Matrix Multiplier

Manuelli Alessio - Bernini Mattia

Introduzione

Il progetto consiste nell'implementare un'architettura in grado di effettuare il prodotto tra due matrici su una scheda di sviluppo ZYbo. Questo tipo di prodotto è particolarmente complesso, in quanto richiede una serie di operazioni successive CPU heavy.

Descrizione del Circuito

Un moltiplicatore di matrici è un circuito digitale che consente di calcolare il prodotto tra due matrici, operazione fondamentale in molti ambiti di elaborazione dati. Nella sua forma base, questo circuito riceve in ingresso due matrici di dimensioni compatibili e restituisce una matrice prodotto, dove ogni elemento è ottenuto come somma ponderata dei prodotti degli elementi corrispondenti delle righe della prima matrice e delle colonne della seconda.

Possibili Applicazioni

Il moltiplicatore di matrici trova applicazione in una vasta gamma di settori, tra cui:

- **Elaborazione di segnali digitali (DSP):** Molte operazioni di filtraggio e trasformazioni, come la trasformata di Fourier discreta (DFT), richiedono la moltiplicazione di matrici.

- **Computer Graphics:** La trasformazione e manipolazione di immagini e modelli 3D utilizza intensamente la moltiplicazione di matrici per operazioni di rotazione, scala e traslazione.

- **Intelligenza Artificiale e Machine Learning:** Le reti neurali, particolarmente nelle loro forme più complesse, sfruttano la moltiplicazione di matrici per calcolare i pesi e i valori di output durante l'addestramento e l'inferenza.

- **Sistemi di controllo:** Le equazioni che regolano i sistemi dinamici, in particolare nelle applicazioni di controllo automatico, spesso richiedono la manipolazione di matrici per la modellazione e l'analisi.

Possibili Architetture

Il moltiplicatore di matrici può essere implementato utilizzando diverse architetture, ciascuna con i propri trade-off in termini di complessità hardware, velocità di calcolo, e utilizzo di risorse:

- **Architettura Basata su Array:** In questa configurazione, una griglia di moltiplicatori e addizionatori è disposta in un array bidimensionale. Questa architettura è altamente parallela e permette un'elaborazione molto veloce, ma richiede un'ampia area di circuito.

- **Architettura a Pipeline:** Questa struttura suddivide l'operazione in più stadi, consentendo di eseguire la moltiplicazione e somma delle righe e colonne in modo sequenziale e con minore consumo di area, a scapito della latenza.

- **Architettura a Memoria:** Utilizza un approccio iterativo in cui un singolo modulo di moltiplicazione e somma viene utilizzato ripetutamente per calcolare ogni elemento della matrice risultato. Questo approccio è più efficiente in termini di area ma più lento.

- **Architettura Sistolica:** In questo caso, il circuito è composto da un array di processori che elaborano i dati in maniera sincrona, passando i risultati parziali tra i nodi adiacenti. Questo tipo di architettura è particolarmente efficiente in termini di parallelismo e consente un alto throughput.

Scopo del Progetto

Il progetto mira a progettare e simulare un moltiplicatore di matrici efficiente. L'obiettivo finale è ottenere un circuito ottimizzato che possa essere integrato in sistemi più complessi per le applicazioni descritte.

Implementazione dell'architettura e Test-plan

E' necessario progettare un circuito che esegua la moltiplicazione tra una matrice A di dimensioni N×M e una matrice B di dimensioni M×K. Il risultato sarà una matrice C di dimensioni N×K.

Le dimensioni N, M, e K possono variare, ma sono limitate da un parametro generico denominato **`MAX_SIZE`**. Ogni elemento delle matrici si considera come rappresentato su un certo numero di bit determinato dal parametro generico **`Nbit`**.

Il circuito presenta i seguenti segnali di ingresso e uscita:

- **clk:** clock del circuito.

- **rst:** reset.

- **A_row, B_column:** segnali per l'inserimento delle righe di A e delle colonne di B.

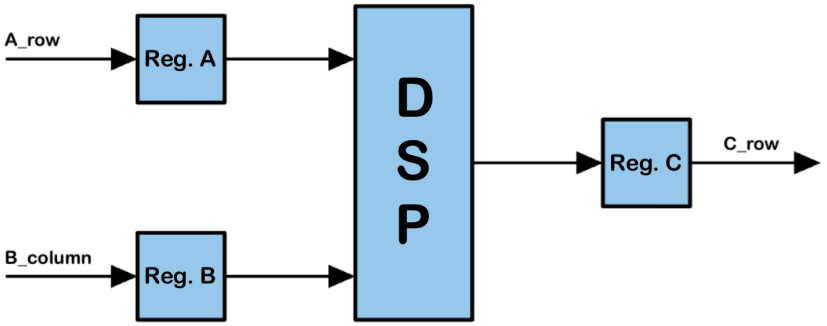
- **C_row:** segnali per l'uscita delle righe della matrice risultante C.

