Capitolo 4

Implementazione Modulatore QPSK

4.1 Sommario

I principali risultati di questa Tesi sono racchiusi in questo capitolo dove vengono illustrati due diverse realizzazioni, una ricalca lo schema classico del modulatore QPSK mentre l'altra è una diretta estensione del progetto ROM Polifase *clock-enable*[1].

Il modello Matlab dell'intero canale di comunicazione comprendente trasmettitore, canale e ricevitore, ha consentito sia una valutazione qualitativa dei risultati tramite gli spettri che una misura quantitativa tramite la BER agevolando inoltre lo sviluppo di entrambe i modelli VHDL.

4.2 Modello Matlab

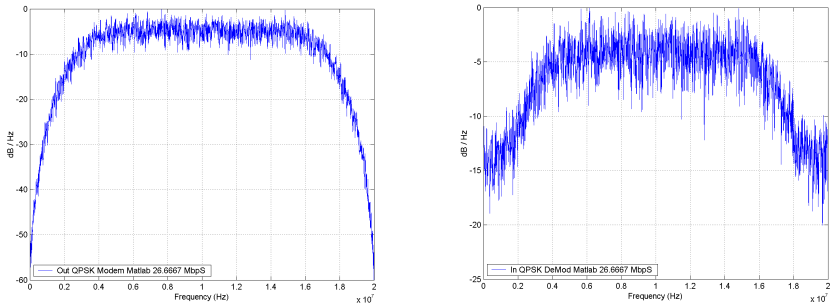
La descrizione del modello è sia in ***QPSKModemPSDeBERVHDLvsMatlab.m*** *(Listato E.4.4)* che in ***QPSKModemPSDFPGAvsPSDMatlab.m*** *(Listato E.4.6)* , la sua definizione si basa, per quel che riguarda il trasmettitore, sullo schema del modulatore classico con $\mathbf{f}_{clk} = 4 \times f_{if}$ di Figura(2.6) mentre per il ricevitore si è utilizzata la teoria della demodulazione con filtro adattato[2] ed il criterio di decisione ML descritto in Sezione(1.3.2), essa è di riferimento anche per la caratterizzazione del rumore introdotto dal canale. Segue una breve descrizione del modello:

Trasmettitore: dal rapporto tra la frequenza di clock ed il ritmo di simbolo viene dedotto il file di coefficienti da utilizzare, essi sono calcolati da ***CreaCoeffsFreqSamplScaled.m*** *(Listato E.3.1)* e memorizzati in *SRRCx3_FreqSampl_scaled.dat*, *SRRCx4_FreqSampl_scaled.dat* e

[1] Sezione(3.4.2)
[2] Sezione(1.3.1)

SRRCx6_FreqSampl_scaled.dat. La sequenza randomica dei bit da trasmettere viene caricata da *bit_tx.dat* ed è anche utilizzata per caratterizzare il modello VHDL e l'implementazione su FPGA al fine di poter effettuare confronti in termini di BER. Dalla sequenza *bit_tx* se ne ottengono due, quella contenente i bit dispari viene applicata al ramo I del modulatore mentre l'altra, contenente i bit pari, viene applicata al ramo Q per poi essere applicate entrambe ai due filtri SRRC polifase, implementati tramite ***applica_polifase.m*** *(Listato E.3.2)* . Le uscite degli interpolatori sono moltiplicate rispettivamente con i campioni del coseno e del seno rispondenti al vincolo $\mathbf{f}_{clk} = 4 \times f_{if}$, i campioni risultanti vengono poi sommati.

Canale: viene sommato del rumore gaussiano al segnale in uscita dal modulatore, nel caso di un rumore quantificato con Eb/No = 6dB l'effetto è rappresentato in Figura(4.1).



(a) Uscita Tx QPSK (b) Ingresso Rx QPSK

Figura 4.1: Effetto del canale sul segnale modulato QPSK

Ricevitore: il segnale QPSK affetto da rumore viene applicato ad un ADC[3] e successivamente moltiplicato per i campioni delle due portanti in quadratura, il successivo filtro adattato utilizza i medesimi coefficienti dell'SRRC in trasmissione ma mantiene inalterato il ritmo dei campioni, i simboli che si ottengono alla sua uscita vengono applicati al decisore implementato tramite la funzione *demodmap* di Matlab. La sequenza di bit ricevuta viene confrontata con quella trasmessa, non essendo stati utilizzati dei bit di segnalazione tale operazione si effettua traslando la sequenza ricevuta di un numero di bit *N_bit_sfasamento_Matlab* rispetto a quella trasmessa, il valore viene determinato in maniera empirica effettuando diverse traslazioni e scegliendo quella cui corrisponde il minimo numero di bit errati.

