Dipartimento Politecnico di Ingegneria e Architettura (DPIA)
Corso di Calcolatori Elettronici

Iterative Multiplier

Studente: Lorenzo Zaccomer

Anno Accademico: **2021/22**

Estratto

Attività svolta per il corso di Calcolatori Elettronici tenuto dal Professore Mirko Loghi nell'anno accademico 2021/2022.

L'obiettivo è stato la progettazione di un moltiplicatore iterativo, a due ingressi, descritto in linguaggio VHDL; questo avendo a disposizione un moltiplicare da 2 bit sintetizzato automaticamente dal tool di sintesi, e sfruttarlo per ottenere il moltiplicatore desiderato, prima un'unità a 16 bit di uscita, e successivamente un'unità da 64 bit, tramite parametrizzazione.

Indice

1 Introduzione

Il moltiplicatore iterativo è stato quindi suddiviso in 3 unità separate:

- *selector* (2.1)
- *internal multiplier* (2.2)
- *resolver* (2.3)

Dove la prima unità, il **selector**, partendo da due operandi di dimensionalità pari a N, produrrà in uscita i due operandi da 4 bit per l'unità successiva, ovvero il **internal multiplier**, il quale quindi, sfruttando il moltiplicatore da 2 bit citato nella parte introduttiva, andrà a produrre un output da 8 bit, ed infine la terza ed ultima unità, il **resolver**, andrà ad accumulare gli output dell'internal multiplier per andare a generare in uscita il risultato complessivo, avente dimensionalità pari a 2N.

La figura 1.1 illustra la cascata dei collegamenti tra i tre blocchi, si vede come il resolver e il selector siano interconnessi tra di loro mediante due opportuni segnali, ma questo dettaglio verrà ripreso più avanti; si evidenzia il fatto che tutti i blocchi hanno dei segnali comuni con la stessa funzionalità operativa, ovvero:

- DATAIN: abilita l'inserimento di nuovi dati solamente se il blocco è pronto a ricevere nuovi dati
- DATAOUT: ci permette di sapere quando il blocco ha effettivamente svolto il suo compito, per cui se è pari ad uno, avremo in uscita l'output atteso
- READY: è un segnale di test, ci permette di sapere quando il blocco è pronto a ricevere nuovi dati

Infine nelle prossime righe si andrà a caratterizzare nel dettaglio i singoli blocchi.