- **{N, M, K}_size:** segnali che indicano il valore di N, M, K.

- **A_valid, B_valid:** segnali di validità per l'ingresso di A e B.

- **C_valid:** segnale di validità per l'uscita della matrice C.

Lo schema a blocchi che mostra il funzionamento dell'architettura può essere così pensato:

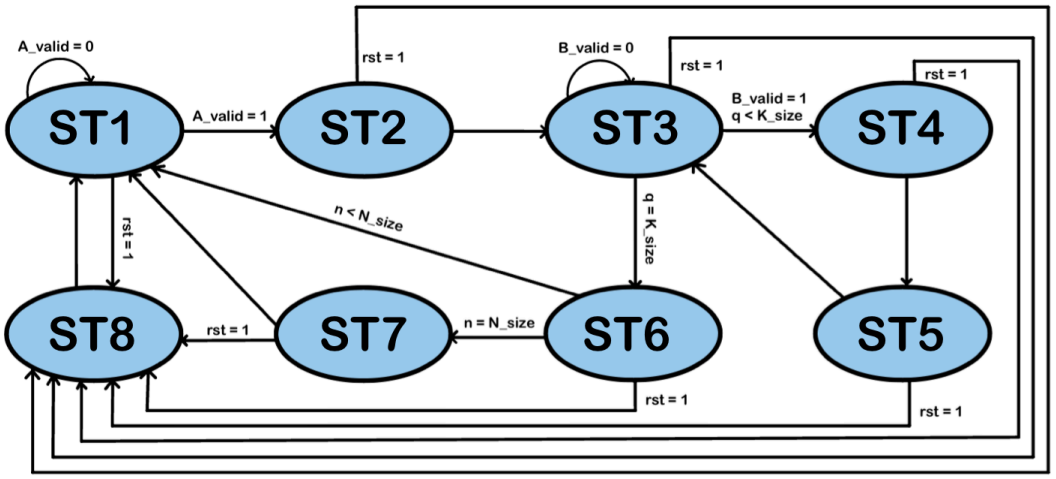


La riga A e la colonna B vengono memorizzate in due registri distinti, le cui uscite vengono collegate ad una rete di elaborazione (somme di prodotti). Questa permette di calcolare tutti i singoli elementi della riga C, i quali vengono memorizzati in un registro apposito. Una volta terminato il calcolo della riga C, questa sarà disponibile in uscita.

Si va ora ad analizzare l'approccio utilizzato per la progettazione del circuito. Si è scelto di basarsi sul funzionamento di una macchina a stati finiti. I segnali di ingresso A_row e B_column sono delle stringhe di bit, ciascuna costituita da un certo numero di bit. Tale numero di bit risulta essere pari al prodotto Nbit*M. Sostanzialmente si considerano i vari elementi di riga e colonna come posti serialmente in un'unica stringa di bit. Si è incentrato il design della struttura facendo riferimento al generico MAX_SIZE. Nel caso in cui si avesse: M < MAX_SIZE, si riempirebbero con degli zeri gli elementi in eccesso. Un ragionamento analogo può essere fatto considerando il segnale di uscita C_row, tenendo conto del fatto che la dimensione di tale stringa di bit deve poter coprire il caso più dispendioso in termini di bit.

Il circuito progettato prevede che vengano inizialmente poste in ingresso una riga e una colonna. Successivamente, con un'opportuna sincronizzazione che si illustrerà in seguito, è necessario porre in ingresso le successive colonne della matrice B, in modo da poter calcolare la corrispondente riga della matrice C. Si può dunque osservare che nell'architettura non è prevista la memorizzazione interna delle matrici A e B, ma di volta in volta è necessario pilotare opportunamente il circuito, in modo da così ottenere le varie righe della matrice risultante.

