.1338 (14.41%)
Sequential	0.000	0.000	0.000	0.000 (0.0%)
Combinational	1.2281e-4	3.6241e-4	44.9091	4.5394e-2 (4.89%)
Total	2.414e-2 mW	0.7504 mW	153.95 µW	0.9285 mW
Total Dynamic Power			774.5513 µW	
Total Leakage Power			153.9501 µW	
Report Power Memoria CLK 150 MHz				
Power Group	Internal Power	Switching Power	Leakage Power	Total Power
IO_pad	0.000	0.000	0.000	0.000 (0.0%)
Memory	0.000	0.000	0.000	0.000 (0.0%)
Black Block	0.000	0.000	0.000	0.000 (0.0%)
Clock Netwotk	-9.2948e-02	96.0018	3.3227e-2	95.91(96.59%)
Register	3.167	4.6580e-3	109.0078	3.2806 (3.3%)
Sequential	0.000	0.000	0.000	0.000 (0.0%)
Combinational	1.5720e-2	4.6389e-2	44.9091	0.107 (0.11%)
Total	3.09 mW	96.053 mW	153.95 µW	99.296 mW
Total Dynamic Power			99.1426 mW	
Total Leakage Power			153.9501 µW	

Dai risultati appare evidente come la potenza dinamica dipenda linearmente dalla frequenza, dimostrando l'inefficienza delle memorie ad alte frequenze. Facendo un rapido confronto il consumo dei due filtri e della memoria sarà circa un milliwatt ottenendo in uscita la stessa frequenza di 150 megahertz. Questa è una delle principali motivazioni che ci spinge a sviluppare logiche basate sui filtri interpolatori limitando l'utilizzo delle memorie.

Report Timing

Il report ha generato gli stessi risultati in entrambi gli scenari, per testarne i limiti dovremo aumentare ulteriormente la frequenza ma questo esulerebbe dallo scopo di questa analisi. Di seguito riporto i risultati ottenuti con la simulazione ad alta frequenza dove si possono apprezzare meglio i numeri.

Report Timing Memoria				
POINT	CELL	INCR	PATH	
clock CLK_REG (rise edge)		0.00	0.00	
clock network delay (ideal)		0.00	0.00	
REG26/R_reg[0]/CLK	SC8T_DFFQX1_CSC20R	110.96	110.96	r
REG26/R_reg[0]/Q	SC8T_AOI22X1_CSC20R	29.89	140.86	f
REG26/U29/Z	SC8T_INVX1_MR_CSC20R	11.71	152.57	r
REG26/R_reg[1]/D	SC8T_DFFQX1_CSC20R	0.00	152.57	r
data arrival time			152.57	
clock CLK_REG(rise edge)		6666.67	6666.67	
clock network delay (ideal)		0.00	6666.67	
REG26/R_reg[1]/CLK	SC8T_DFFQX1_CSC20R	0.00	6666.67	r
library setup time		-16.79	6649.88	
data required time			6649.88	
Data required time			6649.88	
Data arrival time			-152.57	
SLACK (MET)			6497.31	

Come possiamo osservare dai risultati per una logica così semplice non ci sono grossi problemi sulle tempistiche ottenendo un slack molto alto. Il circuito già così com'è potrebbe funzionare ad una frequenza superiore ai cinque gigahertz. La notizia è positiva perché una volta piazzato nel circuito finale non influirà più di molto sulle tempistiche generali.

4.4.3 Report del design finale

Report Area

Conoscendo l'occupazione d'area della memoria e dei filtri dai report precedenti possiamo andare a stimare grossolanamente il consumo totale del design finale.

$$A_{MEM} + 2 * A_{FIR} = \ 17141.33 + 2 * 3302.77 = 23746.87$$

Da come si può notare la parte più gravosa in termini spaziali è la memoria che occuperà più del settanta percento dell'area finale. Gli ulteriori contributi data dalla logica di controllo e dai due moltiplicatori sono di poco conto rispetto alle due entità maggiori. Con queste considerazioni andiamo ad analizzare il report finale.

Report Area Final Design	
Number of ports:	26887
Number of nets	63358
Number of cells:	36269
Number of combinational cells:	25946
Number of sequential cells:	9137
Number of macros/black boxes:	0
Number of buf/inv:	10783
Number of references:	14
Combinational area:	9983.2678
Buf/Inv area:	1459.1283
Non-combinational area:	14099.6
Macro/Black Box area:	0.0000
Net Interconnect area:	undefined
Total cell area:	24082.87
Total area:	undefined

L'area totale calcolata nel report è perfettamente coerente con quella stimata. Analizzando anche gli altri dati riportati si può notare come pure loro rispettino la stessa espressione usate per il calcolo della memoria. Nel complesso ci possiamo ritenere soddisfatti da quanto emerso affermando che il modello rispetta le nostre aspettative. L'unico elemento che dovrebbe essere migliorato è la memoria che risulta un po' troppo dispendiosa in termini di area. Si consiglierebbe di testare altri modelli di memoria, anche più lente ma con ridotti consumi di spazio.

Report Power

Come fatto nel caso precedente andiamo a stimare il consumo di potenza del design finale:

$$P_{MEM} + 2 * P_{FIR} = \ 928.5 + 2 * 81.7 = 1091.9\ \mu W$$

La memoria anche in questo caso si mostra come la componente più dispendiosa responsabile di circa l'ottantacinque percento del consumo di potenza. Andiamo di seguito ad analizzare i risultati del report.

Report Power Final Design				
Power Group	Internal Power	Switching Power	Leakage Power	Total Power
IO_pad	0.000	0.000	0.000	0.000 (0.0%)
Memory	0.000	0.000	0.000	0.000 (0.0%)
Black Block	0.000	0.000	0.000	0.000 (0.0%)
Clock Netwotk	-6.933e-1	750	3.4478e-2	749.4(66.7%)
Register	75.956	5.266e-1	118.2	194.7 (17.4%)
Sequential	7.8623	1.8665e-1	1.261	9.31 (0.8%)
Combinational	14.532	57.828	97.2	169.6 (15.2%)
Total	97.66e μW	808.6 μW	216.7 μW	1122.9 μW
Total Dynamic Power			906.22 μW	
Total Leakage Power			216.7 μW	

Il consumo totale rispetta le nostre stime e possiamo notare una certa coerenza anche degli altri dati. Per esempio, la potenza di switching del clock network è la stessa di quella trovata nei report della memoria. Nel complesso i risultati soddisfano le aspettative e sono comunque inferiori rispetto al consumo del trasmettitore. Dai report precedenti abbiamo visto come la memoria prodotta da Design Vision possa lavorare a frequenze molto alte e questa sua versatilità può risultare problematica. Se prendessimo una memoria progettata per un range di frequenze inferiori si riuscirebbe ad ottenere un minor consumo di potenza. Per la nostra applicazione è superfluo usare una memoria troppo sofisticata quando potremo usarne altre meno veloci.

Report Timing

Il calcolo dei ritardi nei circuiti è un argomento piuttosto complesso da calcolare in quanto si basa su delle simulazioni di tipo Montecarlo. Per stimare lo slack possiamo solo basarci sui report precedenti e ipotizzare che il percorso peggiore venga mantenuto, se non ulteriormente peggiorato. Lo slack atteso sarà quindi qualcosa di simile a quanto visto nel Filtro interpolatore. Andiamo quindi ad analizzare il percorso peggiore rilevato del circuito.

Report Timing Filtro interpolatore 16-2-2-2			
POINT	CELL	INCR	PATH
clock CLK150 (rise edge)		0.00	0.00
clock network delay (ideal)		0.00	0.00
Count_reg[0]/CLK	(SC8T_DFFQX1_CSC20R)	0.00	0.00 r
Count_reg[0]/Q	(SC8T_DFFQX1_CSC20R)	38.26	38.26 r
add_39/A[0]	(Final_Design_DW01_inc_0_DW01_inc_2)	0.00	38.26 r
add_39/U1_1_1/CO	(SC8T_HAX1_MR_CSC20R)	19.57	57.83 r
add_39/U1_1_2/CO	(SC8T_HAX1_MR_CSC20R)	18.85	76.68 r
add_39/U1_1_3/CO	(SC8T_HAX1_MR_CSC20R)	18.85	95.53 r
add_39/U1_1_4/CO	(SC8T_HAX1_MR_CSC20R)	18.85	114.39 r
add_39/U1_1_5/CO	(SC8T_HAX1_MR_CSC20R)	18.74	133.12 r
add_39/U2/Z	(SC8T_XOR2X0P5_CSC20R)	15.81	148.93 r
add_39/SUM[6]	(Final_Design_DW01_inc_0_DW01_inc_2)	0.00	148.93 r
U85/Z	(SC8T_AN2X1_MR_CSC20R)	17.65	166.58 r
Count_reg[6]	(SC8T_DFFQX1_CSC20R)	0.00	166.58 r
data arrival time			166.58
clock CLK150 (rise edge)		6666.67	6666.67
clock network delay (ideal)		0.00	6666.67
clock uncertainty		-100.00	6566.67
Count_reg[6]/CLK	(SC8T_DFFQX1_CSC20R)	0.00	6566.67 r
library setup time		-16.10	6550.57
data required time			6550.57
Data required time			6550.57
data arrival time			-166.58
SLACK (MET)			6383.99

I risultati evidenziano uno slack migliore di quello ottenuto nel filtro interpolatore. La cosa è sicuramente positiva ma viene da chiedersi come questo sia possibile. Conoscendo le dinamiche di una simulazione di tipo Montecarlo ho provato a cercare il percorso precedente manualmente per calcolarne il ritardo ma senza successo. Il percorso non esisteva più così sono andato a guardare direttamente dentro le due logiche scoprendo che vi erano delle componenti diverse; i registri e gli adder avevano una forma e una interfaccia diversa. A fronte di questo posso ipotizzare che il compilatore abbia corretto automaticamente il percorso risolvendo e velocizzando la logica. Una soluzione certa è di difficile individuazione, tuttavia alle frequenze a cui verrà utilizzato di sicuro non avrà problemi.

4.5 Conclusioni sui report

Nel complesso i risultati emersi dai report sono positivi e ci permetterebbero di passare direttamente alla fase successiva di Place and Route. Tuttavia, la memoria utilizzata si è dimostra poco adatta per il tipo di circuito che stiamo progettando:

- **Occupazione d'area:** oltre 70%
- **Consumo di potenza:** oltre 80%
- **Frequenza di funzionamento:** oltre i 5 GHz

Le prestazioni sono fin troppo esose e i consumi troppo elevati da risultare sovradimensionata per il circuito che abbiamo progettato. La soluzione a questo punto sarebbe quella di andare a sostituire la memoria con una di libreria. Le memorie raramente vengono sviluppate con Design Vision in quanto non si riesce ad avere un controllo diretto sulle prestazioni: le celle usate vengono scelte dal compilatore da quelle presenti in libreria. Nel nostro caso la libreria è progettata per utilizzi generici e così anche le standard cell che la compongono; per progettare una memoria con delle caratteristiche diverse dovremo installare una nuova libreria. Fortunatamente per questi casi esistono dei compilatori specializzati che permettono di personalizzare diverse tipologie di memorie. In breve, una volta stabilita la memoria con le caratteristiche più compatibili possiamo, tramite il compilatore, andare a impostare il numero di bit e altri fattori. Non si avranno tutte le libertà che si avevano prima ma rappresenta davvero un buon compromesso. Nel capitolo successivo andremo ad analizzare le memorie messe a disposizione per questa tecnologia così da trovare una alternativa a quella già realizzata. Solo dopo questo procederemo con la fase successiva di Place and Route.

5 Memoria alternativa: scelta ed analisi

Prima di procedere con l'identificazione della memoria facciamo un piccolo riassunto delle librerie di standard cell messe a disposizione dalle Global Foundaries. Questo sarà propedeutico per la fase di place and route dove andremo effettivamente a piazzare le celle qui descritte nel silicio.

5.1 Librerie e kit disponibili

Le librerie Synopsis DesignWare® offrono un'ampia gamma di standard cell atta a soddisfare molti tipi di applicazioni. Per ogni standard cell ne esistono svariate varianti in base alla tecnologia con cui vengono sviluppate, di fatto viene mantenuta la funzione logica e cambiata la tecnologia realizzativa. L'insieme di tutte queste viene suddiviso in base al consumo di potenza, alle performance e all' occupazione di area per una migliore fruizione da parte dei progettisti. Di seguito potete vedere una tabella riassuntiva di tutte le librerie presenti e delle loro applicazioni:

High Performance	High Performance 9T OD, ABB-FBB Options (0.9V, 0.8V)	High-end mobility CPU (up to 2.5GHz)
	High-Density 8T OD, ABB-FBB Options (0.9V, 0.8V)	Mid-end mobility CPU (800MHz to > 2GHz)
Mid-Range	High-Density 8T ABB-FBB Options (0.8V, 0.65V)	GPU/Consumer (up to 1GHz)
	Low Power 7.5T ABB-FBB Options (0.8V, 0.65V)	
	Ultra-Low Power 6.75T ABB-FBB Options (0.8V, 0.65V)	
Low Power	Low Power 7.5T Options (0.5V, 0.65V)	IoT Application (up to 500MHz)
	Ultra-Low Power 6.75T ABB-FBB Options (0.5V, 0.65V) HVT Options (0.8V)	
	Ultra-Low Leakage 8T	
	Ultra-Low Leakage 6.75T UHVT	
	Ultra-Low Leakage Thick Oxide	

Figura 42: tabella riassuntiva delle librerie 22FDX

Per ogni libreria vi sono diversi Kit disponibili in base al tipo di performance richieste al circuito, in particolare per la 8T che stiamo utilizzando abbiamo:

- **Base Kit**: mette a disposizione celle utili per ottimizzazioni delle performance, della potenza e dell'area. Sono principalmente celle del tipo sequenziale, combinatorio e fisiche molto valide e versatili per ogni tipo di progetto.
- **Low Power Flow Kit (LPK):** mette a disposizione celle per la gestione della potenza come power switch, scambiatori di livello, celle di isolamento che sono fondamentali per bassi consumi di potenza.

- **High Performance Kit (HPK):** propone un set di celle ottimizzate da usare in parallelo con le basiche per ottimizzare il consumo di potenza.
- **Engineering Change Order Kit (ECO):** mette a disposizione celle Eco con molteplici funzionalità e a basso costo realizzativo.
- **Level Shifter (Shift/ShiftHL):** mette a disposizione delle celle di shift:
  ShiftHL: solo celle di shift unidirezionali High to Low.
  Shift: celle di shift bidirezionali.