[3] Analog Digital Converter

Lo script ***CreaTabellaBER.m*** *(Listato E.4.1)* consente una valutazione globale delle prestazioni del modulatore, esso infatti confronta sia in forma tabellare che grafica la probabilità d'errore teorica con la BER ricavata sperimentalmente, a tal fine si avvale della funzione ***calcolaBER.m*** *(Listato E.4.2)* che valuta la BER in corrispondenza di un dato valore di Eb/No. Il numero di bit applicati al modulatore è funzione di Eb/No infatti se ad esempio per un dato valore di Eb/No la probabilità d'errore è dell'ordine di 10^{-4} allora il numero di bit da trasmettere è 10×10^4. La Tabella(4.1) è riferita al data rate 110Mbps e alla frequenza di clock 165MHz cui corrisponde

Eb/No (dB)	Probabilità d'errore teorica	BER Modem QPSK Matlab
0	0.07864960	0.07849516
1	0.05628195	0.05538872
2	0.03750612	0.03828828
3	0.02287840	0.02435769
4	0.01250081	0.01584918
5	0.00595386	0.00667334
6	0.00238829	0.00309671
7	0.00077267	0.00084889
8	0.00019090	0.00020659
9	0.00003362	0.00003194

Tabella 4.1: Tabella prestazioni modem Matlab in termini di BER

la curva del BER rappresentata in Figura(4.2).

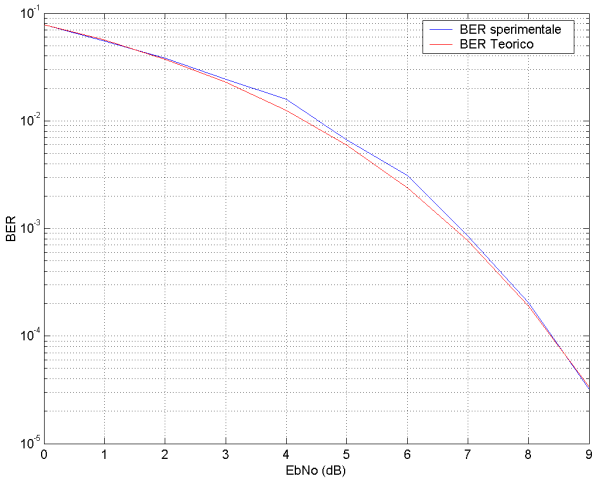


Figura 4.2: Grafico prestazioni modem Matlab in termini di BER

4.3 Modelli VHDL

Dopo aver introdotto la realizzazione VHDL dello schema di principio presentato in Figura(2.6) si passa alla descrizione del progetto ThinModulator che incorpora tutte le ottimizzazioni dedotte nel capitolo precedente e ne introduce altre consentite dall'ortogonalità dei due rami del modulatore QPSK.

4.3.1 Modulatore classico

Lo schema in Figura(4.3) corrisponde a ***Modulator_BlockRAM.vhd***

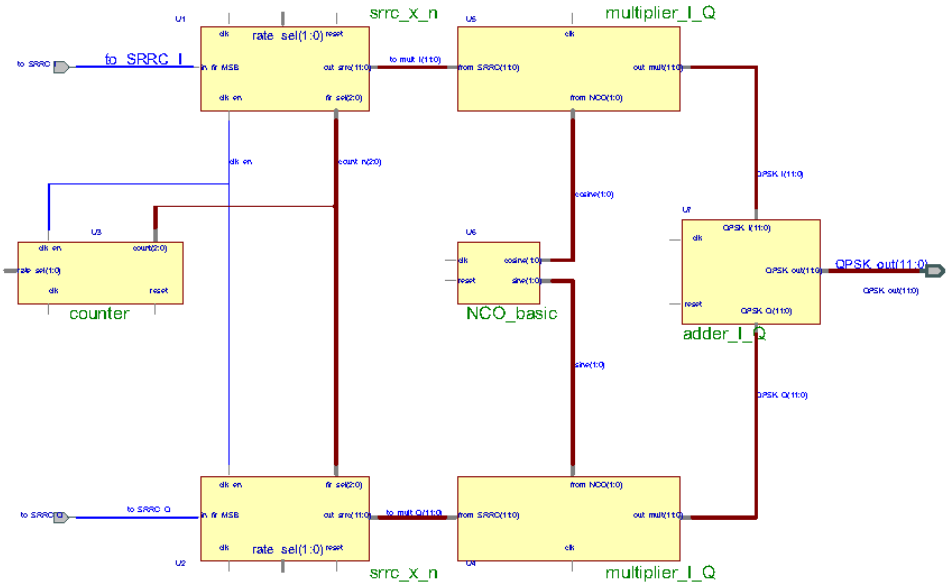


Figura 4.3: Modulatore classico

(Listato F.4.1) , si assume che il modulatore riceva in ingresso sia la sequenza dei bit pari che quella dei bit dispari, esse vengono applicate al filtro SRRC polifase in Figura(4.4) che si differenzia da quello di Figura(3.7) unicamente per il modo in cui vengono memorizzati i risultati delle somme, non più in 3 ROM distribuite nelle CLB bensì in un'unica RAM[4] abilitata soltanto in lettura ed implementata nella BlockRAM[5] tramite un Core Xilinx.