La macchina a stati è così strutturata:

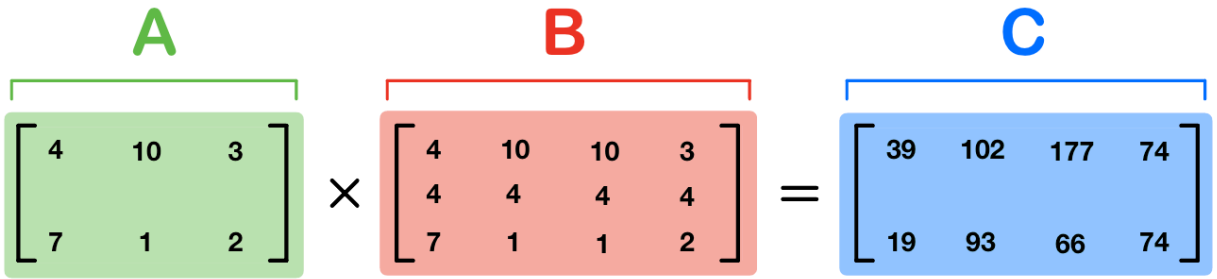


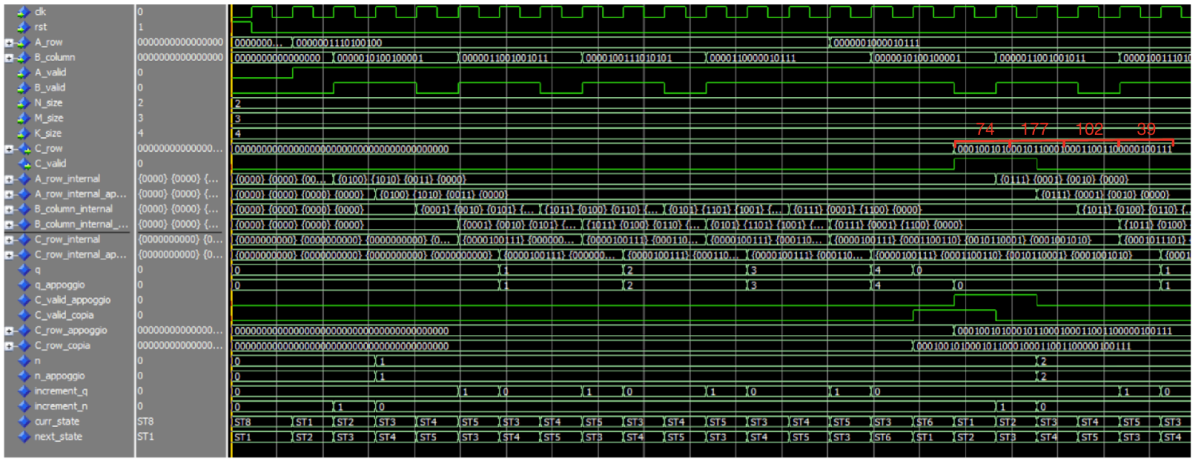
Si ha un primo stato, denominato **ST1**, in cui si attende che l'utilizzatore renda disponibile una riga di ingresso, ovviamente in concomitanza con il segnale A_valid. Quest'ultimo segnale garantisce la validità della riga in termini del suo numero di elementi.
Si ha quindi lo stato **ST2**, in cui si effettua un'operazione di riorganizzazione degli elementi della riga di ingresso. Ciò che si fa è trasferire i vari elementi della riga su un array di vettori logici dimensionato opportunamente, in modo da facilitare le operazioni successive.
Per quanto riguarda gli stati **ST3** e **ST4** si possono fare delle considerazioni analoghe a quelle appena fatte considerando però una colonna di ingresso.
Una volta che risultano presenti la riga e la colonna di ingresso si passa allo stato **ST5**, in cui si va a calcolare l'elemento della riga C effettuando il prodotto riga per colonna.
Successivamente, con un opportuno segnale di conteggio, si verifica che il calcolo della riga sia completato o meno. Nel caso in cui non sia completato, si attende che venga caricata una nuova colonna.
Ogni volta che viene completato il calcolo di una riga della matrice C, nello stato **ST6** viene verificato che sia stato completato il calcolo della matrice risultante con un opportuno segnale di conteggio. Nel caso in cui ciò sia vero, viene raggiunto lo stato **ST7**, in cui viene informato l'utente con una particolare configurazione dell'uscita che il calcolo risulta essere completato.
Infine, lo stato **ST8** non costituisce altro che lo stato riservato all'operazione di reset, necessaria ovviamente all'innesco del circuito.