Nella sua totalità mette a disposizione più di milletrecento celle con centonovanta funzionalità differenti. Nei gruppi principali possiamo trovare: [7] [8]

- Combinatorie: ~120
- Sequenziali: ~32
- Abilitazione PNR: ~20
- Abilitazione a bassa potenza: ~30

Le specifiche elettriche per le librerie 8T che possiamo utilizzare sono invece: [6] [11]

CELL ELETTRICAL SPECIFICATIONS			
Vt supportato	Gamma di voltaggi	Temperatura d'operazione	Lunghezza di canale
SLVT/LVT	Overdrive: 0.90 V	-40°C a 125°C	20 nm 24 nm 28 nm
	Nominal: 0.80 V		
	Under Drive: 0.65 V		
RVT/HVT	Overdrive: 0.90 V		
	Nominal: 0.80 V		

La libreria che andremo ad utilizzare, definita anche precedentemente sarà:

GF22FDX_SC8T_104CPP_BASE_CSC20R_TT_0P80V_0P00V_0P00V_0P00V_25C.db

In questa libreria saranno disponibili solo novecento cinquantanove celle, di fatto si riferisce al solo contributo portato dal kit Base. Il threshold voltage è di tipo RVT (Regular Threshold Voltage). La seguente tabella riporta le specifiche fisiche delle singole celle. [11]

CELL PHYSICAL SPECIFICATIONS	
Specifiche	Valore
Gate Length	20 nm
Cell Heigth	640 nm
Poly Pitch	104 nm
Power and Ground Rail Width	80 nm

Power and Ground Rail Metal Layer	M1
Horizontal Pin Grid	80 nm

Lascio di seguito l’architettura di una cella 8T con annotate alcune misure.

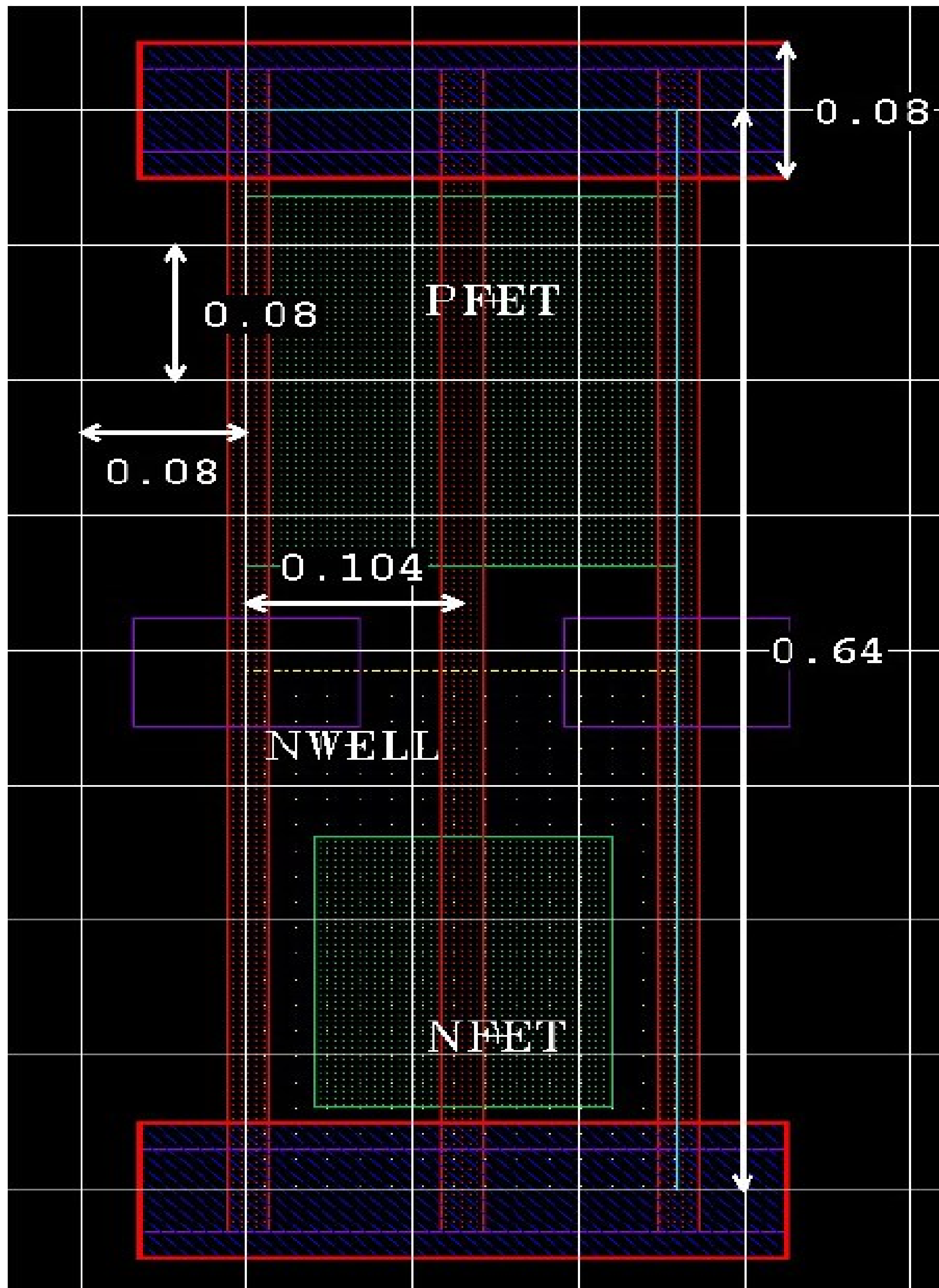


Figura 43: architettura di una cella 8T [9][11]

5.2 Memorie

Le memorie proposte dalla casa madre come affiancamento alle standard cell sono prodotte da INVECAS. Pertanto, non sono incluse nelle normali librerie viste precedentemente ma possono essere utilizzate solo con le apposite librerie e design kit proprietari. Se procedessimo con la realizzazione delle memorie in modo canonico, ovvero sviluppandole in VHDL e sintetizzandole successivamente non si riuscirebbe ad arrivare al risultato desiderato. L'insieme delle memorie proposte, come nel caso delle standard cell, viene suddiviso in librerie a seconda delle loro prestazioni, in questo caso abbiamo:

- **High performance memories** oltre 800MHz
- **Base memories** 300MHz --- 800Mhz
- **Ultra-low power memories** 40Mhz --- 300Mhz
- **Ultra-low leakage memories** 32Khz-40Mhz

Come di consueto ad alte prestazioni vi saranno anche grossi consumi di potenza mentre per prestazioni più modeste i consumi diminuiscono. In ognuna di queste librerie si possono trovare al massimo cinque tipi di memoria:

- **1P RF:** Single port RF
- **2P RF:** Dual port RF
- **SP SRAM:** Single Port SRAM
- **DP SRAM:** Dual Port SRAM
- **ROM:** Read Only Memory

Una volta scelto il tipo di memoria e provveduto ad installare il design kit corrispondente si otterrà un compilatore in grado di creare la memoria desiderata mediante l'inserimento degli appositi comandi. [14] Nel caso generale il minor numero di comandi da utilizzare per poter creare una memoria è il seguente:

*IN22FDX_MEM_genviews *	Indica il nome del compilatore attraverso cui creare la memoria
–corner 'TT 0.8 0.8 0.0 0.0 25'	Indica il nome del PVT(Process-Voltage-Temperature) corner utilizzato.
–macro memory_name	Indica il nome della memoria che si vuole utilizzare secondo la nomenclatura strutturata dalla casa madre.
*–log genviews.log *	Indica il nome del file log che verrà generato.

–out path_directory	Indica la cartella dove andranno messi tutti i file generati, se non esiste verrà generata.
–log-dir $rundir/genlogs	Indica la cartella dove tutti i file log verranno posizionati.

I due comandi di maggiore interesse per noi sono quelli riguardanti il corner e la macro; per il corner la scelta è abbastanza semplice basta scegliere tra quelli riportati nell'apposito file mentre per la macro la cosa è più complessa. [15] Come detto precedentemente il nome deve essere prodotto seguendo la nomenclatura della casa madre che è abbastanza difficile da comprendere se non viene spiegata a dovere. Nel nome deve essere racchiusa ogni informazione necessaria per identificare univocamente le caratteristiche della memoria. Per spiegarmi meglio riporto qui di seguito un esempio per capire come avviene la costruzione del nome. [16]

NOMENCLATURA DELLA MEMORIA **Example: INTTT_S1P_rltg_WwwwwwBbbbMmmCccc**		
Variabile	Descrizione	Valore
TTT	Codice della tecnologia	22FDX
S1P	Nome del prodotto	S1P: Single Port SRAM
r	Ridondanza	B = Bitline N = None
l	Latenza	P = Pipeline F = Flow-through
t	Ottimizzazione delle prestazioni	V = lower Power
g	Power Gating	G = Gated
Wwwwww	Numero totale di parole	00064 a 16384
Bbbb	Profondità dei dati di I/O	004 a 080
Mmm	Mux per colonna	M04, M08, M16
Cccc	Celle per bitline	C128 = densità C064 = performance

Esempio: IN22FDX_SDPV_BFVG_W04096B080M04C128 è una Dual-Port (2RW) SRAM con ridondanza nella bitline, Flow-through, lower Power, power-gated e configurata con 4096 parole di 80 I/O bits in Mux-4 con 128-cells sulla Bitline. [16]

La conoscenza della nomenclatura non è tuttavia esaustiva ma da una visione globale su quello che si può produrre. Per esempio, un determinato numero di parole si può raggiungere solo con particolari configurazioni della memoria. Prima di procedere con la creazione della memoria sarà necessario andare a controllare nei Datasheet quale combinazione meglio si presta alle nostre esigenze. Nelle specifiche della Dual-Port SRAM (SPDV) era riportata la seguente tabella per la configurazione C128 ad alta densità. [17]

MUX	Banks (BANK)	Word Depth (WD)		WD Granularity	I/O Width (BIT)	
		Min	Max		Min	Max
4	1	528	1024	16	16	80
	2	1040	2048	16		
	4	2064	4096	16		
8	1	1056	2048	32	8	40
	2	2080	4096	32		
	4	4128	8192	32		
16	1	2112	4096	64	4	20
	2	4160	8192	64		
	4	8256	16384	64		

Figura 44: Range di configurazioni per la memoria SPDV C128 Densa

Ora che abbiamo tutte le informazioni per riuscire a creare correttamente le memorie andiamo a valutare i pregi che possono dare in fase di progettazione. Nel nostro caso non avendo grossi problemi nelle prestazioni andremo a valutare l'occupazione d'area necessaria per produrre una memoria simile in dimensioni a quella ciclica che abbiamo sviluppato precedentemente (vedi memoria ciclica 2.3.1). Seguendo le specifiche riportate nelle tabelle sovrastanti e con i vincoli di avere almeno 1024 parole totali e ingressi/uscite a 8 bit la memoria che più si avvicina alle nostre esigenze è:

IN22FDX_SDPV_BFVG_W01056B008M08C128

La memoria così generata avrà le seguenti dimensioni fisiche:

- X(µm): 62.242
- Y(µm): 112.558
- Area(µm^2): 7005.853

Rispetto alla memoria prodotta in vhdl porta un risparmio di area molto elevato: 7000 contro i 17000 della precedente. [16] Questi risultati dimostrano la convenienza di usare le memorie prodotte da INVECAS rispetto a implementarle direttamente in VHDL. Nelle pagine successive si andranno a valutare altre specifiche tecniche così da decidere se procedere con l'implementazione di una nuova memoria.

5.3 Specifiche tecniche della memoria

Nei Datasheet forniti dalla casa madre ho trovato molteplici tabelle riguardanti le caratteristiche temporali e il consumo di potenza della memoria che abbiamo deciso di utilizzare. Ritenendo la cosa molto utile al fine di andare a confrontare le prestazioni con la memoria realizzata su Design Vision vado a riproporre qui le tabelle trovate nella documentazione; le stime sull'area erano già state riportate prima al paragrafo precedente. La prima tabella riporta varie stime sui consumi di potenza:

SPDV POWER SPECIFICATIONS			
Parameter	**Symbol**	**W01056B008M08C128**	**Notes**
Read Clock Power VDD (fJ)	PRD-VDD	1006	1, 3
Write Clock Power VDD (fJ)	PWR-VDD	911	1, 3
NOP Clock Power VDD (fJ)	PNOP-VDD	4	1, 3
Read Clock Power VCS (fJ)	PRD-VCS	833	1, 3
Write Clock Power VCS (fJ)	PWR-VCS	1030	1, 3
NOP Clock Power VCS (fJ)	PNOP-VCS	13	1, 3
Leakage Power – Standby VDD (mW)	PLEAK-NOP-VDD	3.023650e-03	3, 4
Leakage Power – Deepsleep VDD (mW)	PLEAK-DS-VDD	3.011130e-03	3, 4
Leakage Power – Powergate VDD (mW)	PLEAK-PG-VDD	3.013300e-03	3, 4
Leakage Power – UCR VDD (mW)	PLEAK-UCR-VDD	1.132263e-03	4, 5
Leakage Power – Standby VCS (mW)	PLEAK-NOP-VCS	3.547010e-03	3, 4
Leakage Power – Deepsleep VCS (mW)	PLEAK-DS-VCS	3.435440e-03	3, 4
Leakage Power – Powergate VCS (mW)	PLEAK-PG-VCS	3.498440e-03	3, 4
Leakage Power – UCR VCS (mW)	PLEAK-UCR-VCS	1.215033e-03	4, 5
Notes: 1. Input pin rise/fall time = 50 ps 2. Clock power, no outputs switching 3. VDD = VCS = 0.8 V, VBN/VBP = 0 V, Temperature = 25 °C, Process = TT (Typical-N, Typical-P) 4. CLK = Low, CEN = High (NOP) 5. VCS = 0.55 V, VDD = 0.55 V, POWERGATE=DEEPSLEEP=L, Temperature = 25 °C, Process = TT			

Da questi dati potremo ricavare molte informazioni utili circa il consumo di potenza della memoria così da compararli con quelli trovati nei report. [16] Tali consumi sono stati calcolati con le stesse condizioni operative a cui noi andremo ad utilizzare il circuito da come si può vedere alla nota numero tre. Nel nostro caso faremo conto di utilizzare la memoria solo nello stato di stand-by come preventivato precedentemente. A pagina successiva potrete trovare la tabella sulle performance temporali. Al suo interno sono riportati vari tempi minimi per il corretto funzionamento della memoria.