Trattandosi di un modulatore QPSK per applicazioni spaziali sono importanti sia le dimensioni che la massima frequenza di clock implementabile, nel progetto della RAM si è scelto di ottimizzare quest'ultimo aspetto inserendo le somme per ogni interpolatore SRRC in 1024 locazioni di memoria

[4] inizializzata con il file generato da ***CreaROM.m*** *(Listato E.3.4)*
[5] Appendice(C.3.4.2)

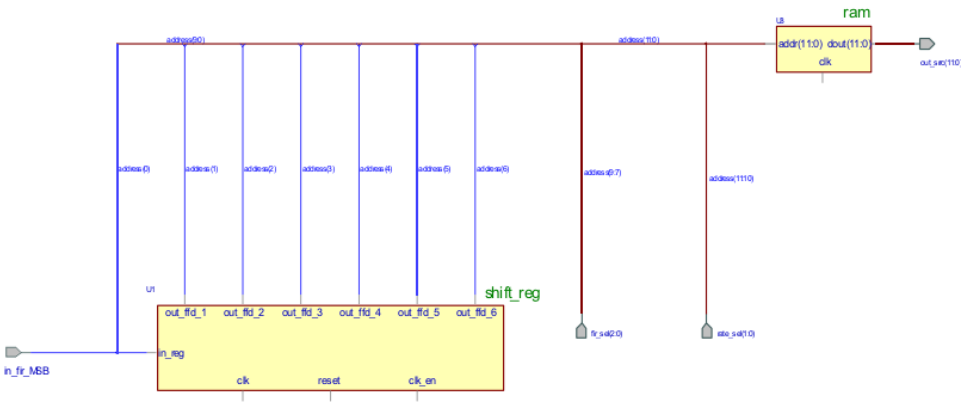


Figura 4.4: SRRCxN versione RAM

indipendentemente dal fatto che siano tutte necessarie oppure no, in tal modo si può selezionare un SRRC oppure un altro applicando il segnale *rate_sel* ad ulteriori 2 bit di indirizzo della RAM la quale quindi non richiede alcuna logica combinatoria a monte della sua rete di indirizzamento intrinseca. La descrizione VHDL del polifase è in ***srrc_x_n.vhd*** *(Listato F.4.6)* mentre il file per la simulazione della RAM è ***ram.vhd*** *(Listato F.4.7)* tuttavia si può evitare l'utilizzo del Core Xilinx istanziando 12 BlockRAM nel formato 1×4096bit come in Figura(4.5).

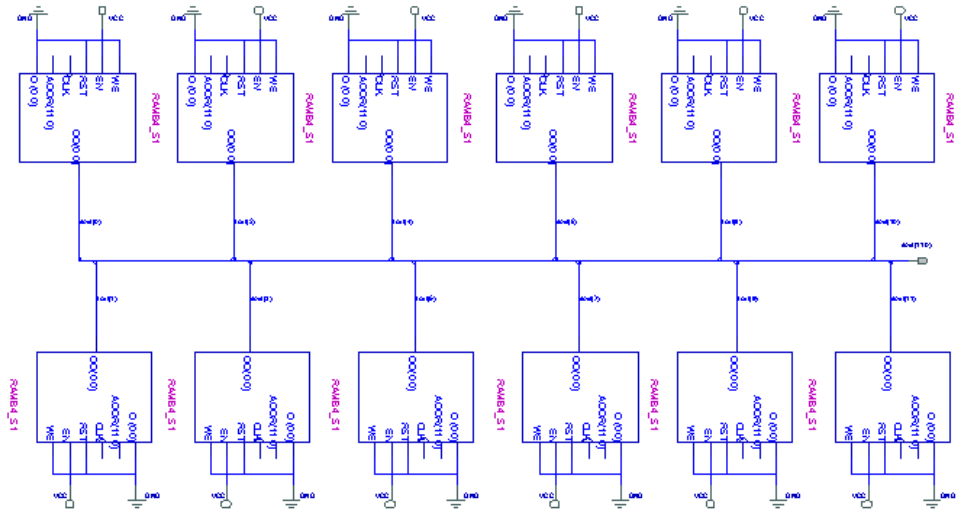


Figura 4.5: RAM 12×4096

Le due portanti in quadratura sono generate da ***NCO_basic.vhd*** *(Listato F.4.5)* a partire dal clock di sistema mediante un contatore modulo 4

alla cui conta ciclica vengono associati i valori di coseno e seno riportati in Tabella(4.2), i campioni delle portanti così generate vengono moltiplicati per

Valore contatore	Coseno	Seno
0	1	0
1	0	−1
2	−1	0
3	0	1

Tabella 4.2: Generazione funzioni trigonometriche col contatore

i campioni provenienti dagli SRRC, in particolare in ***multiplier_I_Q.vhd*** *(Listato F.4.4)* essi vengono lasciati passare, annullati oppure invertiti a seconda del valore della corrispondente portante per poi essere sommati in ***adder_I_Q.vhd*** *(Listato F.4.2)* la cui complessità può essere evitata, sostituendo il sommatore con un più semplice commutatore operante alla frequenza di clock, i due ingressi sono infatti alternativamente nulli. Tale ottimizzazione, così come le prove sperimentali e la successiva implementazione su FPGA, è stata inserita nella rielaborazione del progetto denominata ThinModulator la quale include anche l'interfacciamento tra il modulatore e la sorgente dei dati.