Testbench

Durante la fase di test si è considerato il caso per cui MAX_SIZE = 5. In particolare si sono considerate le seguenti matrici:

A: NxM = 2x3 B: MxK = 3x4 C: NxK = 2x4

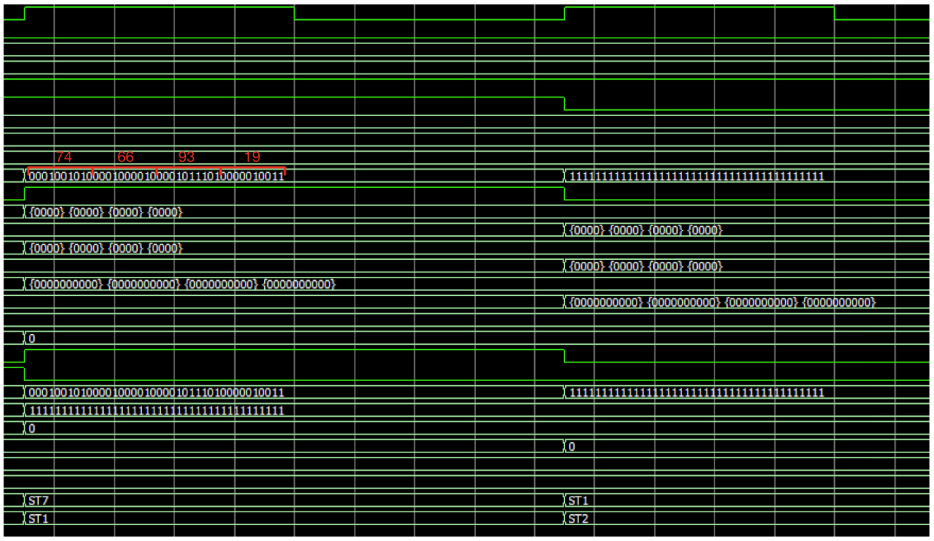




In figura è possibile osservare il risultato della simulazione della moltiplicazione e alcuni

aspetti chiave del processo. Inizialmente, vi è una fase di reset in cui tutti i segnali vengono azzerati. Successivamente, si procede con il calcolo della prima riga della matrice C: si inserisce la prima riga della matrice A e, a seguire, le diverse colonne della matrice B, in modo da calcolare i vari elementi di C. Una volta completato il calcolo della prima riga, questa viene inviata in uscita, seguita dal segnale C_valid attivo (a 1) per un ciclo di clock.

Dopo il calcolo della prima riga, il processo si ripete in maniera identica per la seconda riga e così via, fino a raggiungere lo stato finale, in cui tutti i segnali vengono nuovamente azzerati. L'uscita viene inviata su C_row, C_valid viene portato a 1, e successivamente anche C_row viene impostato a tutti 1, segnalando all'utente che il calcolo dell'intera matrice è terminato.