SPDV TIMING SPECIFICATIONS			
Parameter	**Symbol**	**W01056B008M08C128**	**Note**
Cycle Time, Flow-Through (ps)	tcyc	934	1, 2, 8
Clock High Time (ps)	tckh	150	1, 2, 4, 6
Clock Low Time (ps)	tckl	150	1, 2, 4, 6
Q Access Time Flow-through (ps)	tacc	950	1-4, 8
Q Retain Time Flow-through (ps)	tret	851	1-4, 8
Q Access Time Pipeline (ps)	taccp	197	1-4, 8
Q Retain Time Pipeline (ps)	tretp	176	1-4, 8
Address Setup Time (ps)	tas	134	1, 2, 4, 6
Address Hold Time (ps)	tah	118	1, 2, 4, 6
RDWEN Setup Time (ps)	trs	145	1, 2, 4, 6
RDWEN Hold Time (ps)	trh	103	1, 2, 4, 6
CEN Setup Time (ps)	tcs	109	1, 2, 4, 6
CEN Hold Time (ps)	tch	54	1, 2, 4, 6
Data In Setup Time (ps)	tds	123	1, 2, 4, 6
Data In Hold Time (ps)	tdh	86	1, 2, 4, 6
Bit Write Setup Time (ps)	tbs	119	1, 2, 4, 6
Bit Write Hold Time (ps)	tbh	77	1, 2, 4, 6
Wake-up Time from Power-Gate (ps)	twkup-pg	7363	1, 2, 4
Wake-up Time from Deep-Sleep (ps)	twkup-ds	53838	1, 2, 4
CEN Rise to Power-Gate Active (ps)	tpgs	1054	1, 2, 4
CEN Rise to Deep-Sleep Active (ps)	tdss	1054	1, 2, 4
Power-Gate Min Active Time (ps)	tpgh	1004	1, 2, 4
Deep-Sleep Min Active Time (ps)	tdsh	2009	1, 2, 4
DC Pin Setup Time (ps)	tdcs	530 – 90949	1, 2, 4, 5
DC Pin Hold Time (ps)	tdch	-47 -85170	1, 2, 4, 5
OBSV_CTL Propagation Delay (ps)	td-ObsvC	342	1-4

Notes:

1. Input pin rise/fall time = 50 ps
2. VDD = VCS = 0.72 V, VBN/VBP = 0 V, Temp= 125 °C, Process = SSG
3. Maximum specification
4. Output load = 10 fF
5. Applies to T_BIST, T_LOGIC, T_WBT, T_STAB, RBx, MA_x
6. Setup/Hold Time apply to both A and B Ports (Address/CEN/RDWEN/DATA/BW)
7. These setup/hold tests only apply during Scan Chain load (T_LOGIC=H) Vs CLK_A
8. MA_SAWL=2'b01, all other MA_x signals at default

Nella tabella sulle caratteristiche temporali i valori sono stati calcolati con delle condizioni diverse dalle nostre che tendono a rallentare i tempi e a peggiore le prestazioni. [16] Sebbene la tabella non si riferisca al nostro caso useremo lo stesso i dati in essa contenuti per dare un'idea delle prestazioni. Sul piano operativo non ci sono problemi visto che la memoria lavorerà ad una frequenza molto bassa di 1.7 MHz. Considerate le buone prestazioni della memoria decido di procedere con l'analisi del modello funzionale.

5.4 Dual-Port SRAM (SPDV): Analisi del modello

Una volta deciso il tipo di memoria che vogliamo utilizzare bisogna andare ad analizzare più nel dettaglio le sue caratteristiche tecniche per adattarle al nostro modello. Infatti, le memorie proposte dalle librerie sono progettate per vari tipi di utilizzi e spesso presentano maggiori funzionalità di quelle richieste dal circuito. Per esempio, alcune memorie se usate per apparecchi alimentati a batteria presentano la possibilità di addormentarsi quando non è richiesto il loro utilizzo. Lo scopo di questa analisi è di riuscire a mantenere la stessa interfacce della memoria progettata nei capitoli precedenti rimpiazzando le componenti interne create da noi (la memoria ciclica e il registro) con la memoria di libreria. L'analisi del modello logico fornito nella documentazione è molto importante perché ci permette di capire in maniera rapida le caratteristiche della memoria; fornendoci indicazioni su come andare a strutturare il circuito logico che avrà il compito di gestirla.

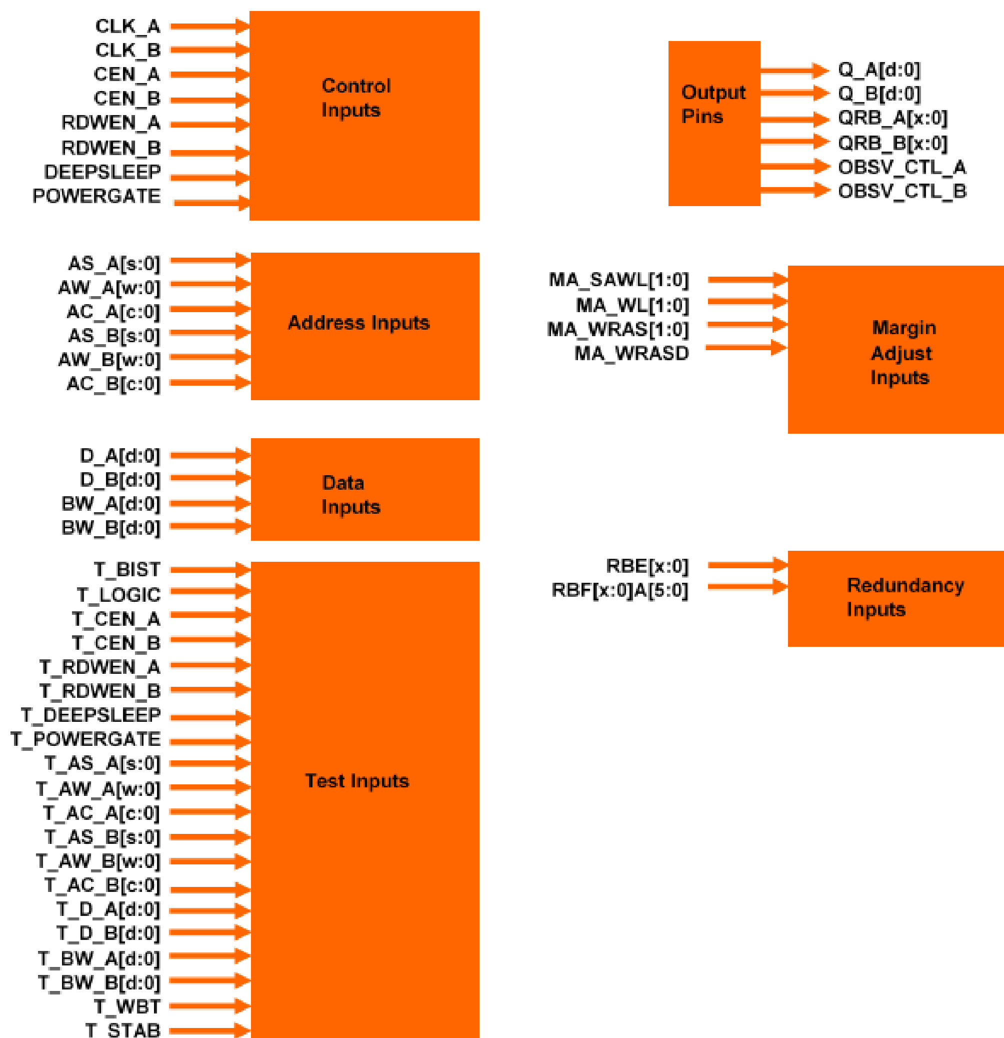


Figura 45: Modello logico SPDV (Dual Port SRAM) [16]

Come si può notare dal modello logico il numero di porte della memoria è di molto superiore a quello che potremo aspettarci da una normale SRAM a due porte. Per semplicità saranno riportati e commentati solo gli ingressi strettamente inerenti ai nostri

interessi tralasciando quelli meno rilevanti come quelli usati per i segnali di test. Gli ingressi che inizieremo ad analizzare sono quelli di controllo.

Control Inputs	
Segnale	Descrizione
CLK_A(/B)	Ingresso per il clock della porta A/B. Tutti gli ingressi della stessa porta si baseranno su questo clock.
CEN_A(/B)	Abilitazione della porta A/B **H**: gli output mantengono lo stato precedente **L**: abilita la lettura o la scrittura della porta A/B.
REWEN_A(/B)	Abilitazione di lettura o scrittura della porta A/B previa abilitazione della stessa (CEN_A(/B)). **H**: seleziona la lettura sulla porta A/B **L**: seleziona la scrittura sulla porta A/B
DEEPSLEEP	Abilitazione asincrona della modalità DEEPSLEEP. Durante questa modalità viene disconnessa l'alimentazione alla memoria perdendo tutti i dati in essa contenuti. **H**: attiva la deepsleep mode. **L**: normale stato di funzionamento
POWERGATE	Abilitazione asincrona della modalità POWERGATE. Durante questa modalità la memoria viene alimentata in modalità Power-Gated mantenendo i dati. **H**: attiva la powergate mode. **L**: normale stato di funzionamento Attenzione: attivare simultaneamente la deepsleep e la powergate mode comporta sempre l'attivazione della DEEPSLEEP ma i modelli usati per la simulazione non supportano questa configurazione.

Nel nostro caso la memoria verrà mantenuta funzionante durante tutto il suo funzionamento senza ricorrere alle due modalità sopraindicate ma è utile citare queste funzionalità in caso di futuri utilizzi. [16] Il secondo tipo di ingressi che andremo ad analizzare sono quelli relativi all'indirizzamento.

Address Inputs	
Segnale	Descrizione
AS_A(/B) [s:0]	Ingresso per l'indirizzo del banco e del sotto-array. $0 \leq s \leq 2$ Dipende dal valore riportato sulla seconda colonna della tabella di configurazione.
AW_A(/B) [w:0]	Ingresso per l'indirizzo della parola. $3 \leq w \leq 6$
AC_A(/B) [c:0]	Ingresso per l'indirizzo della colonna. $1 \leq c \leq 3$ La lunghezza dipende direttamente dal numero di multiplexer per colonna che ho scelto in fase di creazione.

La lunghezza di ogni campo viene scelta automaticamente in base alle specifiche fornite in sede di creazione della memoria. Grazie a questa descrizione si può anche calcolare analiticamente il numero di bit totali usati per i vari campi. Facendo due rapidi calcoli il numero minimo di bit necessari per l'indirizzamento sarà di:

$$Nb_{min} = \log_2(1056) = 10.044 = 11$$

Due campi vengono fissati in fase di progettazione e corrispondono a:

$$s = 0 \quad e \quad c = 2$$

Da qui si può ricavare il valore dell'ultimo campo come:

$$Nb_{min} - num_{bit}[s{:}\,0] \; - num_{bit}[c{:}\,0] = 11 - 1 - 3 = 7$$

$$7 = num_{bit}[6{:}\,0] \; \rightarrow w = 6$$

L'utilizzo di un indirizzo suddiviso in tre sottocampi può risultare troppo macchinoso per il nostro tipo di applicazioni. Per ovviare al problema possiamo ricorrere all'utilizzo di un wrapper; questi non è altro che un involucro che va semplificare il modello logico permettendoci di utilizzare solo parte degli ingressi presenti. Nel caso specifico si vanno a concatenare gli ingressi in modo da crearne uno solo da poter essere riempito in modo sequenziale secondo gli standard a cui siamo abituati. La concatenazione usata per la Dual Port SRAM è la seguente:

{AW[w:2], AS[s:0], AW[1:0], AC[c:0]}

L'indirizzo della parola viene usato come parte maggiormente significativa in quanto può non essere una potenza di due. Per esempio, con la configurazione creata da noi con undici bit avremo a disposizione ben duemila quarantotto indirizzi mentre ne andremo ad utilizzare solo mille e cinquanta. Una errata concatenazione degli indirizzi può produrre malfunzionamenti nel circuito con la perdita di informazioni. Gli ultimi due bit della parola vengono inseriti dopo AS in modo da aggiungere quattro linee di parole per ogni sotto-array. [15] La prossima tipologia di ingressi che andremo ad analizzare sono i dati che dovranno essere memorizzati.

Data Inputs	
Segnale	Descrizione
D_A(/B) [d:0]	Dato in ingresso della porta A/B durante la fase di scrittura dove il bit D_A(/B)[0] è il meno significativo. Il campo d corrisponde alla larghezza delle parole di I/O.
BW_A(/B) [d:0]	Ingresso per l'abilitazione della scrittura dei bit della porta A/B. In base al valore di questi ingressi si decide quali bit della parola D_A(/B) andranno scritti in memoria. **H**: il bit corrispondente verrà scritto in memoria **L**: il bit corrispondente non verrà scritto in memoria Per scrivere l'intera parola basta mantenere tutti i bit di questo ingresso nello stato alto.

La capacità di scrivere in memoria solo un determinato numero di bit alla volta può risultare molto utile in diverse operazioni quando si deve andare a modificare un numero inferiore di bit rispetto alla parola minima d'ingresso. Gli ultimi dati che andremo ad esporre dettagliatamente saranno quelli di uscita.

Output Pins	
Segnale	Descrizione
Q_A(/B) [d:0]	Uscita dei dati della porta A/B. Durante la lettura l'uscita viene aggiornata all'indirizzo prescelto dopo un fronte di salita del clock A/B. I tempi variano a seconda se si decide di usare la Pipeline (più veloce) o la Flow-through.
QRB_A(/B) [x:0]	Uscita per i dati ridondanti in Bit-Line della porta A/B. Usato insieme al comando T-BIST per leggere i dati ridondanti in Bit-Line.
OBSV_CTL_A(/B)	Uscita di controllo per monitorare il corretto funzionamento di alcuni segnali d'ingresso. La sua evoluzione temporale dipende da molteplici fattori.

Secondo la tabella riportata nella documentazione il tempo massimo per avere l'uscita del dato da Q_A(/B) deve essere di:

- **Tacc** = 950 ps in Flow-Through
- **Taccp** = 197 ps in Pipeline

In questo caso abbiamo scelto la modalità più lenta avendo i tempi piuttosto larghi ma è molto utile avere la possibilità di passare ad una configurazione più prestante laddove vi sia necessità. A scopo informativo riporto di seguito un breve commento sulle restanti classi di segnali viste sul modello:

- **Test Inputs:** insieme di segnali per il testare il funzionamento del chip direttamente senza dover usare i principali. Per esempio, se volessi testare la deep-sleep mode potrei farlo direttamente da un pin di test senza usare quello principale che sarà già connesso alla logica principale. I segnali T-BIST e T-LOGIC se usati insieme permettono di fare diversi test sul funzionamento della memoria.
- **Redundancy Inputs:** segnali per l'abilitazione e l'utilizzo della ridondanza.
- **Margin adjust Inputs**: ingressi per aggiustare la sensibilità della memoria in base alle proprie esigenze.

La maggior parte di essi vengono oscurati in fase di simulazione mediante l'utilizzo dei wrapper per cui dovremo preoccuparcene solo in un secondo momento in fase di Place and Route del circuito. Conclusa questa fase di analisi possiamo procedere con la progettazione della logica di controllo ben consapevoli delle potenzialità della memoria. [16]

5.5 Progettazione della logica di controllo

Nella seguente parte si andrà a predisporre la memoria per l'inserimento seriale dei dati così da andare a sostituire la memoria ciclica progettata precedentemente. Il registro da ventisei bit verrà lasciato inalterato visto l'impatto minimo sul consumo totale di area. Ipotizzando che il consumo d'area dipenda principalmente dal numero dei registri singoli totali di ogni logica è possibile ricavare l'occupazione relativa di area come:

$$\%A_{reg26} = \frac{N_{reg26}}{N_{cyclicMem}} = \frac{26}{1024 * 8} * 100 = 0.317\ \%$$

Il risultato dimostra come le prestazioni dipendano per la stragrande maggioranza dalla memoria ciclica. Fatta questa considerazione passiamo a vedere il modello logico che avrà la memoria finita per poter essere usata in sostituzione della memoria ciclica precedentemente impiegata.