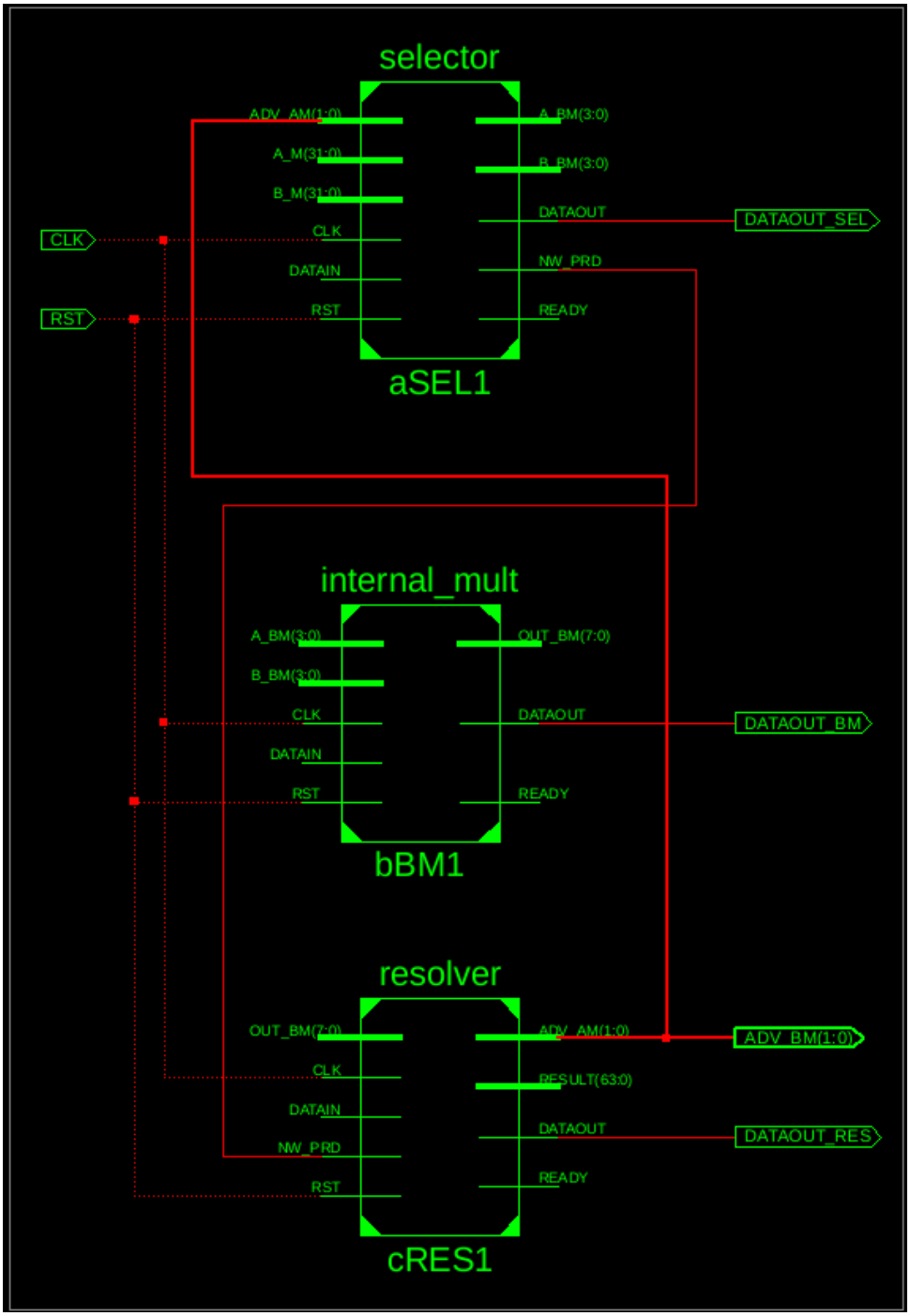

Figura 1.1: Prima rappresentazione circuitale dei tre blocchi tra loro collegati

2 Blocchi costituenti

2.1 Selector

Il selector ha il compito di sezionare i due segnali di input a blocchi di 4 bit alla volta, attività la quale viene svolta in coordinazione con il blocco *resolver*.

La parametrizzazione di tale modulo può essere svolta su tre parametri:

- N: lunghezza operandi
- ITERATIONS: numero di prodotti dei sotto-blocchi da 4 bit (dipende da N)
- DIM_{CNT}: lunghezza per la stringa in binario associata al parametro ITERATIONS, viene impiegata nella ctrl-unit tramite un'opportuna funzione (dipende dal valore decimale associato a ITERATIONS).

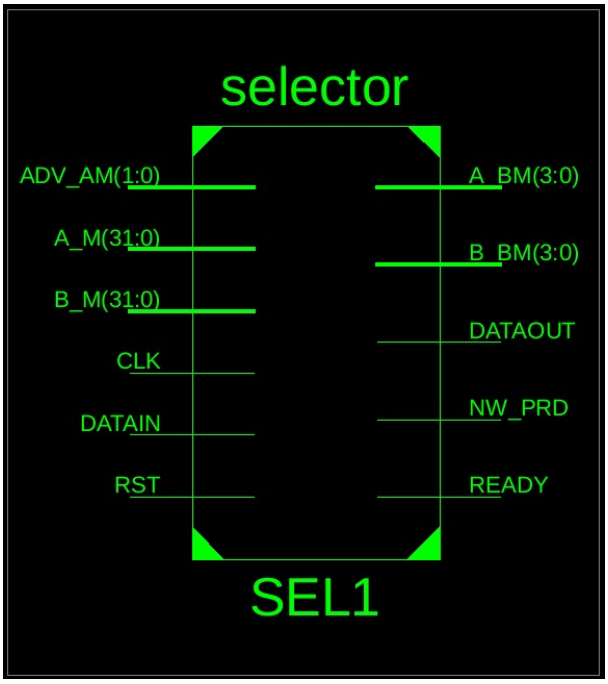

Figura 2.1: Black box selector

Dalla figura 2.1 si possono osservare:

- due input: A_M e B_M, i quali avranno una lunghezza variabile e fissata dal parametro N
- due output: A_{BM} e B_{BM}, di lunghezza fissata pari a 4 bit, i quali fungono da parametri di ingresso del modulo di moltiplicazione vera a propria
- ADV_{BM} (uguale a "11" di default) e NW_{PRD} (di valore pari a '0' di default), i quali sono dei segnali specifici in collegamento con il resolver

2.1.1 Funzionamento

Ma come opera questa coordinazione?