4.3.2 ThinModulator

Il nome ThinModulator[6] evidenzia come in questo progetto siano state inserite tutte le ottimizzazioni rese possibili dalla modulazione QPSK, esse sono evidenziate dal diagramma temporale in Figura(4.6), relativo alla rea-

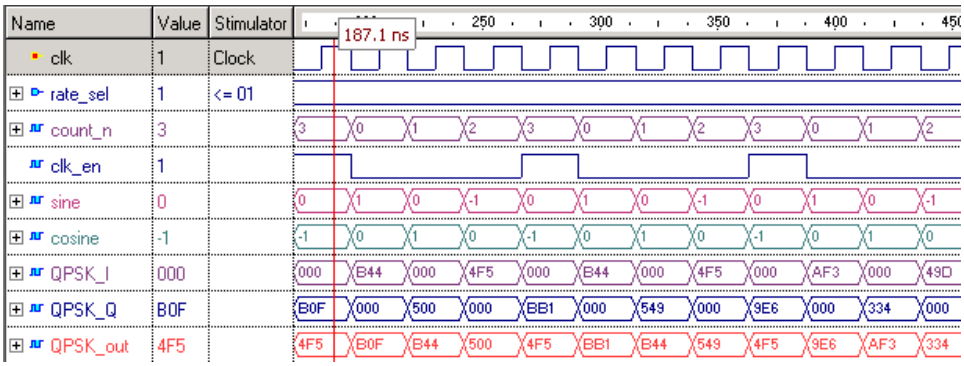


Figura 4.6: Temporizzazioni modulatore classico

[6] *modulatore snello* in inglese

lizzazione VHDL del modulatore classico di Figura(4.3), in particolare si evincono due considerazioni:

1. l'uscita *QPSK_out* del modulatore si ottiene prelevando alternativamente l'uscita *QPSK_I* del moltiplicatore sul ramo I oppure l'uscita *QPSK_Q* del moltiplicatore sul ramo Q.

2. le sequenze seno e coseno sono sfasate tra loro di 90°, immaginando di sommarle algebricamente si ottiene una sequenza che per due periodi di clock vale +1 e per i successivi due periodi vale −1.

su di esse si basa il ThinModulator mostrato in Figura(4.7),

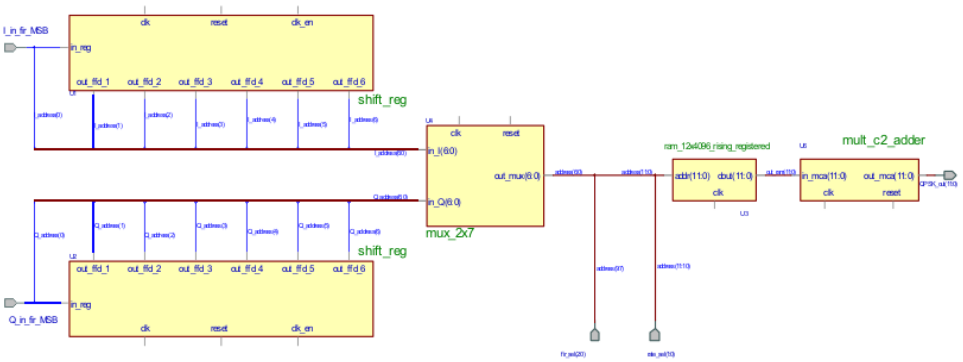


Figura 4.7: ThinModulator

i due interpolatori SRRC vengono realizzati con una unica RAM in quanto il contenuto per il ramo I è uguale a quello del ramo Q inoltre, come evidenziato in Figura(4.6), un campione ogni due dell'uscita del polifase viene annullato nel successivo moltiplicatore, pertanto è sensato intercalare i campioni, cosa che del resto nello schema in Figura(4.3) veniva effettuata dal sommatore d'uscita. Il commutatore ***mux_2x7.vhd*** *(Listato F.5.10)* provvede ad inviare alternativamente all'unica RAM un indirizzo oppure l'altro, vengono poi aggiunti 2 bit per la selezione del FIR[7] ed altri 2 bit per la selezione del data rate. Nell'architettura ThinModulator le moltiplicazioni e la somma finale vengono tutte effettuate in ***mult_C2_adder.vhd*** *(Listato F.5.9)* che opera in maniera ciclica lasciando passare inalterati 2 campioni provenienti dalla RAM ed invertendo i successivi due.