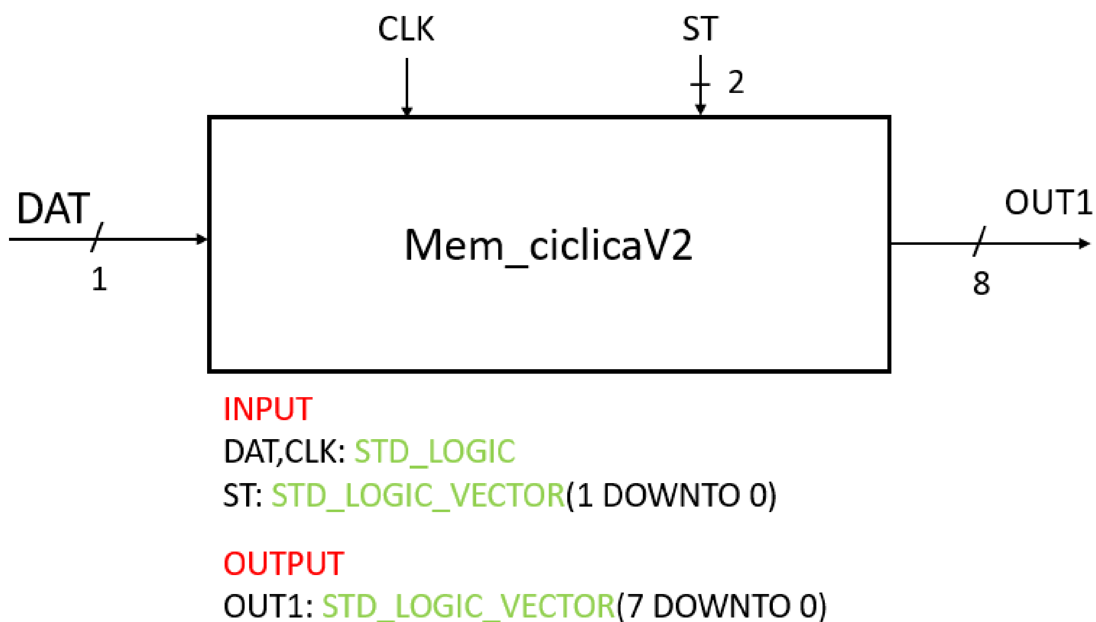


Figura 46: modello logico memoria ciclica implementata con SPDV

La memoria mediante il segnale di controllo ST potrà assumere i seguenti stati:

- **00/11 = Standby**: non fa nulla fino a nuovo ordine.

- **01 = Programming**: memorizza nella memoria SPDV i dati provenienti dal segnale d'ingresso seriale DAT, con la frequenza fornita dal segnale di clock che in questo caso sarà quello esterno.

- **10 = Reading**: legge la memoria SPDV fornendo i dati in OUT1 con la frequenza del clock esterno CLK. Il clock sarà quello fornito internamente dal trasmettitore.

Lo stato corrente sarà ricavato a monte in base allo stato che andremo ad imporre alla memoria di testing. La memoria progettata in questa maniera rimane sempre accesa

senza la possibilità di andare negli stati di deepsleep o powergate. Fatte queste considerazioni bisogna andare a vedere come predisporre la logica interna per gestire autonomamente il controllo della memoria. Nella fase di inserimento dei dati avendo un solo bit come ingresso dovremo utilizzare una maschera andando a memorizzare solo uno degli otto bit disponibili. Il segnale BW_A, analizzato precedentemente, ha appunto questa capacità abilitando o disabilitando la memorizzazione di ogni singolo bit. Grazie all'utilizzo della maschera il segnale dei dati DAT può essere connesso direttamente ad ognuno degli otto bit d'ingresso della memoria, tanto verrà memorizzato solo in uno di essi. Nel fare questo dobbiamo inoltre provvedere ad aggiornare coerentemente il campo degli indirizzi così da non avere sovrascritture e garantire un riempimento omogeneo della memoria. Il modo più semplice di implementare queste funzionalità è quello di creare un counter a quattordici bit così da rappresentare unicamente ogni singolo bit della memoria. Con i tre bit meno significativi si andrà a ricavare la parola da fornire in ingresso a BW_A mentre gli altri indicheranno l'indirizzo della memoria. Il modello del counter che si andrà ad utilizzare sarà il seguente:

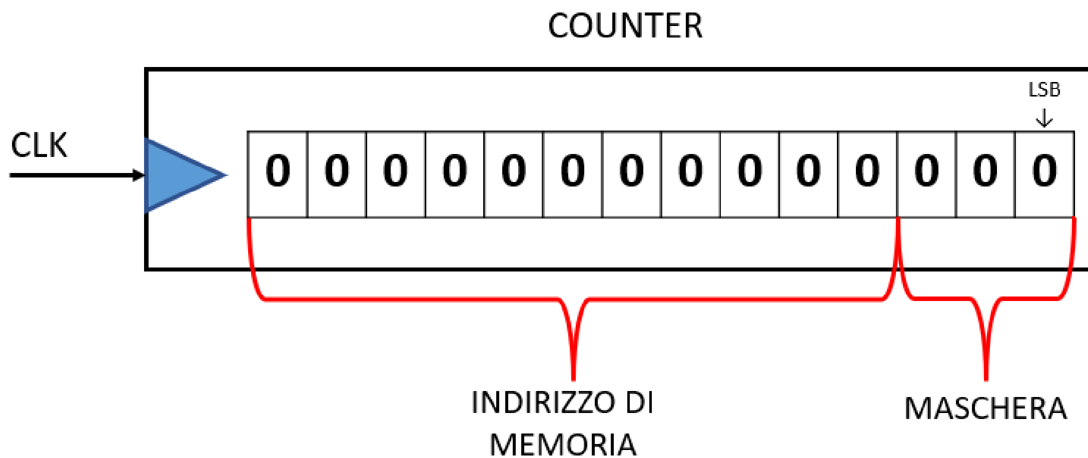


Figura 47: modello del counter utilizzato per la gestione della memoria.

Guardando al segnale BW_A dovremo predisporre una logica per alzare i bit d'ingresso coerentemente con la maschera usata dal counter. La cosa si riconduce ad un semplice AND logico che dia un uno come uscita solo quando la combinazione dei bit è quella prescelta. Le funzioni logiche che comanderanno ogni singolo canale saranno dunque:

- BW_A[7] <= not C[2] **and** not C[1] **and** not C[0]
- BW_A[6] <= not C[2] **and** not C[1] **and** C[0]
- BW_A[5] <= not C[2] **and** C[1] **and** not C[0]
- BW_A[4] <= not C[2] **and** C[1] **and** C[0]
- BW_A[3] <= C[2] **and** not C[1] **and** not C[0]
- BW_A[2] <= C[2] **and** not C[1] **and** C[0]
- BW_A[1] <= C[2] **and** C[1] **and** not C[0]
- BW_A[0] <= C[2] **and** C[1] **and** C[0]

Il valore 000 andrà ad attivare la memorizzazione sul bit significativo analogamente al flusso dei dati che è in formato Big Ending. Per evitare di eccedere il numero di

indirizzi disponibili andrò a bloccare la memorizzazione una volta riempito l'ultimo registro disponibile. Nella fase di programmazione si avranno quindi due fasi:

- **Scrittura:** dove il segnale CEN_A è alto e il numero di indirizzi è minore o uguale del massimo.

- **Refrattario:** dove il segnale CEN_A è basso e il numero dei registri e superiore al massimo.

Quando si giunge nel periodo refrattario l'unico modo di ritornare in una nuova fase di scrittura è quello di cambiare lo stato della memoria in uno degli altri e di avviare una nuova programmazione. Volendo rendere la memoria più flessibile per future applicazioni andiamo a predisporre un meccanismo di riempimento e di lettura parziale della memoria. Per implementare questa funzionalità andremo a creare un registro dedicato che salvi l'ultimo indirizzo correttamente riempito. In fase di lettura quando si andranno a leggere successivamente i vari registri il counter sarà resettato, andando a ricominciare la lettura dall'inizio, una volta raggiunto il valore indicato sul registro. La figura seguente ne mostra la dinamica:

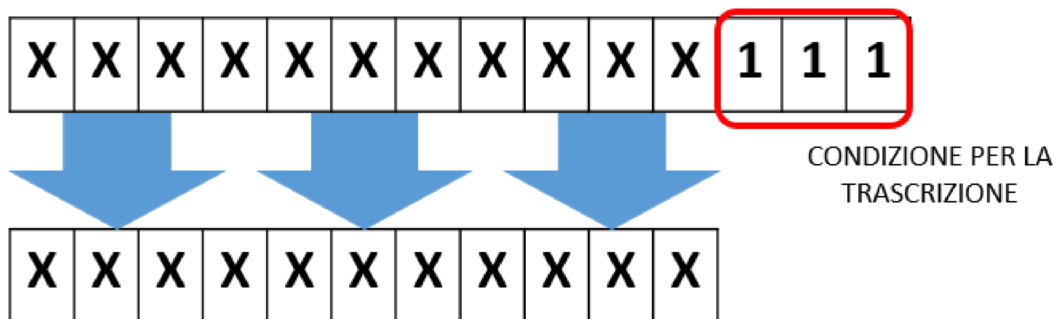


Figura 48: modello per il registro di salvataggio dell'indirizzo

Con questa ultima specifica abbiamo pienamente concluso con la progettazione della logica di controllo nella prossima fase andremo a vedere alcune caratteristiche temporali della memoria.

5.6 Conclusioni sulla memoria alternativa

Nei paragrafi precedenti sono stati riportati molti dati circa le caratteristiche tecniche della memoria, andiamo ora a riportarli qui in una maniera più ordinata:

- **Area:** l'area della sola memoria è stata stimata in circa 7005 um^2; quindi, considerando che la logica aggiuntiva del controllo e del registro necessiterà complessivamente di 51 singoli registri. Come sappiamo dal paragrafo 4.4.2 sono i singoli registri ad avere l'occupazione di area più significativa, dai conti fatti uno ne occupa circa 1,53 um^2. Possiamo dunque stimare l'area complessiva come qualcosa intorno a 7083 um^2 apportando un risparmio stimato in circa il 58% rispetto alla prima implementazione.
- **Potenza**: considerando i dati riportati in tabella possiamo già intuire i consumi che otterremo: da una prima occhiata il leakage power sarà al massimo qualche decina di microwatt, contro i 153 del caso precedente. Questo ragionamento può essere fatto considerando anche la potenza dinamica andando ad elaborare i dati riportati in Joule, anche se stime più accurate possono essere fatte solo dagli appositi strumenti. Il risultato che si ottiene sarà molto probabilmente migliore di quello ottenuto in precedenza. Inoltre, per limitare ulteriormente i consumi si possono usare gli stati di Deepsleep e Powergate.
- **Velocità di clock:** i dati riportati in tabella sono troppo complicati da essere analizzati nel dettaglio ma stando alle specifiche generali questa memoria può funzionare in un range di frequenze compreso tra 300 e 500 megahertz. Quindi anche nel caso peggiore rispetta ampiamente le nostre esigenze.

Nel complesso l'utilizzo di una memoria di libreria comporta notevoli vantaggi riguardanti il consumo d'area e la potenza. La velocità di clock massima è ridotta ma per quella a cui andremo effettivamente ad utilizzarla non comporta alcun problema. Il prossimo passo da compiere sarebbe quello di andare a realizzare veramente questa struttura in VHDL ed analizzarla; tuttavia, per alcuni problemi di compatibilità software non si è riusciti a far funzionare il compilatore per la sintesi della memoria. Non riuscendo ad ottenere il modello della memoria siamo costretti a procedere con la fase di place and route con il modello precedentemente creato. L'analisi appena conclusa potrà sempre essere riutilizzata in futuro qualora si decidesse di ottimizzare il chip o per altri progetti dove si intenda usare la stessa tipologia di memoria.

6 Place and Route con Innovus™

La fase di place and route nell'ambito dei circuiti integrati è molto importante perché permette di passare da un modello puramente digitale, la gate net-list generata precedentemente con Design Vision, ad un modello fisico che può essere implementato nel silicio. Da come si può desumere dal nome di questa fase, le principali operazioni che si andranno a compiere saranno il piazzamento delle standard cell e il loro successivo indirizzamento, volto a concretizzare le funzioni logiche della cella che costituiscono. Il numero effettivo di operazioni da effettuare per completare correttamente un Place and Route è ben più alto di due. Ognuna di queste presenta diverse insidie soprattutto nelle tecnologie più recenti dove l'estrema miniaturizzazione delle componenti genera regole sempre più stringenti. Il grafico seguente ne mostra un breve estratto.

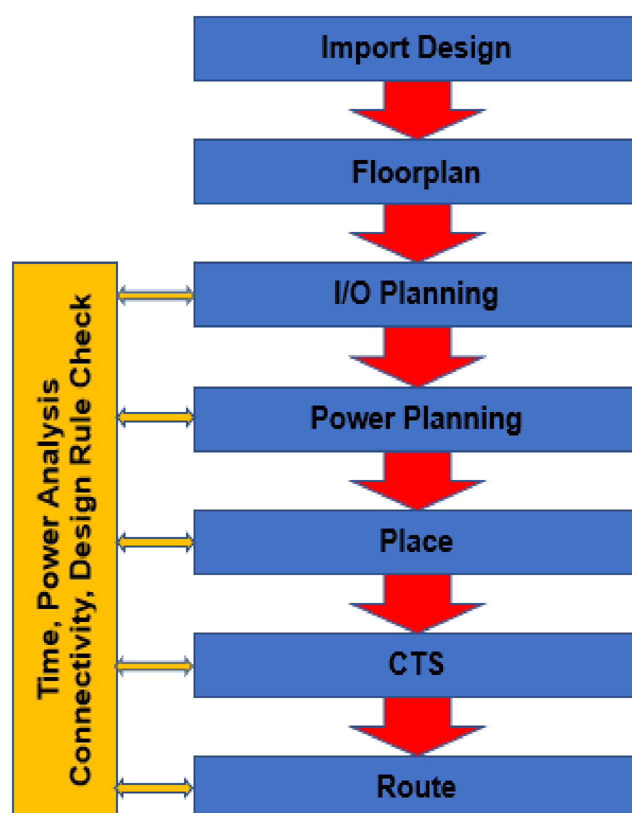


Figura 49: Flusso di progettazione nella fase di Place and Route

Andiamo di seguito ad analizzare ogni fase nel dettaglio:

- **Import Design**: fase iniziale in cui si vanno a specificare le componenti chiave che il compilatore utilizzerà per la realizzazione e l'analisi del circuito. Tra queste possiamo avere la gate net-list, le librerie, i corner per il calcolo dei ritardi etcc.
- **Floorplan**: serve per specificare al compilatore le caratteristiche spaziali del circuito come l'occupazione d'area complessiva.