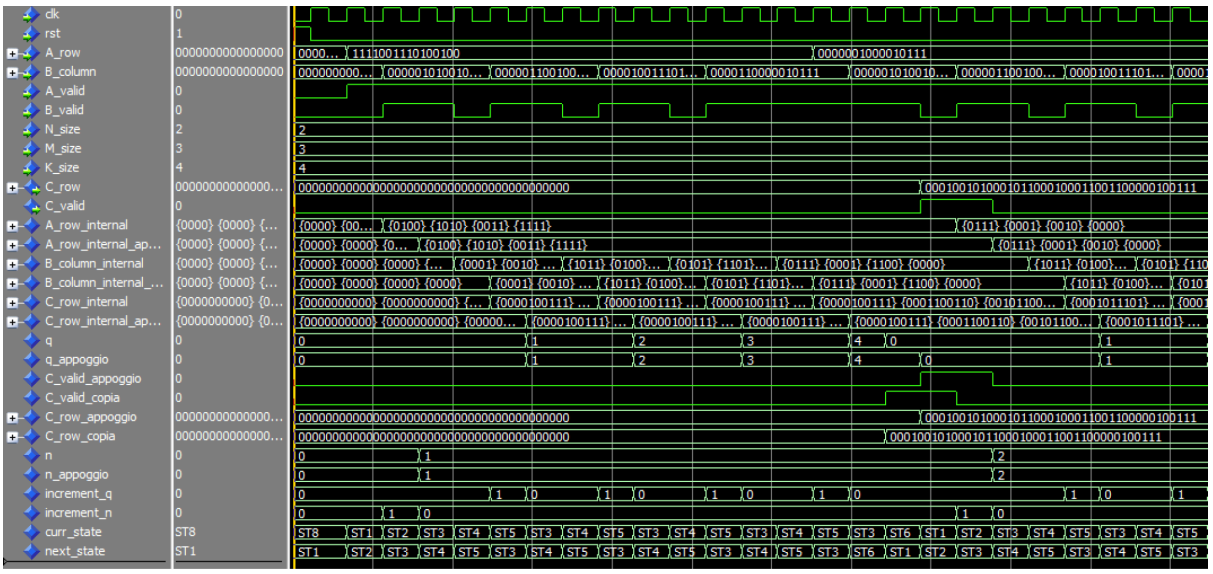


Possibile errore da parte dell'utente e relativa soluzione

Si supponga che l'utente consideri M = 3 (3 elementi per la riga A) e che inserisca erroneamente anche un quarto elemento, come mostrato in figura.

```
A_row <= "1111001110100100";
A_valid <= '1';
```

Il sistema è resistente a questo tipo di errori. Questo perché l'operazione di elaborazione è stata abilitata tramite un if solamente per gli elementi validi nella riga di B. Se quindi, ad esempio, si associano a questa riga di A le stesse colonne di B dell'esempio precedente, si può notare come il risultato sia rimasto invariato, proprio perché il quarto elemento della riga di A non è stato preso in considerazione.

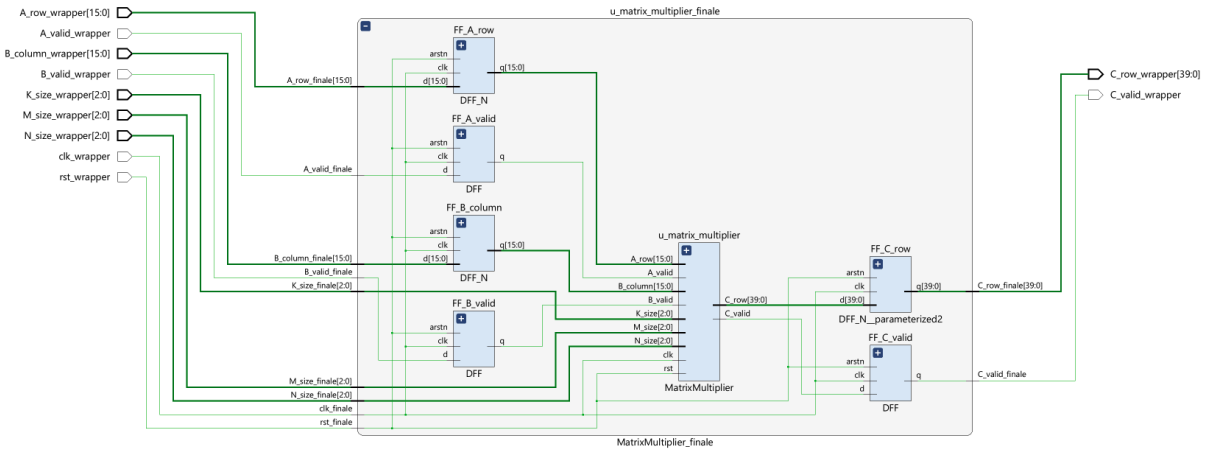


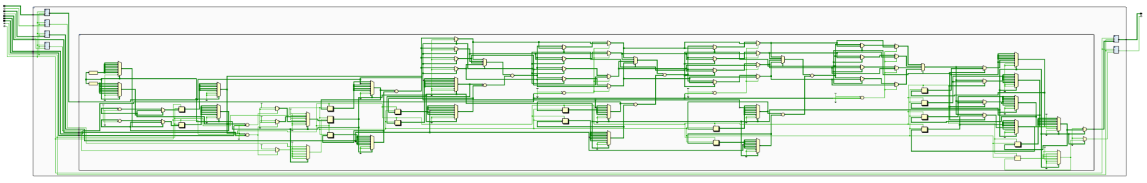
Sintesi logica

Completato lo studio della fase di simulazione per verificare il corretto funzionamento logico con Modelsim, è possibile analizzare l'implementazione dell'architettura su Vivado.

La sintesi su Vivado ha richiesto alcune modifiche al codice VHDL poiché venivano generati degli warning. In particolare, è stata evitata l'inferenza di latch utilizzando variabili di appoggio che hanno permesso di assegnare il valore corretto alla variabile principale nel caso in cui, all'interno della macchina a stati, non le fosse assegnato alcun valore. Dopo queste modifiche, sono emersi ulteriori warning riguardanti i loop combinatori, dovuti al fatto che le variabili utilizzate per contare gli eventi non erano sincronizzate con il clock. Per risolvere, sono state sincronizzate queste variabili con il clock.