Il ThinModulator necessita per il suo funzionamento del contatore programmabile ***counter.vhd*** *(Listato F.3.2)* descritto nella Sezione(3.4.2), esso genera il segnale di abilitazione per la tecnica *clock-enable* e la conta ciclica che va a selezionare l'N_{esimo} banco di 128 locazioni nella RAM corrispondente all'N_{esimo} FIR del polifase SRRC.

[7] e quindi del banco di 128 locazioni di memoria

Il modulatore QPSK effettivamente implementato sulla FPGA è quello il cui schema a blocchi è rappresentato in Figura(4.8), esso include anche

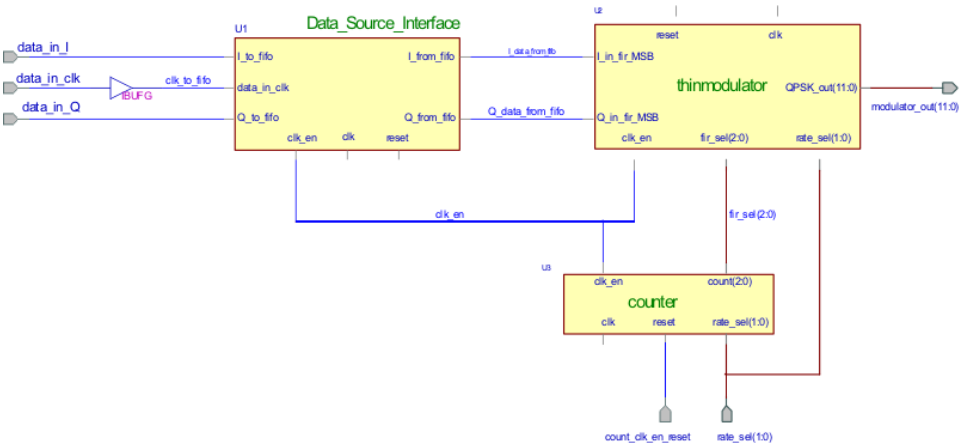


Figura 4.8: FIFO RAM ThinModulator

una interfaccia verso il mondo esterno resasi necessaria in quanto il clock associato ai dati provenienti dal *pattern-generator* ha la stessa frequenza del segnale *clock-enable* ma la relazione di fase tra i due è aleatoria e pertanto si possono verificare situazioni non previste in fase di progetto.

Le possibili soluzioni al problema sono due:

1. utilizzare il clock dei dati per produrre il clock di sistema, da esso poi si ricava il segnale *clock-enable*, la soluzione è impraticabile in quanto le specifiche implicano la moltiplicazione del clock per fattori 3, 4 e 6 mentre la FPGA Virtex consente soltanto la moltiplicazione per 2 utilizzando una ClkDLL[8][9] come in Figura(4.9). oppure per 4 mettendone due in cascata.

2. una FIFO[10] consente di mettere in fase le due temporizzazioni, essa è costituita da un registro a scorrimento nel quale i dati vengono scritti utilizzando la temporizzazione che giunge dal *pattern-generator* e letti utilizzando quella prodotta localmente dal contatore programmabile ***counter.vhd*** *(Listato F.3.2)* .

L'unica soluzione che soddisfa le specifiche del modulatore è la 2) tuttavia essa richiede che le due temporizzazioni abbiano la stessa frequenza altrimenti dopo poco tempo la FIFO o si svuota o si riempie completamente determinando così perdita di informazione, per soddisfare questo vincolo si

[8] Appendice(C.3.4.1)
[9] la frequenza minima in ingresso alla ClkDLL è 25MHz
[10] First In First Out

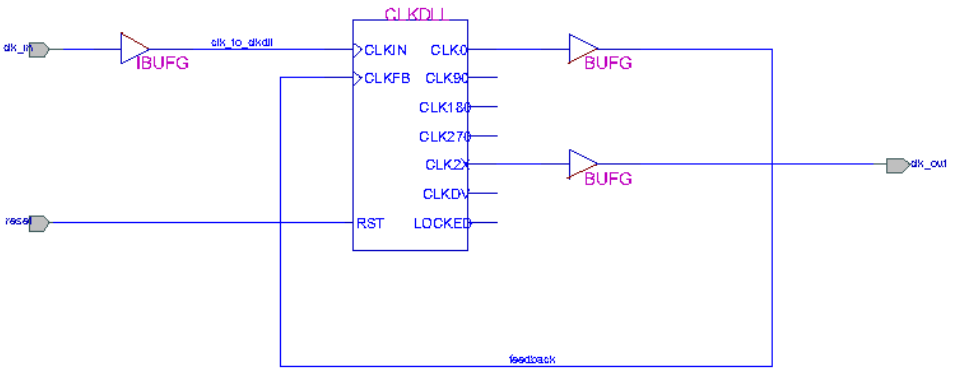


Figura 4.9: Clock×2

è scelto di inviare al *pattern-generator*, quale sincronismo per l'emissione dei dati, il segnale di *clock-enable* che si ottiene a partire dal clock di sistema[11].