- **I/O Planning:** fase in cui si vanno a disporre spazialmente gli ingressi e le uscite della cella lungo il perimetro della stessa.
- **Power Planning**: progettazione e stanziamento di tutte le strutture per la corretta alimentazione delle celle. La progettazione della griglia di alimentazione avviene sempre prima del piazzamento delle celle.
- **Place**: posizionamento spaziale delle standard cell all'interno di una predefinita area di silicio.
- **Clock Tree Syntesis**: creazione dell'albero di clock con la possibilità di produrre le prime analisi temporali del circuito. Nel nostro caso basta utilizzare il file dei vincoli precedentemente creato su Design Vision (.sdc).
- **Route**: instradamento delle entrate/uscite delle standard cell in modo da rendere operativa l'intera struttura. Dopo questa fase potranno essere fatte delle analisi per testare i risultati ottenuti e garantirne la correttezza. [18]

Il processo si fermerà alla fase di routing in quanto lo scopo principale dell'analisi è quello proporre un modello valido per l'implementazione del circuito usando standard cell a ventidue nanometri. Il fatto di utilizzare una tecnologia così recente porterà a dover compiere diverse operazioni prima di poter passare alla fase successiva. In alcuni punti dovranno essere fatte delle ottimizzazioni specifiche per le complicate caratteristiche nell'avere una tecnologia tridimensionale. Il compilatore usato da Innovus è stato ottimizzato nel corso degli anni sulle tecnologie planari mentre sulle nuove tecnologie presenta delle ambiguità che devono ancora essere sanate. Per poter settare correttamente il compilatore sarà necessario agire manualmente tramite appositi comandi. Molti dei quali sono stati trovati analizzando un design kit, appositamente studiato dalla casa madre, in cui erano presenti degli script per le ottimizzazioni. [22] Una volta completata la fase di indirizzamento saranno riportati alcuni report per garantire il funzionamento del circuito dopodiché potremo concludere il capitolo.

6.1 Setup di Innovus ed esportazione del Design

Come abbiamo visto anche nel caso di Design Vision prima di iniziare a lavorare con il tool bisogna andare a studiare l'ambiente di lavoro per evitare di incorrere in spiacevoli errori. Accompagnando quello che è il normale flusso progettuale di Innovus la prima cosa che viene menzionata subito dopo l'avvio del tool è la versione utilizzata e i suoi prodotti complementari:

- Innovus v20.11-s130_1 (64bit)
- NanoRoute 20.11-s130_1 NR200802-2257/20_11-UB (database version 18.20.512) {superthreading v2.9}
- AAE 20.11-s008 (64bit)
- CTE 20.11-s059_1
- SYNTECH 20.11-s028_1
- CPE v20.11-s013
- IQuantus/TQuantus 19.1.3-s260

Le librerie che andremo ad utilizzare essendo di tecnologie molto recenti non si prestano bene ad essere utilizzate con programmi troppo datati. In particolare, le nuove tecnologie tridimensionali a 22 nm, come quella che andremo ad utilizzare, hanno bisogno di particolari procedure d'implementazione più complesse rispetto a quelle classiche usate sulle logiche planari. Negli anni i programmi come innovus cercano di sopperire a questo problema andando a creare nuove procedure per semplificare la progettazione, diminuendo il carico di lavoro del progettista. Per questo motivo avere un programma aggiornato all'ultima versione risulta molto conveniente nell'implementazione delle tecnologie più recenti. Parlando ora di librerie quelle che abbiamo usato nell'implementazione di questo chip sono:

- **22FDSOI_10M_2Mx_5Cx_1Jx_2Qx_LB_104cpp_tech.lef**: descrive il metal stack che andremo ad utilizzare nell'implementazione del chip. Con queste specifiche si possono avere ben 43 strati di cui: 11 sono adibiti al routing, 7 sono quelli principali e 3 sono quelli riservati ai pin. [20][21]

- **22fdsoi_standard_site.lef**: serve per anticipare l'inserimento della libreria successiva, al suo interno vengono richiamate tulle le librerie disponibili della tecnologia. (104cpp 100cpp etcc).

- **GF22FDX_SC8T_104CPP_BASE_CSC20R.lef:** è la libreria principale dove vengono definite tutte le standard cell che si andranno ad utilizzare. [19]

Per settare le librerie basta semplicemente eseguire il comando:

set init_lef_file { percorso/library_name }

Prestando attenzione all'ordine che deve essere quello usato nell'elenco qui sopra. Il tutto verrà inserito in un file nel formato *.global* che sarà richiamato al momento dell'importazione del design. Oltre a questo, vi saranno altri comandi per impostare i

file generati nella fase precedente con Design Vision e altri ancora per il settaggio. Un altro file da includere all'interno del globale è il file MMMC che servirà per settare l'analisi dei ritardi, andando anche a selezionare alcune librerie specifiche messe a disposizione dal produttore. In queste librerie sono contenute tutte le informazioni per andare a calcolare i ritardi nel modo più accurato possibile secondo diversi scenari d'utilizzo:

- **TT:** usano una configurazione operativa tipica.
- **FF:** usano una configurazione operativa veloce.
- **SS:** usano una configurazione operativa lenta.

Ognuno dei quali viene poi riportato a diverse temperature e a differenti voltaggi. [7] La libreria che ho utilizzato per il calcolo dei ritardi in questo caso è stata:

GF22FDX_SC8T_104CPP_BASE_CSC20R_TT_0P80V_0P00V_0P00V_0P00V_25C.lib

Una volta identificata la libreria, la sequenza di comandi da compiere non è molto banale motivo per cui riporto tutta la serie di comandi che ho utilizzato per completare l'intera operazione:

create_library_set	Specifica di associare una libreria TCL(Tool Command Language) temporale con una libreria cdB(constant Database) /UDN. Questo permettere di trattarle come fossero un'unica entità semplificando il loro richiamo in comandi successivi.
-name libSetName	Indicherà il nome sotto cui tutte queste librerie saranno raccolte.
-timing string	Specifica il percorso della libreria temporale per il calcolo dei ritardi, nel nostro caso quella citata qui sopra.

Una volta creato un library set per le librerie desiderate bisognerà andare a impostare le modalità con cui calcolare i ritardi attraverso il comando:

create_rc_corner	Un corner RC indica al software tutte le informazioni necessarie per estrarre, annotare e usare efficacemente i valori di RC per il calcolo del ritardo. Il corner creato deve essere successivamente passato come attributo per svolgere la sua funzione.
-name RC_corner_name	Indica il nome del corner che verrà creato.
-T 25	Specifica la temperatura, in Celsius da usare per ricavare il valore delle resistenze interne.

-qx_tech_file fileName	Specifica il nome di un file Quantus technology da usare per l'estrazione dei valori di RC. Questo file viene prodotto direttamente dal venditore, nel nostro caso era il seguente: *10M_2Mx_5Cx_1Jx_2Qx_LBthick/nominal/qrcTechFile* Il loro inserimento è obbligatorio per le tecnologie più avanzate inferiori ai 30nm.
Vi sono poi altri comandi usati per il settaggio del fattore di scala prima e dopo il routing che sono stati lasciati ai loro valori di default. Ne lascio comunque le specifiche di seguito: *-preRoute_cap 1.0 -preRoute_clkres 0 -preRoute_clkcap 0* *-postRoute_res {1 1 1} -postRoute_cap {1 1 1} -postRoute_xcap {1 1 1}* *-postRoute_clkres {0 0 0} -postRoute_clkcap {0 0 0}*	

Ora che si ha creato anche il corner RC si può andare ad impostare il calcolo del ritardo vero e proprio attraverso il comando:

create_delay_corner	Un delay corner fornisce al compilatore tutte le informazioni necessarie per il calcolo del ritardo.
-name Delay_name	Indica il nome del delay corner appena creato.
-rc_corner rcCornerObj	Indica il nome del rc_corner che si vuole utilizzare.
-library_set lib_set_name	Indica il nome della library_set da associare.
-opcond_library libName	Indica la libreria interna per il calcolo dei ritardi a cui le specifiche verranno attribuite. Nel nostro caso quella citata in precedenza.

Per concludere l'operazione dovremo infine usare i seguenti comandi:

- *create_constraint_mode -name IFIR2 -sdc_files {IFIR2.sdc}* : per associare una lista di file SDC ad una specifica constraint mode.
- *create_analysis_view -name Typ_Case -constraint_mode {IFIR2} -delay_corner {corner1}*: permette di creare una struttura per l'analisi associata ad una delay corner e ad una constraint mode.

Una volta completato questi comandi il file MMMC è pronto e può essere richiamato da quello globale durante l'importazione del design. [18] Questo avviene semplicemente andando su **File→Import Design** e caricando il file *.global* appena creato e indicando un nome per i pin di potenza e di terra. Io ho scelto VDD per il pin di potenza e VSS per quello di terra come usato anche nelle librerie.

6.2 Floorplan

Dopo il caricamento dei file creati da design vision la creazione del floorplan è il primo passo per la progettazione del chip e consiste in varie fasi dove la prima è quella di progettare le dimensioni totali del chip. A questo punto si dovrebbe vedere a schermo la sagoma vuota dell'area complessivamente stimata dai modelli per la realizzazione del chip. Andando poi su floorplan view è possibile anche vedere di quali parti è composta: in questo caso sono solo due perché gli stadi da 2 sono stati raccolti a loro volta in un'unica entità. Per settare il floorplan direttamente dell'interfaccia basta andare su **Floorplan→Specify Floorplan** e inserire i seguenti valori:

- Core utilization: 0.74 (default dato dalle librerie)
- Core to left: 10 µm
- Core to right: 10 µm
- Core to top: 10 µm
- Core to bottom: 10 µm

Questa serie di comandi andrà a creare un contorno di dieci micron intorno al cuore, riportato a schermo come un tratteggio, che servirà per i vari tracciati di alimentazione e di routing. Tradotto in comandi quello che verrà fatto sarà:

floorPlan -d {W H Left Bottom Right Top}

Dove i valori di larghezza e altezza vengono presi direttamente dalle specifiche mentre i margini sono quelli che abbiamo deciso noi.

6.2 I/O Planning

Ora cha abbiamo stabilito l'area che il nostro chip andrà ad occupare bisogna definire come andrà a interconnettersi con le strutture preesistenti andando posizionare i pin di ingresso e di uscita. Per fare questo dobbiamo osservare l'area in cui il chip andrà posizionato, in particolare le porte d'ingresso del trasmettitore che saranno poi connesse alle porte d'uscita del filtro.

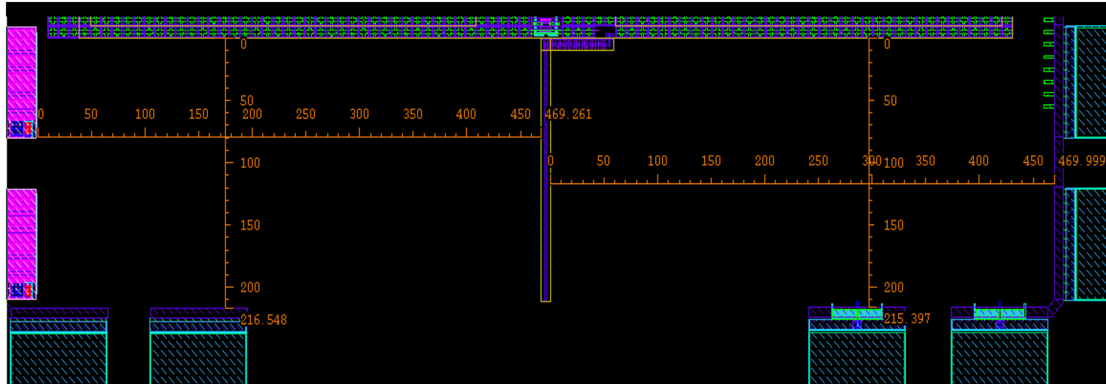


Figura 50: area disponibile per l'inserimento del filtro[1]

Nella figura sovrastante si può osservare come gli ingressi del trasmettitore siano posti in alto centralmente; di fatto sono il piccolo quadratino blu che si vede in alto al centro. Se andiamo a ingrandire quello che troviamo è il seguente:

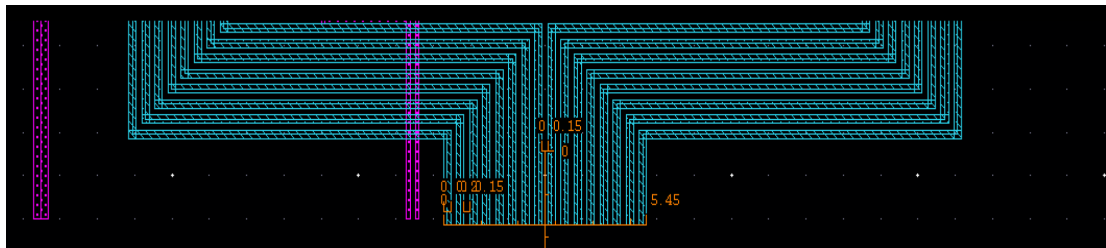


Figura 51: Bit di ingresso del trasmettitore[1]

Nella figura ingrandita si possono vedere quattro porte:

- **EN_MIX**: posta all'estrema sinistra in viola abilita il funzionamento del filtro.
- **CLOCKGEN_A0(A1)_in:** posti a centro-sinistra in viola permettono di controllare la sorgente di clock interna, sono il segnale SEL.
- **DAC_I_7→ DAC_I_0:** posti a centro-sinistra in blu sono il dato immaginario da fornire al filtro.
- **DAC_Q_0→ DAC_Q_7:** posti a centro-destra in blu sono il dato reale da fornire al filtro.

Tenendo conto di tutto questo le uscite del filtro saranno posizionate centralmente nella parte alta in modo speculare rispetto alle entrate del trasmettitore così da favorire l'inserimento del chip nella zona prestabilita. Gli ingressi al momento non saranno presi in considerazione visto che saranno posizionati meglio in fasi successive e saranno piazzati centralmente sul lato sinistro.[1]

6.4 Power Planning

Una volta che si ha definito il floorplan il secondo passo è quello di provvedere al power planning che andrà a definire i vari percorsi per l'alimentazione di ogni singolo transistor del nostro circuito. In questa fare è prevista la creazione di anelli e tracciati che permettano di propagare e rendere fruibile la potenza messa a disposizione dalla sorgente. In questo caso il voltaggio, come anche indicato dai nomi delle librerie usate per il calcolo dei ritardi al punto uno sarà:

- **VDD** = 0.8 V
- **VSS** = 0 V
- **BIAS NWELL** = 0 V
- **BIASE PWELL** = 0 V

In innovus la prima cosa da fare sarà andare a connettere le varie sorgenti specifiche messe a disposizione dalla libreria di tecnologia con quelle globali che nel nostro caso sono state chiamate VDD e VSS. Le sorgenti che dovranno essere connesse saranno:

- **VDD → VDD:** principale pin di potenza
- **VSS → VSS:** principale pin di terra
- **VDD → VNW_P:** PMOS in NWELL
- **VSS → VPW_N:** NMOS in SXCUT

Le sorgenti globali in questo caso hanno lo stesso nome dei pin usati dalle librerie ma è solo un caso dato dalla ridondanza dei nomi. L'attribuzione delle sorgenti ai pin può essere fatta con il comando:

GlobalNetName	Specifica la sorgente globale dove andrà collegato il pin, in questo caso VDD o VSS
-type pgpin	Specifica che i pin di terra o di potenza elencati con il parametro *-pin* devono essere collegati.
-pin pin_name	Serve indicare il pin da connettere alla sorgente globale
*-instanceBasename **	Specifica il nome delle istanze di tipo foglia che devono essere connesse alle sorgenti globali. La wildcard * indica di usare un modello di nomi base.