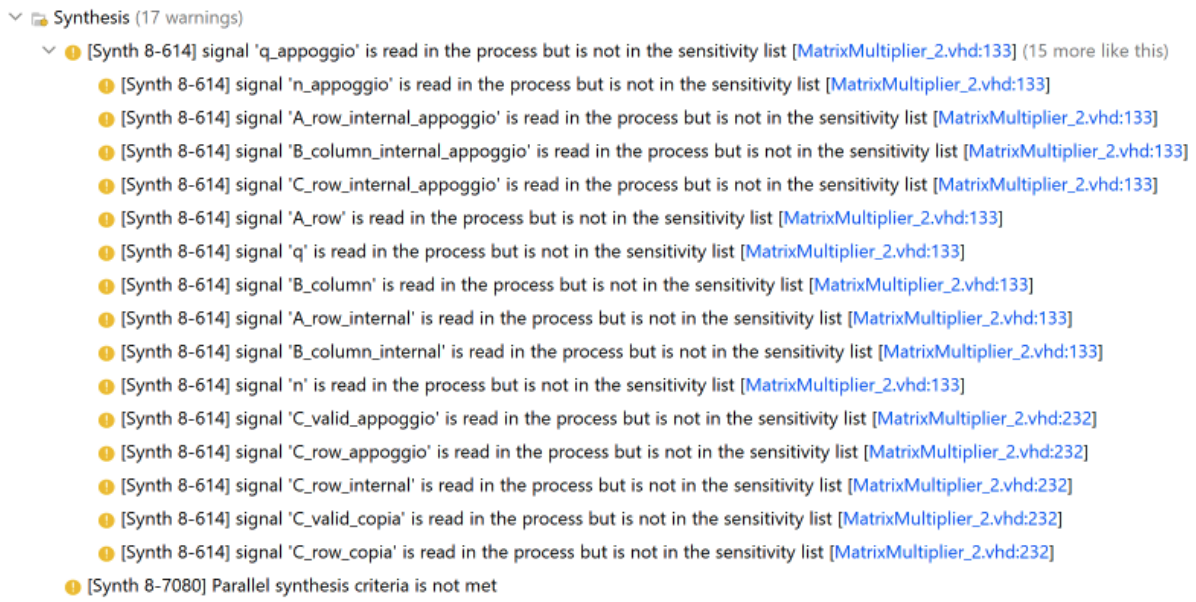
Elaborated design





Dall'osservazione dell'elaborated design è possibile notare come il sintetizzatore abbia interpretato il circuito come precedentemente descritto, ovvero una serie di registri e operazioni di somma e prodotto.

Dopo aver fatto la sintesi risultano presenti i seguenti warning:



Gli warning in questione indicano che i segnali riportati vengono letti all'interno di un processo, ma non sono stati inclusi nella relativa sensitivity list. La motivazione è legata al fatto che la maggior parte di questi sono segnali di appoggio e quindi non devono essere posti in sensitivity list. Questi infatti devono far riferimento al segnale originale associato. Per quanto riguarda gli altri segnali, come A_row e B_column, non sono stati posti in sensitivity list in quanto per essi si fa riferimento ai segnali A_valid e B_valid: si considerano una riga e una colonna come pronte solo quando tali segnali risultano essere ad un livello logico alto.

Report sul timing

Il clock utilizzato presenta una frequenza pari a 62.5 MHz. Nella pre-implementazione è stato ottenuto nel Timing Report un Worst Negative Slack di 3.388 ns.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,388 ns	Worst Hold Slack (WHS):	0,254 ns	Worst Pulse Width Slack (WPWS):	7,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	226	Total Number of Endpoints:	226	Total Number of Endpoints:	260

All user specified timing constraints are met.