Per fare ciò vengono impiegati i segnali ADV_{BM} e NW_{PRD}, rispettivamente di input e output per il blocco selector, e dualmente di output e input invece per il blocco resolver, e in base ai valori che tali segnali assumono, verranno svolte opportune e giustificate operazioni.

Si prenderà ora come riferimento l'esempio 2.1.

All'inizio abbiamo i due input in ingresso al selector, il quale in uscita andrà a produrre la prima sequenza indicata in esempio, ovvero i primi 4 LSB di entrambi.

A questo punto tale modulo si mette in attesa finché il resolver non lo sblocca con un nuovo valore di ADV_{BM}, il quale potrà essere "00" oppure "01" [1], questa attesa è necessaria perché bisogna attendere l'esecuzione di due operazioni, l'operazione di moltiplicazione eseguita dal modulo intermedio e il salvataggio di quest'ultimo risultato da parte del resolver, le quali non avverranno in un solo ciclo di clock, altrimenti il selector andrebbe a ruota libera nel sezionamento degli input.

Una volta che il resolver ha salvato il risultato parziale, andrà a fissare il valore di ADV_{BM} pari a "01", per cui significa che la stringa A_{BM} andrà shiftata di 4 bit verso sinistra, mentre B_{BM} dovrà rimanere costante; la conferma dell'esecuzione di tale operazione la si osserva con il cambio del valore di NW_{PRD}, posto ad 1 dal selector, così facendo il resolver torna in una condizione valida per leggere l'input successivo.

A questo punto in uscita avremo la seconda sequenza, dove appunto si nota che la stringa B_{BM} è costante; rimanendo coerenti al flusso di esecuzione descritto prima, il nuovo valore di ADV_{BM} sarà pari a "00", per cui è necessario ripristinare l'intera stringa A_{BM} (altrimenti un nuovo shift eseguito dal selector produrrebbe la stringa "0000"), inoltre viene eseguito uno shift di 4 bit a sinistra dell'operando B_{BM}.

L'operazione complessiva all'interno di questo modulo avrà un numero di iterazioni pari al valore fissato dalla costante ITERATIONS, in questo caso pari a 4.

[1] la motivazione verrà meglio definita nella parte dedicata al resolver

Esempio 2.1 *Coordinazione moduli selector e resolver*

A_{BM} = ***11111010*** *e* B_{BM} = ***11010101***

ITERATIONS = 4

1a sequenza: A_{BM} = *1010 e* B_{BM} = *0101*

2a sequenza: A_{BM} = *1111 e* B_{BM} = *0101*

3a sequenza: A_{BM} = *1010 e* B_{BM} = *1101*

4a sequenza: A_{BM} = *1111 e* B_{BM} = *1101*

2.2 Internal Multiplier

Tale modulo, costituito da due input da 4 bit, esegue il prodotto tra questi due per portare in uscita quindi un output da 8 bit.

La figura 2.2 definisce la sua *black box*:

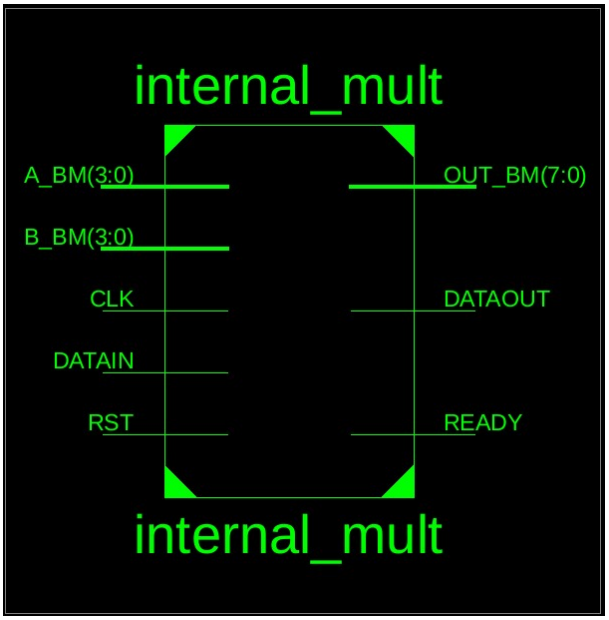

Figura 2.2: Black box internal multiplier

Tale modulo è indipendente dagli altri due moduli, perciò volendo può essere usato separatamente.