Una volta che si hanno impostato le sorgenti globali si procederà con una ulteriore fase di settaggi per ottimizzare il piazzamento delle strutture sopra elencate. In particolare, analizzando il design kit messo a disposizione dalle global foundaries, sono emersi dei comandi specifici per impostare il posizionamento dei Vias. I Via sono essenzialmente dei piccoli fori rivestiti in rame o in altro materiale conduttore in modo da mettere in comunicazione tracciati tra diversi strati di materiale. [18]

setViaGenMode	Serve per impostare il piazzamento dei via nella creazione di anelli e strisce attraverso
-align_merged_stack_via_metals ***true***	Allinea i via in delle pile
-create_double_row_cut_via ***1***	Cercare di espandere i via per avere un taglio addizionale (1 o 2). Da un taglio 1xn si può passare a un taglio 1x(n+1). **1:** crea via con un taglio addizionale quando i cavi negli strati superiori e inferiori sono sufficienti e non necessitano un cambio di forma
-cutclass_preference ***bar***	Specifica le preferenze per il taglio dei via nell'inserimento dei via speciali. **bar**: indica di usare vias a doppio taglio
-optimize_via_on_routing_track ***true***	Permette di liberare i tracciati di routing spostando o modificando leggermente i via in strati intermedi, non applicabile a strati dove esistano cavi o pin. Applicabile sono ai via generati e non a quelli predefiniti.
-Optimize_Cross_via ***true***	Genera via cercando di diminuire l'occupazione di area in ogni strato di materiale limitandone i bordi.

Altri due comandi trovati per il settaggio delle strisce verranno riportati di seguito. [22]

setAddStripeMode	Comando usato per impostare la funzione AddStrip
-stacked_via_bottom_layer ***M1***	Specifica il più basso strato dove un via può essere posizionato.
-optimize_stripe_for_routing_track ***shift***	Specifica i settaggi per lo spostamento dei tracciati per evitare spazi proibiti. **Shift**: la funzione addStripe cambia automaticamente i tracciati per preservare le risorse di routing.

Una volta aver lanciato questi comandi si può passare alla realizzazione vera e propria dell'anello e della relativa griglia di alimentazione. Gli stati prescelti per la creazione dell'anello sono stati JA e C5 seguendo l'indicazione fornita nei datasheet. Il comando può essere eseguito dalla linea di comando, come quelli precedenti, ma è più facile e

immediato andare su **Power→Power Planning→Add Ring** e seguire l'inserimento guidato**.** Le specifiche riguardo la creazione dell'anello sono riportate di seguito:

- Sorgenti: VDD VSS
- Stato superiore e inferiore: JA
- Strato destro e sinistro: C5
- Width: 1.8 μm
- Spacing: 1.3 μm
- Offset: 1.8 μm

Fatto questo si è proceduto con l'inserimento delle strisce per favorire una migliore fruizione della potenza messa a disposizione dalle sorgenti. Per farlo è bastato andare su **Power→Power Planning →Add Stripe** e inserire le seguenti specifiche:

- Sorgenti: VDD VSS
- Stato verticale: C5
- Offset: 40 μm
- Spacing: 50 μm

Il prossimo passo sarà l'instradamento speciale che di fatto andrà a sfruttare la struttura appena creata per creare i percorsi che alimenteranno le standard cell che andremo a posizionare nelle fasi successive. Il comando può essere eseguito andando su **Route→Special Route** e assicurandosi che nelle sorgenti globali siano impostate VDD e VSS. Le principali azioni che può compiere questo comando sono:

- Creare Pad rings
- Collegare pin su specifici tracciati
- Connettere i piedini agli anelli/tracciati antistanti
- Connettere i cavi non ancora connessi
- Instradamento dei pin delle standard cell

Nel nostro caso lo scopo principale è quello di rendere accessibile a ogni standard cell le sorgenti per il suo corretto funzionamento. Alla fine di questo comando potremo notare sullo schematico la creazione di molteplici tracciati orizzontali alternati tra quella di potenza e quella di massa. Per connettere tutte le varie strutture appena create avremo bisogno di un ultimo comando per l'allocazione dei via. Il comando può essere usato andando su **Power→Power Planning →Edit Power Via** e premendo su OK, senza doverlo inserire manualmente dalla finestra di comando. Con questo ultimo comando potremo osservare l'apparizione di molteplici puntini, rappresentanti appunto i via, nella struttura di alimentazione. Arrivati a questo punto è consigliabile verificare la connettività, lanciando il comando *VerifyConnectivity* e se tutto è apposto si potrà procedere con la prossima fase. [22]

6.5 Ottimizzazioni Pre-Placement

6.5.1 Ottimizzazione del compilatore

Con lo sviluppo delle tecnologie più avanzate e il successivo cambiamento dei processi di produzione creare delle tecniche per il calcolo dei ritardi diventa sempre più complesso. Le maggiori difficoltà si hanno in particolare con:

- Ritardi e interpretazione delle forme d'onda distorte.
- Molteplici cause di distorsione delle onde:
    1. Correnti di Back Miller.
    2. Rumore d'ingresso.
    3. Non linearità dei gates.
    4. Sistemi RC distribuiti.
- Applicazioni a voltaggio molto basso.

I nuovi tool per il calcolo dei ritardi nei nodi più avanzati presentano diversi livelli di precisione temporale. Il comando *setDesignMode* abilita automaticamente l'impostazione di una specifica STA (Static Time Analysis) del nodo per avere la precisione massima. Il comando che ho utilizzato per comunicare al compilatore la tecnologia le specifiche della tecnologia è stato:

setDesignMode	Comando utilizzato per settare la tecnologia di produzione globalmente senza dover agire manualmente su molteplici comandi.
-process 22	Specifica il tipo di tecnologia utilizzato per ottimizzare tutti i vari processi.

Oltre a questo, si possono anche impostare due altri parametri che sono:

-topRoutingLayer C4	Delimita il routing, dettagliato e globale, fino allo strato superiore incluso, in questo caso fino a C4.
-flowEffort extreme	Configura il flusso di progettazione per dare i migliori risultati a scapito del tempo di elaborazione.

L'impostazione dello strato C4 come massimo strato disponibile per il routing è dovuta al fatto che nello starti C3 e C2 sono presenti i pin d'ingresso del trasmettitore. Per precauzione è quindi meglio lasciarli liberi dai percorsi di routing relegandoli agli stari inferiori. [18][22]

6.5.2 Celle di Welltap

Le celle di Welltap sono dei particolari tipi di filler richiesti da alcune tecnologie per limitare la resistenza tra le connessioni di terra o di potenza del substrato, non svolgono alcun tipo di funzione logica. Hanno solo due tipi di connessione:

- N-well rispetto all'alimentazione VDD.
- P-substrate rispetto alla massa VSS.

Generalmente queste celle erano incluse automaticamente in ogni standard cell ma comportavano un notevole consumo di area, come si può vedere dalla figura sottostante.

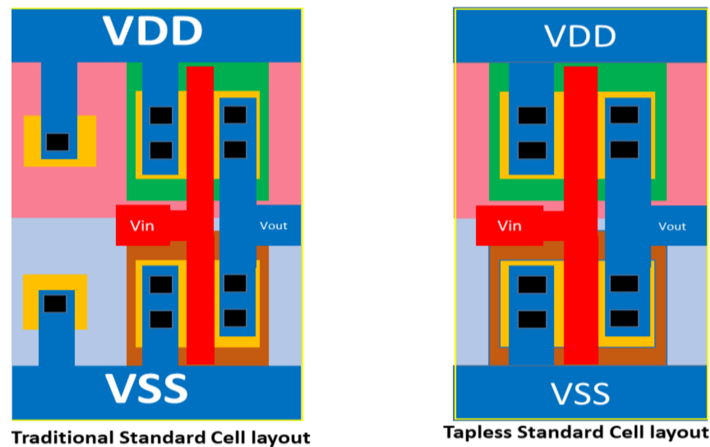


Figura 52: Standard cell tradizionale vs Tapless

Nelle nuove tecnologie queste devono essere inserite tramite appositi comandi prima del piazzamento delle standard Cell. Le celle saranno distribuite a intervalli regolari le une dalle altre secondo le specifiche della tecnologia. [23][24] Per settare il loro posizionamento ho usato il seguente comando con le specifiche consigliate nel design kit per il loro posizionamento. [18]

addWellTap	Comando utilizzato per settare i Well Tap.
-cell SC8T_TAPZBX10_CSC20R	Indica il nome della cella di Well Tap.
-prefix WELLTAP	Indica il prefisso utilizzato per le celle di Well Tap.
-cellInterval 158.08	Specifica la distanza tra il centro di un welltap il centro del welltap successivo.
-inRowOffset 5.304	Specifica l'offset in micron del primo welltap della riga.
-checkerboard	Piazza le celle di wellTap a scacchiera

6.5.3 Celle di Endcap

Le nuove tecnologie a 22nm basandosi su transistor non più planari ma tridimensionali GF22X necessitano di un numero considerevolmente maggiore di regole DRC (Design Rule Checking). Il maggior numero di regole produce numerose problematiche nella progettazione del design e nel place and route delle standard cell. Un'area particolarmente complessa da gestire è quella adibita alle periferie dei blocchi logici. Per risolvere il problema sono state progettate particolari tipi di celle chiamate ENDCAP che evitino l'incorrere di violazioni DRC. Nella precedente versione a 28nm erano disponibili due soli modelli: destro e sinistro, mentre ora è più complesso. In aggiunta a destro e sinistro abbiamo: sopra e sotto, concavo e convesso. Per meglio spiegare la nuova struttura facciamo riferimento all'immagine sottostante. [6]

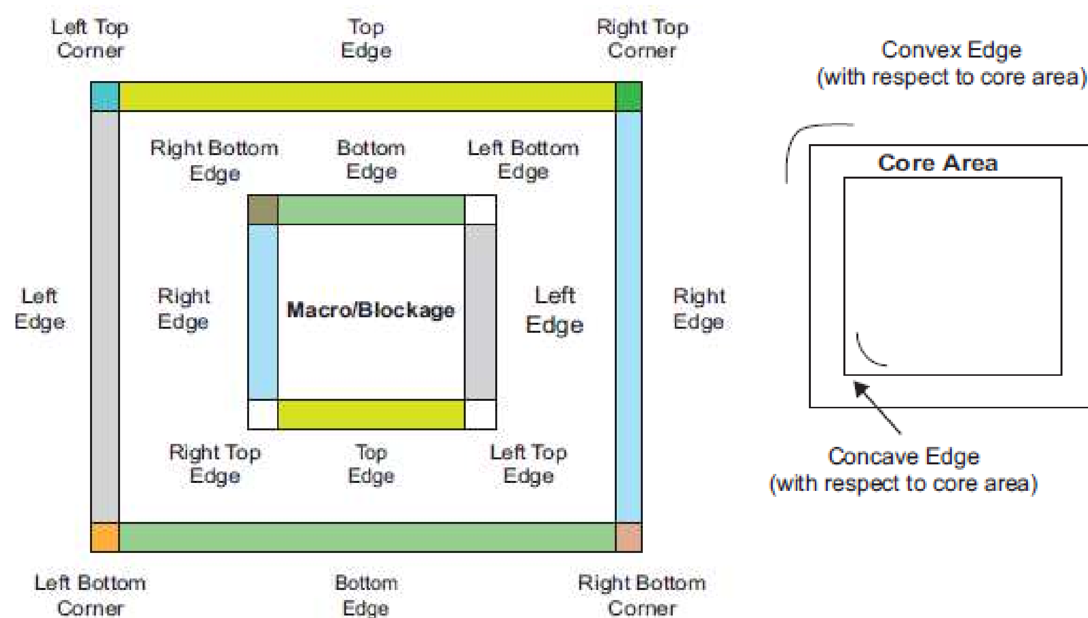


Figura 53: Struttura ENDCAP per GF22X

Come si vede in figura mentre le celle usate per la costruzione dei lati possono essere utilizzate sia internamente che esternamente, le celle d'angolo sono invece specifiche per ogni angolo. Per suddividerle, quelle interne che racchiudono il cuore vengono dette concave, facendo riferimento all'angolo concavo che producono rispetto al chip, mentre quelle più esterne vengono dette convesse facendo stavolta riferimento all'angolo convesso prodotto rispetto all'esterno. Tutte queste celle dispongono inoltre di due diversi tipi di orientazione a seconda del loro collegamento con la griglia di alimentazione.

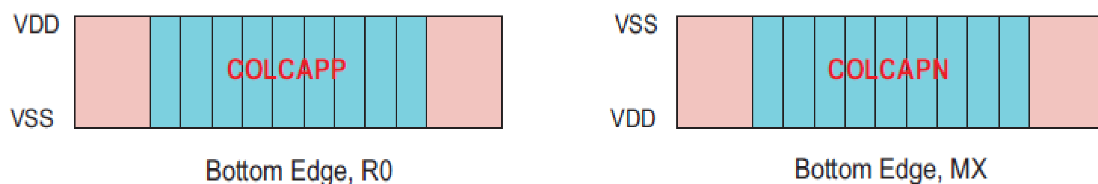


Figura 54: esempio orientazione celle ENDCAP

Questo tipo di celle utilizzate per il sopra e il sotto devono essere raccordate con quelle d'angolo al fine di una corretta implementazione. Le celle usate per i lati vengono

alternate automaticamente per l'alternanza dei tracciati VDD e VSS che si hanno nella logica e ovviamente devono essere raccordate con gli angoli: prima di un angolo orientato secondo R0 devo avere una cella di colonna orientata secondo MX. [6]

La presenza di ben dodici celle da allocare nelle posizioni corrette e con le dovute orientazioni richiede molto lavoro, motivo per cui sono state create delle funzioni di piazzamento automatico dove basti specificare il nome delle celle costituenti. Sebbene semplifichi notevolmente il lavoro può comunque presentare delle criticità: la nomenclatura non è assoluta ma dipende da produttore a produttore, infatti, Innovus definisce le posizioni rispetto alla posizione degli n-well, diversamente rispetto alle Global Foundries. Con la definizione di Innovus gli Endcap di destra e sinistra devono essere invertiti. Per maggiore sicurezza i comandi per il settaggio degli ENDCAP sono stati presi da un design Kit appositamente studiato per Innovus. I comandi utilizzati sono stati:

setEndCapMode	Per poter definire le varie celle
-bottomEdge	*SC8T_COLCAPNX1_CSC20R*
-topEdge	*SC8T_COLCAPPX1_CSC20R*
-leftEdge	*SC8T_ROWCAPANTENNARX11_CSC20R*
-rightEdge	*SC8T_ROWCAPANTENNALX11_CSC20R*
-leftBottomEdge	*SC8T_CONCAVENRX11_CSC20R*
-rightBottomEdge	*SC8T_CONCAVENLX11_CSC20R*
-leftTopEdge	*SC8T_CONCAVEPRX11_CSC20R*
-rightTopEdge	*SC8T_CONCAVEPLX11_CSC20R*
-rightTopCorner	*SC8T_CNREXTANTENNAPLX11_CSC20R*
-rightBottomCorner	*SC8T_CNREXTANTENNANLX11_CSC20R*
-leftBottomCorner	*SC8T_CNREXTANTENNANRX11_CSC20R*
-leftTopCorner	*SC8T_CNREXTANTENNAPRX11_CSC20R*
addEndCap	Per aggiungere l'ENDCAP appena definito alla logica

Come si può vedere anche dalle lettere evidenziate in rosso, le celle definite come di sinistra dalle Global Foundries devono essere rimappate come di destra su Innovus e viceversa. Questo tipo di errore può risultare molto difficile da identificare senza la possibilità di consultare degli esempi già funzionanti. [18] [22]

6.6 Placement

Dopo aver completo tutte ottimizzazioni per la tecnologia che stiamo utilizzando possiamo finalmente procedere con il piazzamento delle standard cell. Il compilatore in questa fase andrà a considerare tutte le celle piazzate nelle fasi precedenti e quelle presenti nella gate-netlist in modo da ottimizzarne la connettività. Le celle verranno interconnesse tra di loro solo nella fase di routing ma è necessario che siano posizionate nel modo corretto così da:

- Minimizzare la lunghezza totale dei cavi.
- Minimizzare la congestione delle connessioni.