Lo schema dell'interfaccia tra il modulatore e la sorgente dati è in Figura(4.10) cui corrisponde ***Data_Source_Interface.vhd*** *(Listato F.5.3)* , la

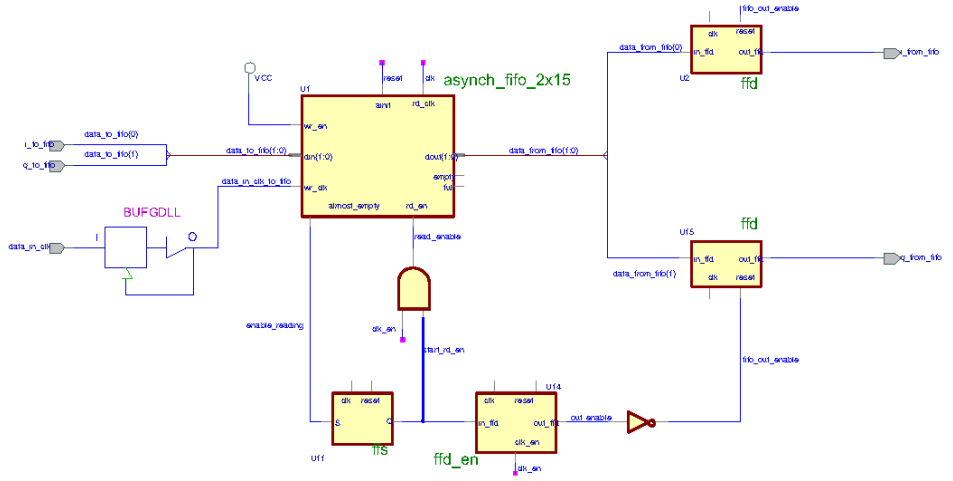


Figura 4.10: Interfaccia modulatore – sorgente dati

FIFO è stata implementata con un Core Xilinx per la cui simulazione si utilizza ***asynch_fifo_2x15.vhd*** *(Listato F.5.4)* , il clock in scrittura è quello proveniente dal *pattern-generator* mentre quello in lettura è il clock di sistema tuttavia l'abilitazione alla lettura è determinata dal segnale *clock-enable* proveniente dal ***counter.vhd*** *(Listato F.3.2)* . La logica che gravita intorno

[11] 40MHz per la scheda DINI sulla quale è stato implementato il modulatore

alla FIFO ha il compito di consentire la prima lettura soltanto dopo che essa sia stata parzialmente riempita, si deve cioè attendere che il segnale *almost-empty* passi per la prima volta al livello logico basso.

4.3.2.1 Risultati sperimentali

4.3.2.1.1 Test VHDL La medesima sequenza dati applicata al modello Matlab è stata applicata alla descrizione VHDL, l'uscita è poi nuovamente esportata in Matlab dove, dopo una conversione dalla rappresentazione in complemento a due al formato reale, ne viene prodotto e visualizzato lo spettro confrontandolo con quello ottenuto con il modello Matlab del modem, i due sostanzialmente coincidono come mostrato in Figura(4.11) riferita ad un modulatore interpolante 3 e che, per garantire conformità con

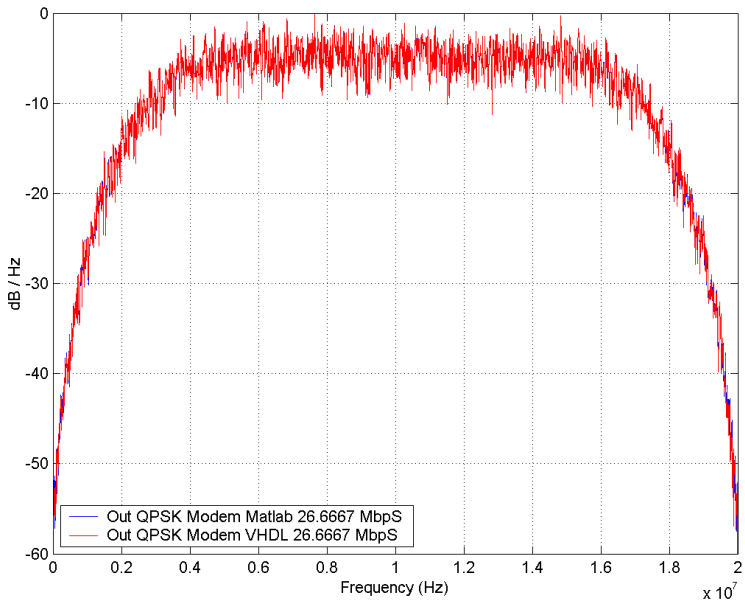


Figura 4.11: Confronto Modulatore x3 VHDL – Matlab

i risultati dell'implementazione su FPGA[12], opera ad una frequenza di clock di 40MHz. Per una verifica più puntuale lo stesso script ***QPSKModemPSDeBERVHDLvsMatlab.m*** *(Listato E.4.4)* realizza anche una misura di BER, a tal fine viene aggiunto del rumore gaussiano dovuto al canale ed il segnale risultante è poi applicato al demodulatore che richiede di esser tarato con la procedura inserita quale commento nello script, la corretta taratura del demodulatore consente di ottenere misure di BER praticamente identiche ai risultati teorici.