A seguito dell'Implementazione si è ottenuto il Timing Report sotto riportato, il quale tiene conto anche delle interconnessioni. E' possibile notare come il WNS sia aumentato rispetto a ciò che si aveva nella fase di pre-implementazione.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,053 ns	Worst Hold Slack (WHS):	0,132 ns	Worst Pulse Width Slack (WPWS):	7,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	226	Total Number of Endpoints:	226	Total Number of Endpoints:	260

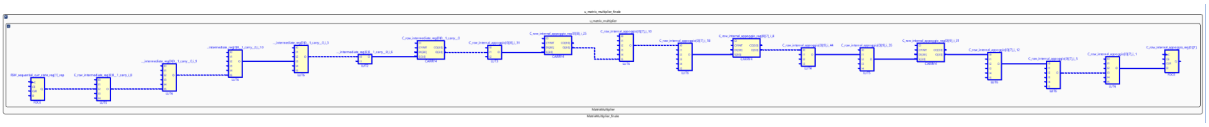
All user specified timing constraints are met.

Con questo valore finale di Worst Negative Slack ottenuto è possibile calcolare una possibile massima frequenza di clock, che risulterà essere superiore a quella utilizzata.

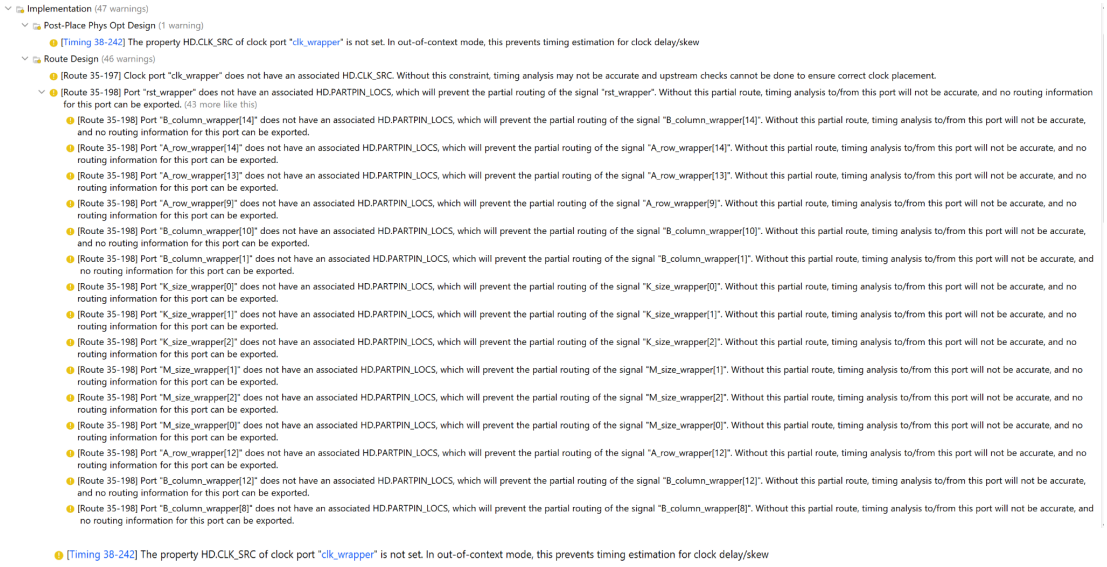
$$f_{max} = \frac{1}{t_{ck} - WNS} = 77.238\,MHz$$

E' stato quindi possibile passare ad esaminare i vari cammini critici. Si può notare come il cammino con il WNS più alto sia il Path 1, evidenziato nell'immagine sottostante.

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	3.053	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[1][7]/D	12.944	4.804	8.140	16.000	clk_125	clk_125
Path 2	3.073	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[0][9]/D	12.872	5.189	7.683	16.000	clk_125	clk_125
Path 3	3.087	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[2][7]/D	12.908	4.804	8.104	16.000	clk_125	clk_125
Path 4	3.165	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[0][7]/D	12.780	4.804	7.976	16.000	clk_125	clk_125
Path 5	3.185	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[3][7]/D	12.808	4.804	8.004	16.000	clk_125	clk_125
Path 6	3.374	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[2][9]/D	12.571	5.189	7.382	16.000	clk_125	clk_125
Path 7	3.378	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[3][9]/D	12.567	5.189	7.378	16.000	clk_125	clk_125
Path 8	3.383	17	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[1][9]/D	12.564	5.189	7.375	16.000	clk_125	clk_125
Path 9	3.405	16	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[1][5]/D	12.540	4.608	7.932	16.000	clk_125	clk_125
Path 10	3.442	16	176	u_matrix_multipli...r_state_reg[0]/C	u_matrix_multipl...ggio_reg[0][5]/D	12.551	4.608	7.943	16.000	clk_125	clk_125



L'implementazione ha portato alla luce alcuni warning, in quanto è stato implementato il tutto in modalità out of context. Di conseguenza non vengono associati i pin della ZYbo ai segnali di input e output.