Se il piazzamento viene fatto nel modo corretto si ridurrà il numero degli strati di materiale occupati e anche il consumo di potenza complessivo del circuito. Prima di compiere questa operazione bisogna specificare l'area dove l'utilizzo di alcuni strati di materiale è proibito andando su **Place→Specify→Placement Blockage** e selezionare lo strato desiderato. Fatto questo è possibile lanciare il routing andando su **Place→Place Standard Cell** o tramite il comando "*place_opt_design*". Una volta che il compilatore avrà terminato la procedura si potrà visualizzare a schermo il risultato del piazzamento per osservare la distribuzione delle celle.

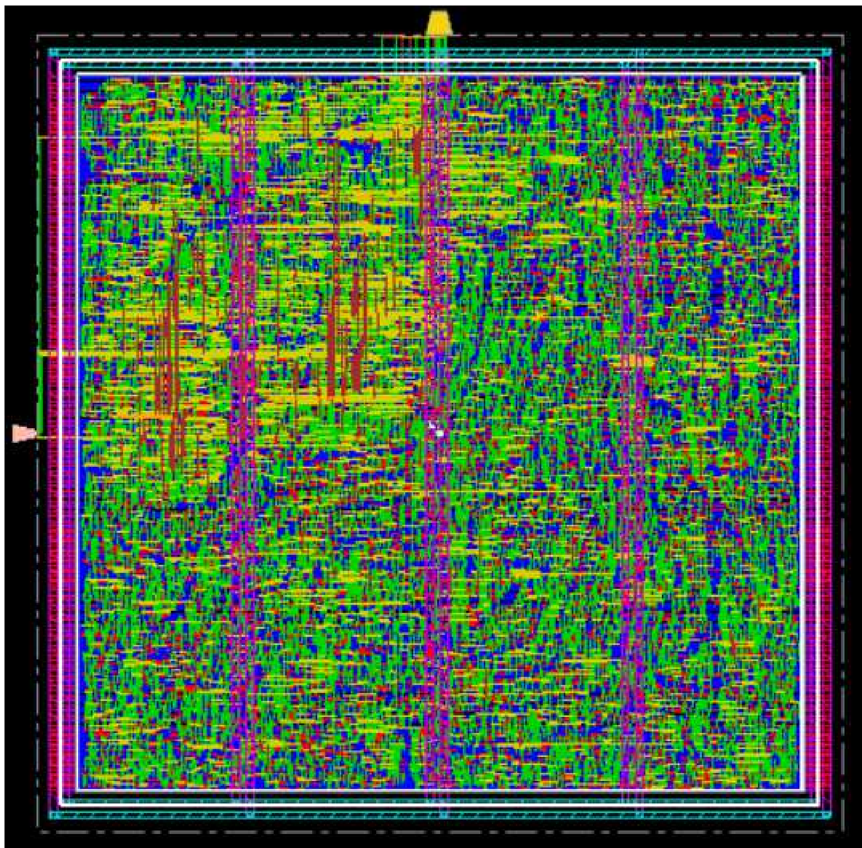


Figura 55: Innovus physical view dopo il piazzamento delle standard cell

Come si può osservare dalla figura riportata da innovus si possono distinguere due regioni: una più affollata in alto a sinistra che rappresenta i due filtri e tutto il resto che sono principalmente i registri della memoria.

6.7 Clock Tree Synthesis

Una volta avvenuto il piazzamento delle standard cell la prima cosa da fare è andare a creare l'albero di clock del circuito. A questo livello la propagazione dell'impulso di clock non avviene in maniera sincrona in tutto il circuito ma ci saranno dei ritardi dovuti alla sua struttura interna. Nel nostro caso avendo un albero di clock molto complesso il ritardo generato può essere notevole in particolare nella memoria dove il segnale di clock ha il maggiore fan-out visto l'elevato numero di registri. Tuttavia, il fenomeno diventa rilevante per alte frequenze mente a quella attuale di utilizzo il circuito non dovrebbe avere grossi problemi. Le informazioni usate per la sintesi saranno prese dal file dei vincoli generato precedentemente su Design Vision (.sdc) e dal file MMMC creato in fase di setup. Per attivarlo dovremo inserire manualmente i seguenti comandi:

- *create_ccopt_clock_tree_spec -file ccopt.spec*
- *source ccopt.spec*
- *ccopt_design*

Completate queste operazioni è possibile visualizzare sul circuito la struttura del clock e operare le prime simulazioni per vedere se qualche percorso viola le condizioni.

Figura 56: Albero di clock Innovus

Guardando la struttura appare ancora più chiaro la differenza tra le aree occupate dal filtro e quelle occupate dalla memoria. Nel report Timing che è stato lanciato non sono state rilevate violazioni temporali motivo per cui possiamo procedere con l'analisi del circuito.

6.7 Ottimizzazioni Pre-Route

6.7.1 Celle di Filler

Prima di procedere con l'ultima fase del routing (detailed routing) dobbiamo piazzare le celle di filler per garantire una rifinitura ottimale del circuito. Se si facesse un controllo delle violazioni DRC prima del piazzamento dei filler si potrebbero trovare errori quali " *Nwell minimum space not met*" dovuti alla presenza di questi vuoti non colmati. Le celle di Filler andranno a riempire tutti gli spazi vuoti presenti tra le varie standard cell garantendo continuità alle sorgenti VDD/VSS e al substrato. Tra le varie celle di filler è sempre presente una cella di grandezza minima capace di colmare anche il più piccolo vuoto presente nella logica; gli spazi vuoti tra le celle non hanno dimensioni aleatorie ma vengono fissate a priori in modo da essere chiusi completamente con l'inserimento dei filler. In Innovus il comando che ho usato per settare il posizionamento delle celle filler è stato:

addFiller	Comando per poter inserire le celle filller
-prefix	Indica il prefisso utilizzato per le celle di filler
-cell	Specifica la lista di celle da usare come filler nel design corrente. In questo caso le celle erano: *SC8T_FILLX1_CSC20R* *SC8T_FILLX2_CSC20R* *SC8T_FILLX3_CSC20R* *SC8T_FILLX4_CSC20R* *SC8T_FILLX5_CSC20R* *SC8T_FILLX8_CSC20R* *SC8T_FILLX16_CSC20R* *SC8T_FILLX32_CSC20R* *SC8T_FILLX64_CSC20R* *SC8T_FILLX128_CSC20R* I diversi numeri di ogni singola cella sottintendono alla loro occupazione di area: la X1 sarà la più piccola mente la X128 sarà la più grande.

Se dopo il piazzamento dei filler riscontriamo l'insorgere di alcune violazioni DRC è necessario ripetere il comando aggiungendo l'attributo *-fixDRC* per rimpiazzare i filler che causano queste violazioni. Per controllare che non vi siano rimaste aree vuote è possibile lanciare il comando checkFiller generando un resoconto di tutte le aree lasciate vuote senza l'inserimento di un filler. [18][22]

6.7.2 Ottimizzazioni NanoRoute

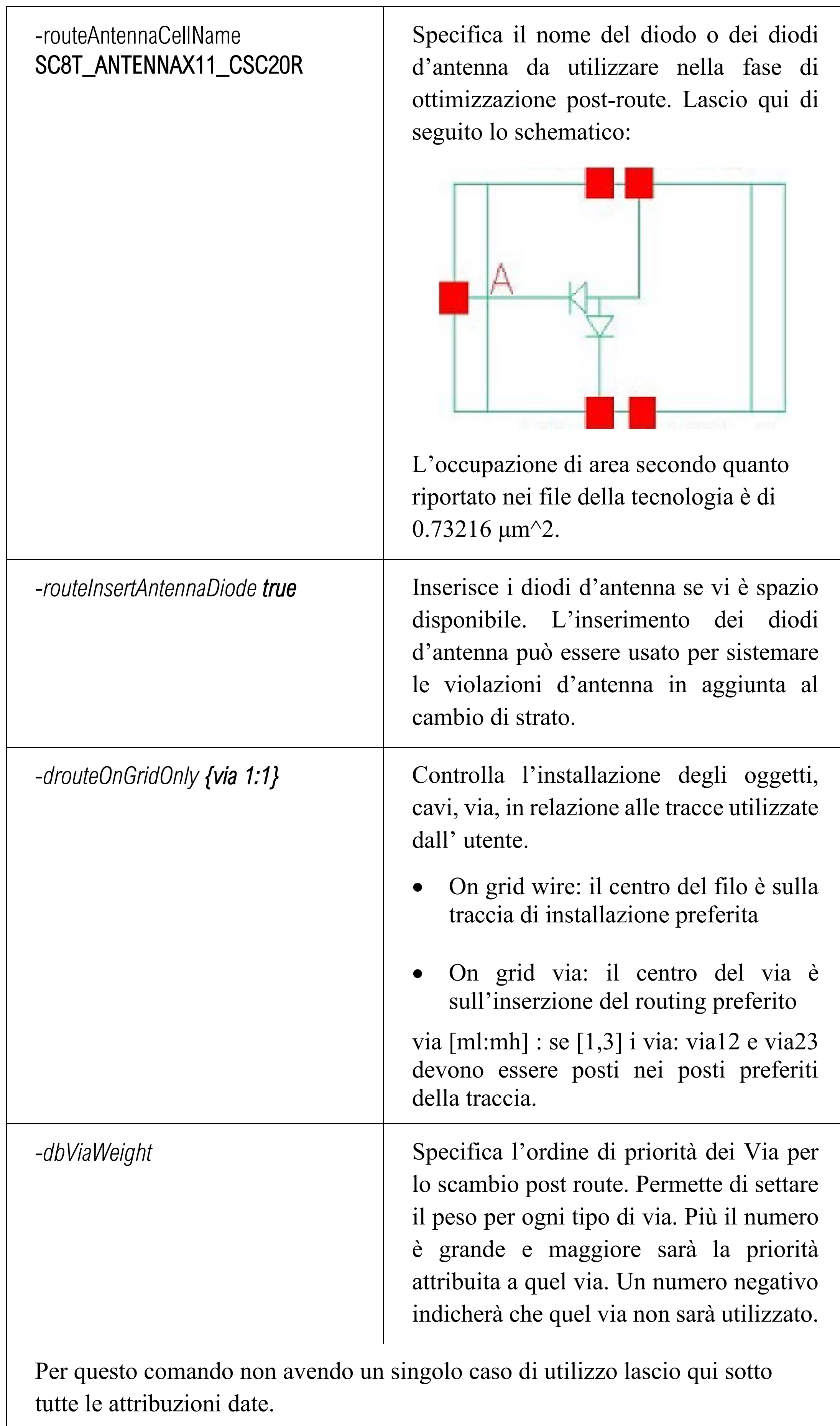
Nelle tecnologie più avanzate, come quella che stiamo usando a 22 nm, è molto importante prestare attenzione al settaggio del nanoroute. Il tool ha il compito di gestire l'instradamento dei tracciati e dei via all'interno della logica, ovvero di interconnettere tutte le standard cell posizionate nelle fasi precedenti. Nel farlo si trova a dover fronteggiare diversi problemi tra cui: area, potenza, tempo, integrità dei segnali…etc che devono essere gestiti simultaneamente in base alle performance ricercate, visto l'elevata interconnessione tra di essi. Attualmente il tool è ottimizzato per le tecnologie più datate e molti comandi utili per le nuove tecnologie risultano disabilitati di default. Ricercando nel design kit messo a disposizione da innovus sono riuscito a estrapolare una serie di comandi specifici per la libreria che stiamo utilizzando. Come primo step bisogna resettare il nanoroute per evitare l'incorrere di comportamenti indesiderati usando il comando:

setNanoRouteMode ***-reset***

Fatto ciò, si può passere alla fase di impostazione vera e propria andando a lanciare i seguenti comandi. [18] [22]

setNanoRouteMode	Comando usato per impostare il nanoroute.
-drouteAutoStop ***false***	Controlla l'insorgere di violazioni durante il routing, se settato su true può stoppare il processo di routing.
-drouteNoTaperOnOutputPin ***true***	Proibisce il routing dettagliato da manomissione delle standard cell, macro cell e blocchi di pin
-droutePostRouteSpreadWire ***auto***	Differenzia i percorsi in base alla percentuale di tracciati. Il router muove i tracciati ma non i vias. **Auto:** differenzia se il timing effort è settato su alto *setDesignMode -flowEffort high*
-routeConcurrentMinimizeViaCountEffort ***high***	Specifica lo sforzo del programma per cercare di ridurre i vias complessivi del sistema
-routeReserveSpaceForMultiCut ***true***	Riserva dello spazio per posizionare dei via multistrato nella fase di post-Route
-routeWithSiDriven ***true***	Previene o riduce le diafonie

-routeWithTimingDriven ***true***	Minimizza le violazioni temporali andando ad analizzare lo slack di ogni percorso, la drive strength di ogni standard cell, la massima capacità e i limiti di transizione
-droutePostRouteSwapVia ***true***	Permette di cambiare i via da singoli a multipli o viceversa cambiando i percorsi critici.
-drouteUseMultiCutviaEffort ***low***	Specifica lo sforzo contro l'aumento del rapporto tra via(piedini) doppi e singoli. Per doppi si intende quelli che attraversano tre metalli. Uno sforzo alto aumenta il rapporto e diminuisce il numero di via(piedini) totali del design. Più si diminuisce il numero dei singoli e migliore sarà il prodotto. **low:** routing normale con solo via singoli.
-routeWithViaInPin ***1:1***	Racchiude completamente i via(piedini) tramite geometrie all'interno dei pin delle standard cell. Se impostato come vero evita l'occorrere di violazioni del tipo MINSTEP per l'accesso ai pin. **1:1** = Specifica che i pin sul metallo 1 devono racchiudere i percorsi
-routeWithViaOnlyForStandardCellPin *1:1*	Abilita i pin delle standard cell ad essere accessibili solo tramite via e non attraverso accessi planari. **1:1** = usare sempre i via per raggiungere i pin del metallo 1
-routeUseAutoVia ***true***	Abilita l'uso di via generati internamente
-routeEnforceNdrOnSpecialNetWire ***true***	Specifica che le regole di non-default saranno rafforzate in tutti i segmenti speciali degli specifici network.
-drouteFixAntenna ***true***	Il programma ripara le violazioni d'antenna saltando tra gli strati di materiale.