L'operazione di moltiplicazione è stata implementata seguendo il classico approccio *shift and add*.

2.2.1 Funzionamento

Si prenderà ora come riferimento l'esempio 2.2.

Salvati i due operandi di input (A_{BM} e B_{BM}), si andranno ad estrarre inizialmente i primi due bit di ognuno e verranno salvati sui registri, OP_A e OP_B, successivamente verrà svolta una moltiplicazione, con un risultato di 4 bit (1a sequenza).

A questo punto, alla successiva iterazione (2a sequenza), gestita mediante un contatore (CNT_{BM}) si shifta a destra di 2 bit l'operando A_{BM} e si effettua una nuova moltiplicazione tra i due operandi OP_A e OP_B, poiché ora OP_A avrà un nuovo valore, tale risultato viene infine viene a sua shiftato di due bit a destra e sommato con il precedente, andando ad ottenere R_{PM}.

Infine tale iterazione viene svolta altre due volte, ovvero quando si arriva alla condizione CNT_{BM} = M (ovvero 4), l'ultimo risultato parziale viene salvato in ACC_{BM}, shiftato a sinistra di due bit per ottenere il risultato complessivo dell'intera operazione.

Esempio 2.2 *Esempio moltiplicazione binaria mediante shift and add*

A_{BM} = ***1011*** *e* B_{BM} = ***1010***

1a sequenza: OP_A = 11, OP_B = 10, OP_R = 0110

2a sequenza: OP_A = 10, OP_B = 10, OP_R = 0100

R_{PM} = 010110

3a sequenza: OP_A = 11, OP_B = 10, OP_R = 0110

4a sequenza: OP_A = 10, OP_B = 10, OP_R = 0100

ACC_{BM} = 010110 → 01011000 *(shift a sinistra di 2 bit)*

Fine iterazione:

$$\begin{array}{r} 010110 \\ +\ 01011000 \\ \hline 01101110 \end{array}$$

2.3 Resolver

Tale modulo ha il compito di salvare i risultati parziali ottenuti dal moltiplicatore interno durante le varie iterazioni, e produrre alla fine il risultato finale della moltiplicazione.

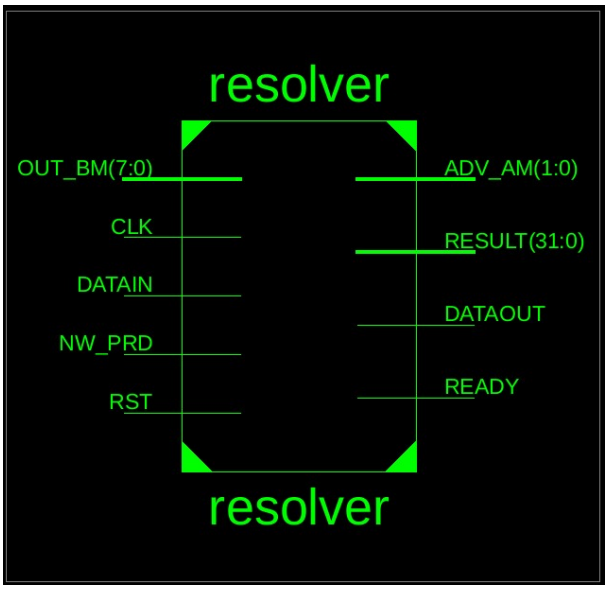

Figura 2.3: Black box resolver

Al suo interno quindi vengono svolte delle operazioni di *shift and add* tra i vari risultati parziali ottenuti a partire dai MSB dei registri, questo per mantenere costante le dimensionalità delle operazioni.

La parametrizzazione di tale modulo può essere svolta su tre parametri:

- N: lunghezza operandi
- REPETITION: numero di operazioni complessive che tale modulo deve andare a svolgere
- DIM_{CNT}: lunghezza per la stringa in binario associata al parametro REPETITION, viene impiegata nella ctrl-unit tramite un'opportuna funzione (dipende dal valore decimale associato a REPETITION).

Dall'immagine 2.3 si possono osservare:

- un input: è l'uscita del moltiplicatore interno
- un output: è il risultato complessivo della moltiplicazione
- ADV_{BM} (uguale a "11" di default) e NW_{PRD} (di valore pari a '0' di default), i quali sono dei segnali specifici in collegamento con il selector

2.3.1 Funzionamento

Si consideri l'esempio 2.3.