[12] la scheda DINI ospita un quarzo a 40MHz eventualmente modificabile

4.3.2.1.2 Test FPGA La sequenza da trasmettere è stata caricata nel *Pattern Generator* tramite il file di testo prodotto da ***CreaSequenzaPatternGenerator.m*** *(Listato E.2.2)* del quale si propone il seguente stralcio:

```
ASCII     000000
ASCDOWN
FORMAT:MODE FULL
FORMAT:CLOCK EXTERNAL, LEFifty
LABEL dummy, 24
LABEL data_in_SRRCxN_tx_Q, 4
LABEL rate_sel_x3, 4
LABEL reset, 4
LABEL data_in_SRRCxN_tx_I, 4
VECTOR
000000 0 0 1 0
000000 0 0 1 0
000000 0 0 1 0
000000 0 0 1 0
000000 0 0 1 0
000000 0 0 0 0
000000 0 0 0 0
000000 0 0 0 0
000000 0 0 0 0
000000 0 0 0 0
*M
000000 0 0 0 0
000000 1 0 0 0
000000 1 0 0 0
000000 0 0 0 1
000000 0 0 0 0
000000 0 0 0 1
000000 1 0 0 1
000000 1 0 0 1
000000 0 0 0 0
...... . . . .
000000 0 0 0 1
```

essa segnala al *pattern-generator* di predisporsi per ricevere dall'esterno il sincronismo per il clock associato ai dati emessi i quali sono allocati sui canali *data_in_SRRCxN_tx_I* e *data_in_SRRCxN_tx_Q* mentre *rate_sel_x3* imposta il data rate e *reset* consente l'inizializzazione della FPGA. Il segnale *reset* è al livello 1 soltanto nella sequenza che precede *M in quanto questa viene emessa una sola volta mentre la sequenza successiva viene ripetuta ciclicamente un numero infinito di volte.

Le connessioni tra *pattern-generator*, scheda DINI ed *analizzatore di stati logici* sono descritte nel seguente testo estratto dal file .ucf il quale viene utilizzato per immettere i vincoli fisici e temporali nella sintesi del progetto sulla FPGA Xilinx Virtex1000BG560-4:

```
################################################
NET "clk" LOC = "D17"; #clock a 40MHz interno
################################################
#    POD CLK PATTERN GENERATOR                 #
################################################
NET "data_in_clk" LOC = "A17"; #JP11 pin 12
NET "data_in_clk_sync" LOC = "C28"; #JP1 pin 20
################################################
#      POD 1 PATTERN GENERATOR                 #
################################################
NET "reset" LOC = "AE31"; #JP12 pin 10
NET "data_in_I" LOC = "AD29"; #JP12 pin 8
NET "data_in_q" LOC = "V29"; #JP12 pin 3
################################################
#      POD 2 PATTERN GENERATOR                 #
################################################
NET "rate_sel<0>" LOC = "V31"; #JP12 pin 4
NET "rate_sel<1>" LOC = "Y29"; #JP12 pin 6
#
################################################
#      POD 2 ANALIZZATORE STATI                #
################################################
NET "modulator_out<0>"  LOC = "C29"; #JP6 pin 1
NET "modulator_out<1>"  LOC = "A31"; #JP6 pin 3
NET "modulator_out<2>"  LOC = "B30"; #JP6 pin 5
NET "modulator_out<3>"  LOC = "C30"; #JP6 pin 7
NET "modulator_out<4>"  LOC = "D28"; #JP6 pin 9
NET "modulator_out<5>"  LOC = "F29"; #JP6 pin 13
NET "modulator_out<6>"  LOC = "H31"; #JP6 pin 15
NET "modulator_out<7>"  LOC = "P30"; #JP6 pin 19
NET "modulator_out<8>"  LOC = "A28"; #JP6 pin 4
NET "modulator_out<9>"  LOC = "C27"; #JP6 pin 8
NET "modulator_out<10>" LOC = "D32"; #JP6 pin 10
NET "modulator_out<11>" LOC = "E30"; #JP6 pin 12
NET "data_out_clk" LOC = "B29"; #JP1 pin 19
################################################
#        AZIONAMENTO MANUALE                   #
################################################
NET "count_clk_en_reset" LOC = "AK24"; #JP12 pin 15
```

Al fine di agevolare il raggiungimento dei vincoli temporali con stati scelti i pin della FPGA più vicini alle 12 BlockRAM utilizzate nel progetto, esse si trovano infatti distribuite in due colonne laterali.