-routeAntennaCellName **SC8T_ANTENNAX11_CSC20R**	Specifica il nome del diodo o dei diodi d'antenna da utilizzare nella fase di ottimizzazione post-route. Lascio qui di seguito lo schematico: A L'occupazione di area secondo quanto riportato nei file della tecnologia è di 0.73216 μm^2.
-routeInsertAntennaDiode ***true***	Inserisce i diodi d'antenna se vi è spazio disponibile. L'inserimento dei diodi d'antenna può essere usato per sistemare le violazioni d'antenna in aggiunta al cambio di strato.
-drouteOnGridOnly ***{via 1:1}***	Controlla l'installazione degli oggetti, cavi, via, in relazione alle tracce utilizzate dall' utente. • On grid wire: il centro del filo è sulla traccia di installazione preferita • On grid via: il centro del via è sull'inserzione del routing preferito via [ml:mh] : se [1,3] i via: via12 e via23 devono essere posti nei posti preferiti della traccia.
-dbViaWeight	Specifica l'ordine di priorità dei Via per lo scambio post route. Permette di settare il peso per ogni tipo di via. Più il numero è grande e maggiore sarà la priorità attribuita a quel via. Un numero negativo indicherà che quel via non sarà utilizzato.
Per questo comando non avendo un singolo caso di utilizzo lascio qui sotto tutte le attribuzioni date.	

```
setNanoRouteMode -dbViaWeight {VIA*0_30_0_35_*V1_R 8}
setNanoRouteMode -dbViaWeight {VIA*0_30_0_40_*V1_R 10}
setNanoRouteMode -dbViaWeight {VIA*0_30_0_50_*V1_R 20}
setNanoRouteMode -dbViaWeight {VIA*0_30_0_60_*V1_R 22}
setNanoRouteMode -dbViaWeight {VIA*0_30_2_53_*AY_R 20}
setNanoRouteMode -dbViaWeight {VIA*0_30_2_63_*AY_R 22}
setNanoRouteMode -dbViaWeight {VIA*0_25_0_53_*A*_R 20}
setNanoRouteMode -dbViaWeight {VIA*0_25_0_63_*A*_R 22}
setNanoRouteMode -dbViaWeight {VIA*0_30_20_20_*X_V1 26}
setNanoRouteMode -dbViaWeight {VIA*0_30_20_20_*X_AY 26}
setNanoRouteMode -dbViaWeight {VIA*0_25_18_18_*X_A* 26}
setNanoRouteMode -dbViaWeight {*BAR_V_0_30_20_20_V1 28}
setNanoRouteMode -dbViaWeight {*BAR_H_0_30_22_22_AY_R 28}
setNanoRouteMode -dbViaWeight {*BAR*_22_22_A*_R 28}
setNanoRouteMode -dbViaWeight {VIA*0_30_20_20_*2CUT_V_V1 30}
setNanoRouteMode -dbViaWeight {VIA*0_30_20_20_*2CUT_H_AY 30}
setNanoRouteMode -dbViaWeight {VIA*0_25_18_18_*2CUT_*_A*_R 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_YS 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_WT 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_WA 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_YX 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_YR 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_XD 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_YZ 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_XA 30}
setNanoRouteMode -dbViaWeight {*2CUT_*_JQ 30}
```

L’insieme di questi comandi permette di settare il nanoroute appositamente per la tecnologia che stiamo utilizzando. Fatto questo si può passare alla fase finale del routing. [18] [22]

6.8 Route

Ora che abbiamo piazzato le celle di filler e ottimizzato il compilatore possiamo procedere con il routing andando a definire i percorsi per interconnettere tra di loro le entrate e le uscite di ogni singola standard cell presente nel design. Il processo si divide in due fasi principali:

- **Global Routing:** suddivide l'area in blocchi e ne assegna i cavi.
- **Detailed Routing:** definisce il layout dei cavi in ogni blocco.

Per lanciare il routing possiamo andare su **Route→Nanoroute→Route.** Una volta che il compilatore avrà portato a termine il processo potremo andare a condurre diversi test sui risultati ottenuti:

- **Analyze Congestion:** per analizzare la congestione dei cavi.
- **Verify DRC:** per verificare se ci sono state delle violazioni delle regole
- **Report Timing (Post Route):** per garantire che le tempistiche siano rispettate

Per poter operare alcune di queste analisi bisogna prima cambiare il tipo di analisi del circuito e metterla il OCV(On Chip Variation) attraverso il comando:

setAnalysisMode –analysisType onChipVariation

Ora possiamo andare ad analizzare i risultati nel dettaglio. Partendo dalla congestione possiamo osservarla andando a disabilitare la visualizzazione dei cavi lasciando a schermo soltanto le standard cell. Riporterò di seguito un'immagine di quanto trovato.

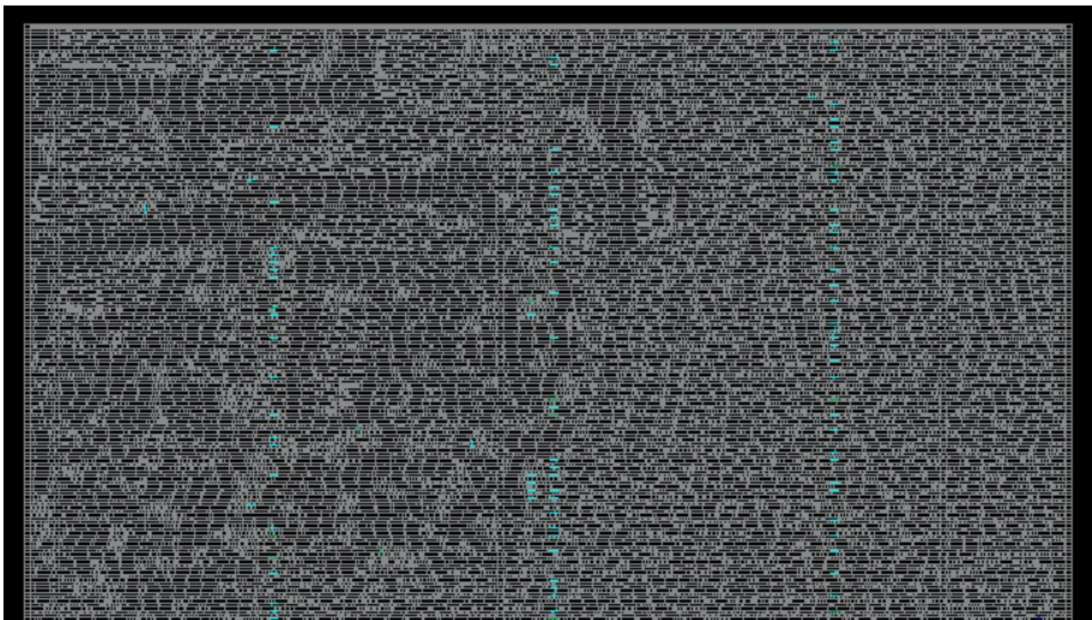


Figura 57: Analisi della congestione dopo il piazzamento

L'immagine è presa volutamente ingrandita per osservare meglio le differenze di congestione tra le due logiche: filtri e memoria. La congestione maggiore si ha nelle aree della memoria dove le celle sono più compatte mentre è meno evidente nei filtri dove appaiono più buchi. Passando al controllo delle regole DRC possiamo lanciare il comando *verifyDRC* ottenendo un report completo delle violazioni il tutto il circuito. Nel nostro caso non sono state trovate violazioni dimostrando il corretto setup del

compilatore. Finiamo dunque andando ad analizzare il Report Temporale ottenuto andando su **Timing→Report Timing(Post-Route)** e lanciando l'analisi . A fine della compilazione verrà riportata una tabella come questa:

Time Design Summary				
Setup mode	all	reg-to-reg	reg2cgate	default
WNS (ns):	5.792	6.574	N/A	5.792
TNS (ns):	0.000	0.000	N/A	0.000
Violating Paths:	0	0	N/A	0
All Paths:	9527	8749	N/A	9017

Dai risultati appare chiaro come non ci siano percorsi che violino le condizioni garantendo il corretto funzionamento del circuito. La nostra analisi può dunque giungere al termine essendo riusciti a sviluppare una procedura per l'implementazione del circuito in una tecnologia a ventidue nanometri.

7 Considerazioni finali

Da come abbiamo avuto modo di constatare nel corso di questa tesi la progettazione di un chip a standard cell è un processo complesso e dettagliato che richiede una notevole dose di conoscenza e una buona pianificazione. Nel corso del progetto si sono dovuti affrontare diversi ambiti della progettazione e interfacciarsi con differenti strumenti di progettazione avanzati, da Matlab® a Innovus™. Un aspetto fondamentale del progetto è stato il passaggio dal modello del filtro puramente teorico, sviluppato in Matlab, al suo modello in VHDL. All'inizio il modello teorico ci ha permesso di definire accuratamente la struttura del filtro e di orientare la progettazione al meglio, identificando due modelli ottimali. Successivamente, trasponendo il modello astratto in un layout più concreto siamo andati a stimare una vasta gamma di fattori, in particolare il consumo di potenza, la velocità di clock, l'area del chip. Il rimandare la scelta del modello ad uno stadio più avanzato è stato necessario per poter assimilare un numero adeguato di informazioni tali da compiere la scelta migliore. Inoltre, nelle fasi iniziali, abbiamo testato molto accuratamente i risultati ottenuti per garantire che rispettassero le nostre aspettative. Testare accuratamente i risultati è stata una pratica fondamentale per il successo del progetto assicurando che ogni step sia stato completato nel modo corretto. Il concetto è che bisogna andare ad implementare solo le cose che sappiamo di certo funzionino ed evitare di perdere tempo su cose irrealizzabili basate su un progetto sbagliato a monte. Nel percorso abbiamo dovuto fare anche dei passi indietro andando a riprogettare parte della logica della memoria in quanto poco conforme alle specifiche. Infatti, nella fase di progettazione iniziale abbiamo sviluppato un layout a complessità inferiore così da ottenere in breve tempo un modello funzionante da poter testare. L'ottimizzazione delle singole componenti può essere sempre fatta in un secondo momento se ritenuta necessaria. Nonostante gli sforzi fatti però non siamo riusciti ad ottenere la memoria desiderata. Tuttavia, l'analisi svolta sulla memoria potrà sempre ritornare utile in future applicazioni evitando di consultare direttamente la documentazione. Passando all'ultima parte della progettazione la ricerca del corretto setup di Innovus è stata molto ostica per la mancanza di una guida che ci potesse indirizzare. Per fortuna siamo riusciti ad ottenere un esempio funzionante da cui ricavare i comandi e trarre delle linee guida. Nella speranza che possano tornare utili in futuro ho riportato qui tutte le informazioni che ho trovato al riguardo. Complessivamente possiamo ritenerci soddisfatti sulle prestazioni finali ottenute in tutti i campi d'analisi e considerando anche le caratteristiche del trasmettitore con cui lavorerà.

Bibliografia

1) "A Reactive Passive Mixer for 16-QAM Cartesian IoT Transmitters in 22 nm FD-SOI CMOS", L. Tomasin, D. Vogrig, A. Neviani, A. Bevilacqua, Radio Frequency Integrated Circuits Symposium (RFIC) 2023
2) https://it.mathworks.com/help/matlab
3) https://it.wikipedia.org
4) Design Compiler® User Guide, Version P-2019.03
5) Design Vision™ User Guide, Version Q-2019.12
6) GLOBALFOUNDRIES® 22FDX® Standard cells 8T, User Guide Version 0.7
7) DesignWare® GLOBALFOUNDRIES 22nm FD SOI, LOGIC LIBRARY USAGE GUIDELINES, Application Note-0311
8) www.synopsys.com/designware
9) GLOBALFOUNDRIES® 22FDX® GF22FDX_SC8T_104CPP_BASE_CSC20RVT, Standard Cells Library, Release Notes, Known Issues, and Recommendations, Version: 5.30
10) GLOBALFOUNDRIES® 22FDX® GF22FDX_SC8T_104CPP_BASE_CNRX-SDB-C20RVT, TT 0P90V 25 °C Standard Cells Library Version 2.00
11) GLOBALFOUNDRIES® 22FDX® GF22FDX_SC8T_104CPP_BASE_CNRX-SDB-C20RVT, TT 0P80V 25 °C Standard Cells Library Version 5.00
12) GLOBALFOUNDRIES® 22FDX® GF22FDX_SC8T_104CPP_BASE_CNRX-SDB-C20RVT, TT 0P65V 25 °C Standard Cells Library Version 1.00
13) GLOBALFOUNDRIES® 22FDX® GF22FDX_SC8T_104CPP_BASE_CNRX-SDB-C20RVT, TT 0P50V 25 °C Standard Cells Library Version 1.00
14) INVECAS IP Portfolio in 22FDX™, Bhaskar Kolla , The Annual Tokyo SOI Workshop 31st May to 1st June 2017
15) GLOBALFOUNDRIES® 22FDX® Embedded Memory Applications Application Note Version 1.04, April 17-2020
16) GLOBALFOUNDRIES® 22FDX® Dual-Port SRAM (SDPV) Datasheet Version 1.20, June 01-2020
17) GLOBALFOUNDRIES® 22FDX® Dual-Port SRAM (SDPV) User Guide Version 1.02, June 01-2020
18) Innovus Text Command Reference, Product Version 21.15, Last Updated in August 2022
19) 22FDX®-EXT MSOA Enablement, Application Note, Version 2.3

20) 22FDX®-EXT V1.0_2.0a Process Design Kit, Innovus Techfiles Component Release Notes, Version 1.0_2.0a
21) GLOBALFOUNDRIES TECHNOLOGY PORTFOLIO, Updated version 2.0 from 12 July 2019
22) 22FDX INNOVUS based Digital Design, Reference Flow (RFL), Design Methodology Team, March 31, 2020
23) https://teamvlsi.com/2020/08/well-tap-cell-in-asic-design.html
24) https://teamvlsi.com/2020/05/latch-up-prevention-in-cmos-design.html