Ad ogni iterazione, il risultato parziale generato dal moltiplicatore interno viene salvato in un registro interno (BM) e sommato con il precedente risultato (operazione da 8 bit), dove tranne nel caso iniziale, dove i registri sono inizializzati a zero, esso viene precedentemente shiftato di 4 bit verso destra.

Se la sotto-sequenza non è completa (esempio siamo al punto A3*B1), allora il resolver porrà ADV_{BM} pari a "01" e questa condizione verrà mantenuta fin quando NW_{PRD} non sarà uguale a '1', ciò avverrà quando il selector avrà correttamente letto il nuovo valore di ADV_{BM} e sarà andato nel relativo stato (vedasi ASM Chart), questa condizione significa che il selector deve mantenere il sotto-blocco di B (esempio B1) e generare un nuovo sotto-blocco di A (esempio A2).

Questo viene fatto fino a quando non si ottiene una sotto-sequenza completa, ovvero quando P_{SHIFT} = REPETITION, allora essa viene salvata svolgendo in maniera analoga le medesime operazioni viste precedentemente, ma con dimensionalità differente e in funzione della dimensionalità degli operandi; ma in questo caso, rispetto a prima, ADV_{BM} sarà pari a "00", significa che ora il selector deve ripristinare la stringa A (altrimenti si avrebbe sempre "0000") e generare un nuovo sottoblocco di B (esempio B_2).

Questo ciclo di operazioni viene svolto fin quando N_{SHIFT} = REPETITION, il quale è il contatore relativo alle sotto-sequenze complete.

Esempio 2.3 *Esempio salvataggio operandi modulo resolver*

A = ***0010 1000 0110 0001*** *e B =* ***1000 0111 0010 1001***

1a sotto-sequenza: prodotto tra B_1 *= 1001 e sotto-blocchi da 4 bit di A, ovvero* A_1 *= 0001 -* A_2 *= 0110 -* A_3 *= 1000 -* A_4 *= 0010*

$$\begin{array}{r} \textit{(A1*B1) 0000 1001} \\ \textit{(A2*B1) + 0011 0110 0000} \\ \textit{(A3*B1) + 0100 1000 0000 0000} \\ \textit{(A4*B1) + 0001 0010 0000 0000 0000} \\ \hline \textit{0001 0110 1011 0110 1001} \end{array}$$

2a sotto-sequenza: 0000 0101 0000 1100 0010

3a sotto-sequenza: 0001 0001 1010 1010 0111

4a sotto-sequenza: 0001 0100 0011 0000 1000

Fine iterazione:

$$\begin{array}{r} \textit{0000 0101 0000 1100 0010} \\ \textit{0000 0101 0000 1100 0010 0000} \\ \textit{0001 0001 1010 1010 0111 0000 0000} \\ \textit{0001 0100 0011 0000 1000 0000 0000 0000} \\ \hline \textit{0001 0101 0101 0001 1001 1110 1000 1001} \end{array}$$

3 Iterative Multiplier

Si passa quindi alla descrizione del moltiplicatore iterativo, obiettivo di questo progetto, la quale sarà più breve e coincisa rispetto alle precedenti, essendo questa un mero collegamento tra le precedenti unità.

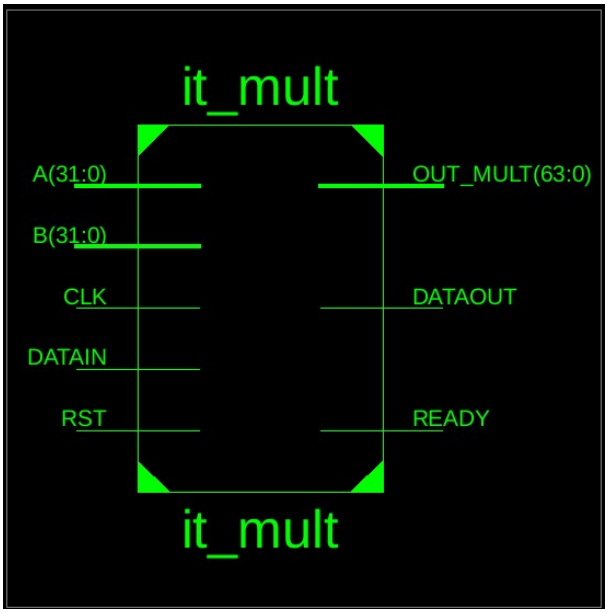

Figura 3.1: Black box iterative multiplier

Dall'immagine 3.1 si possono osservare gli input e l'output che si aspetta da un moltiplicatore.