Nel seguito è possibile visualizzare le risorse occupate da tutto il dispositivo, quindi quelle del wrapper (sia in assoluto, sia in termini %). Inoltre è riportato anche il report sul consumo di potenza.

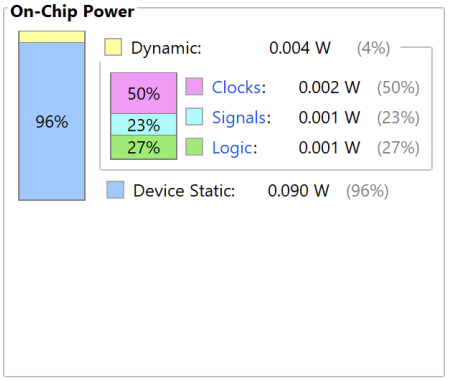
Resource	Utilization	Available	Utilization %
LUT	508	17600	2.89
FF	260	35200	0.74

Report sulla potenza

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

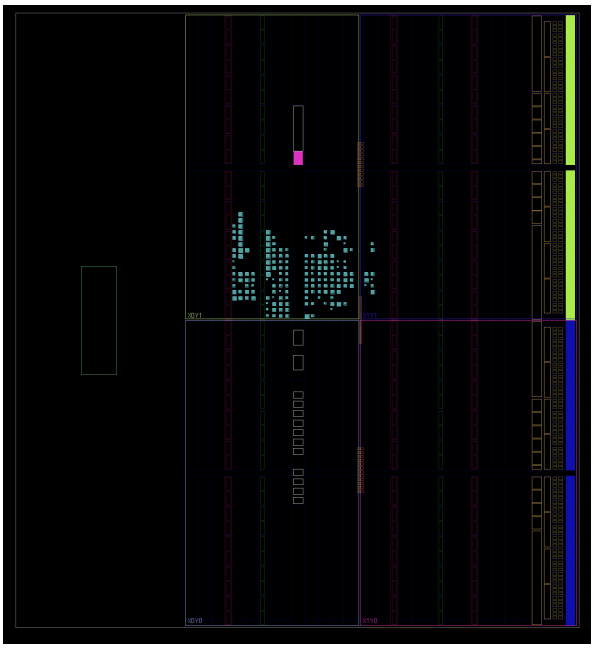
Total On-Chip Power:	**0.094 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,1°C**
Thermal Margin:	58,9°C (5,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



È riportata la potenza statica, la quale è quella di tutta l'FPGA che è accesa e che quindi consuma (riportata in azzurro). E' possibile notare come la potenza statica, rispetto a quella dinamica, sia molto maggiore in termini percentuali. Questo perchè il design progettato risulta piccolo rispetto all'intera scheda.
È riportata la potenza associata ai vari Clocks, ovvero tutta la potenza consumata da tutti gli elementi ad essi associati. È riportata anche la potenza dei segnali. Per Vivado, i segnali sono tutti gli attraversamenti delle matrici di interconnessione della fabric dell'FPGA (tutti i multiplexer che permettono a vivado di fare i vari collegamenti). La logica è invece costituita da LUT, Ripple Carry Adder e FF.

A implementazione finita è necessario controllare che ci sia una parte di area della FPGA in cui sono state allocate le risorse di interesse:



Conclusioni

L'implementazione del matrix multiplier in VHDL, e la sua successiva sintesi su Vivado, ha evidenziato come l'utilizzo delle risorse su FPGA sia strettamente proporzionale a due parametri chiave: MAX_SIZE e Nbit. Con l'aumentare di questi parametri, si ha un incremento significativo delle risorse impiegate, quali le unità logiche, la memoria e i registri. Questo comportamento era previsto, dato che l'aumento della dimensione delle matrici (MAX_SIZE) e del numero di bit necessari per rappresentare i dati (Nbit) impatta direttamente sull'architettura interna del moltiplicatore.

Inoltre, l'approccio adottato per gestire ingressi e uscite tutti riferiti al parametro MAX_SIZE ha introdotto una certa ridondanza nei calcoli e nella struttura finale del circuito. Sebbene ciò abbia garantito una maggiore flessibilità e scalabilità dell'architettura, ha tuttavia comportato un costo in termini di efficienza, soprattutto nelle risorse occupate su FPGA. Questo aspetto potrebbe essere ottimizzato in future implementazioni, riducendo il carico computazionale e l'overhead architetturale per migliorare l'efficienza complessiva.