L'*analizzatore di stati logici* consente di prelevare dall'uscita del modulatore un pacchetto di 64000 campioni adiacenti, applicandoli allo script Matlab ***QPSKModemPSDFPGAvsPSDMatlab.m*** *(Listato E.4.6)* sono stati ottenuti gli spettri mostrati nelle figure (4.12), (4.13) e (4.14), essi evidenziano una forte similitudine tra il risultato teorico ed il risultato sperimentale.

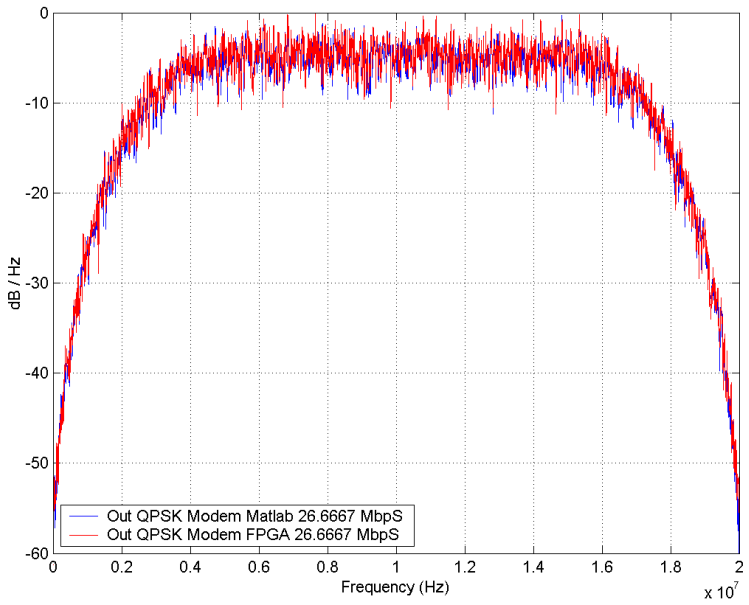


Figura 4.12: Confronto Modulatore x3 FPGA – Matlab

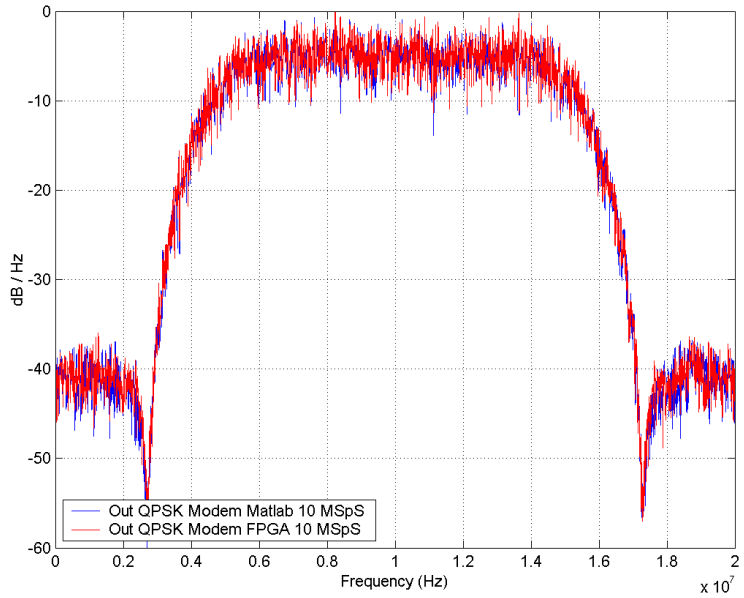


Figura 4.13: Confronto Modulatore x4 FPGA – Matlab

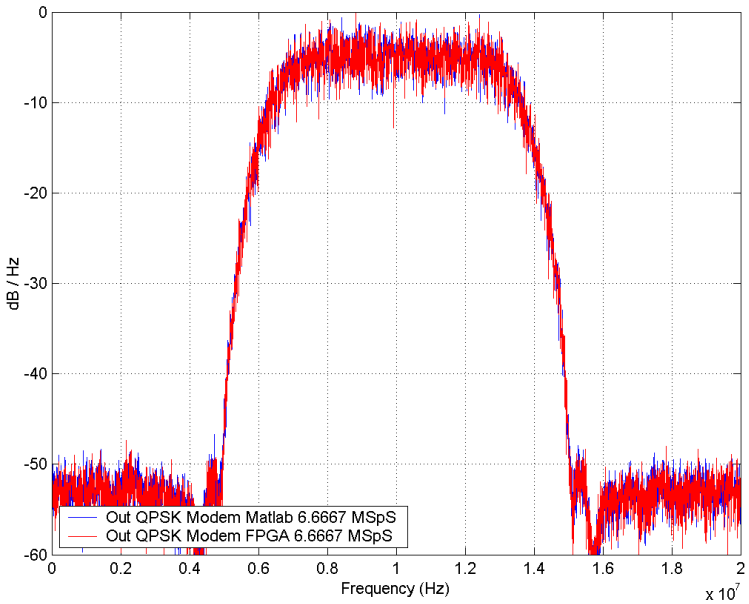


Figura 4.14: Confronto Modulatore x6 FPGA – Matlab