Prendendo di riferimento il relativo chart, vengono caricati in due registri interni gli operandi di ingresso, e allo stato successivo viene posto ad '1' il segnale DATAIN relativo al selector ($DATAIN_{SEL}$, significa quindi che i dati sono stati correttamente caricati e pronti per essere manipolati.

A questo punto allo stato successivo viene attivato il selector, inoltre il segnale $DATAIN_{SEL}$ viene posto a '0', può essere una scelta inusuale ma questo viene fatto questo perché il selector non esegue le sue operazioni in un ciclo di clock, perciò tale segnale rimane a '1' per un solo ciclo, in modo tale che quest'ultimo possa a sua volta salvare gli operandi correttamente, oltre ad evitare che, per via del loop, il selector veda nuovamente dei dati in ingresso quando non necessario.

Conclusa la sua (temporanea) operazione, il selector rimarrà in attesa di una nuova interazione, questo tramite il segnale ADV_{BM}, nel frattempo gli output appena prodotti vengono salvati in due registri interni e passati poi al moltiplicatore interno, il quale verrà attivato in maniera analoga al modulo precedente.

Si passa a questo punto al modulo resolver, il quale opera come descritto nel sottocapitolo dedicato, dal chart si osserva che se ADV_{BM} è pari a "00" o "01", allora si torna allo stato $EXEC_{SEL}$, perciò il selector dovrà produrre due nuovi input per il moltiplicatore interno.

A questo punto, il resolver ha concluso tutte le sue operazioni e prodotto in uscita il risultato complessivo della moltiplicazione, il quale coinciderà con l'output del moltiplicatore iterativo.

4 Sviluppo

Ho scelto il classico approccio *divide et impera* per la progettazione di questo moltiplicatore, così facendo ho ridotto la complessità progettuale totale, questo è stato fatto sviluppando i moduli come unità a sé stanti, avendo ben chiaro in mente cosa dovessero svolgere, i quali combinati insieme mi hanno permesso di ottenere il moltiplicatore iterativo richiesto.

Successivamente sono stati parametrizzati in modo tale da essere adattabili a diverse dimensionalità di stringhe in ingresso.

5 Simulazioni

Il modulo di moltiplicazione interno, e successivamente il moltiplicatore iterativo nella sua interezza, concluse le operazioni di sviluppo in VHDL, sono stati testati mediante il moltiplicatore sintetizzato automaticamente da ModelSim, ovvero lo standard multiplier, presente nel repository, e forniti gli stessi input ad entrambi, hanno fornito gli stessi risultati.

Il modulo che risulta più lento è il moltiplicatore interno, poiché impiega molti più cicli di clock per completare le sue operazioni rispetto agli altri due moduli.

6 Sintesi

La sintesi, su FPGA, è stata svolta tramite il programma Xilinx ISE, indicando come scheda di valutazione la Xilinx Spartan-6, in particolare il modello *xc6slx45-3csg324*; nella cartella reports (link) sono presenti i file generati, sia per i singoli moduli, sia per il modulo complessivo, in particolare i report presenti sono i *translation*, *map*, *place and route*.

7 Conclusioni

I ragionamenti svolti nel loro complesso sono risultati sensati, questo perché a parte piccole dimenticanze o sistemazioni minime nel codice, le unità sviluppate separatamente prese hanno prodotto i risultati attesi durante la fase di simulazione su ModelSim.

Un possibile miglioria, o piccolo dettaglio estetico, è il segnale ADV_{BM}, il quale volendo potrebbe essere un semplicemente un bit '0' oppure '1', e usare il '-' al posto del valore "11", ma funzionalmente svolge comunque la sua funzione.

Infine se ci volesse porre l'obiettivo di rendere il moltiplicatore più veloce, si potrebbe andare ad operare inserendo un riconoscitore di sequenza, il quale va' a riconoscere tra i due operandi delle sequenze note, come per esempio "0000", oppure "0010", in questo modo, invece di svolgere ugualmente tutto il flusso di esecuzione, esso venga semplicemente bypassato producendo un semplice risultato noto (primo caso tutto zero, nel secondo un semplice shift a sinistra di 1 bit); ma ciò esula completamente dall'obiettivo di questo progetto.