Progetto finale reti logiche

Davide Di Marco - Matricola 912032

Anno Accademico 2020-2021

Indice

1 Introduzione

Per la stesura progetto si è deciso un'architettura semplice, funzionale, che rispetti le specifiche proposte in un tempo ragionevole, anzi considerevolmente sotto i tempi proposti, al netto delle opportune ottimizzazioni. L'obiettivo è stato ottenere un lavoro efficiente e facilmente modificabile.

1.1 Funzionamento in sintesi

Di seguito viene elencato sinteticamente il funzionamento del componente (verrà presentato successivamente anche tramite un'opportuna macchina a stati) attraverso un numero preciso di passi:

- Reset e attesa del segnale start.
- Inizializzazione dei registri disposti alla funzione di counting.
- Lettura del primo Byte e salvataggio all'interno del registro predisposto al numero di colonne.
- Lettura del secondo Byte e salvatggio all'interno del registro predisposto al numero di righe.
- Interruzione della lettura e inizio di un ciclo di somme del valore delle colonne con sé stesso, lungo quanto il valore contenuto nel registro delle righe.
- Con il valore preciso del prodotto Colonne-Righe, corrispondente al formato dell'immagine, si può continuare a passare in ingresso i Byte dal secondo in poi, decrementando il valore del prodotto appena ottenuto, così da sapere esattamente quanti Byte bisogna leggere.
- Durante lo scorrimento dei Byte in ingresso, questi vengono confrontati con dei registri inizializati nel primo step (Registro Min a 255 e Max a 0), al termine di questo confronto si otterrà in Min e Max, rispettivamente il valore minimo e massimo dei pixel passati in lettura.
- Riporto al secondo Byte (Primo pixel) il registro contenente gli indirizzi, così da procedere all'equalizzazione dell'immagine in questo modo:

$$delta\_value = Max\_value - Min\_value$$
$$Shift\_level = 8 - \lfloor log_2(delta\_value + 1) \rfloor$$

- Calcolato il valore di shifting relativo all'immagine in questione, si procede nuovamente a leggere tutti pixel e ad applicare, su ciascuno di essi, lo shifting.

$$temp\_pixel = (current\_pixel - Min\_value) << shift\_level$$
$$new\_pixel\_value = min(255, temp\_pixel)$$

- Applicato lo shifting, si procede alla scrittura nell'apposito indirizzo. La posizione in cui effettuo tale scrittura deriva dalla somma tra l'address attuale e il valore del prodotto Colonne-Righe.
- Letti ed equalizzati tutti i pixel, il segnale done viene portato alto, indice di fine equalizzazione e, una volta che il segnale start torna basso, ci si ritrova nella fase iniziale, pronti a terminare o ricevere un nuovo segnale start con relativa immagine.

1.2 Obiettivi Personali

Letta la specifica e abbozzata una prima soluzione, si è andati alla ricerca delle ottimizzazioni possibili. L'obiettivo imposto fu quello di ottimizzare il tempo con cui il sistema dovesse equalizzare l'intera immagine. Naturalmente ciò implicava un maggiore utilizzo di area occupata, si è scelto quindi di optare per una soluzione mediata, che richiedesse un numero di registri adeguato e che permettesse anche una facile comprensibilità e manutenzione del componente stesso. I tempi e l'area occopuata sono riportarti nel Paragrafo 3: Sintesi.

1.3 Note aggiuntive

Si è scelto di non utilizzare il prodotto per evitare operatori onerosi, dal momento che il prodotto poteva allo stesso modo essere gestito tramite due semplici operatori: somma e sottrazione, con relativi registri e multiplexer. Sorvolando sul costo, non è univocamente determinabile in quanto tempo tale operazione viene eseguita. Posso ottenere con precisione tale valore usando appunto, gli operatori di somma e sottrazione nel modo specificato nella sezione 1.1. Lo stesso ragionamento è stato fatto con lo shifting, all'inizio si era optato per un ciclo while, in modo da poterlo eseguire senza preoccuparmi di quale valore dovessi shiftare. Successivamente si è scelto di utilizzare uno shifting singolo eseguito per tante volte quante il valore risultante dalla funzione di shifting.

2 Architettura

2.1 Macchina a stati finiti

Con la macchina a stati finiti, si vuole rappresentare il comportamento del componente. Qui si è scelto di usare un modello a macrostati per non entrare troppo nel dettaglio dei singoli stati componenti. Il comportamento complessivo è espresso nel paragrafo 1.1. Di seguito si esprime con una rappresentazione schematica e una breve descrizione, gli stati e i segnali utilizzati. Le eventuali ottimizzazioni dovute a casi speciali sono indicate nel paragrafo successivo.

Figura 1: Implementazione della FSM

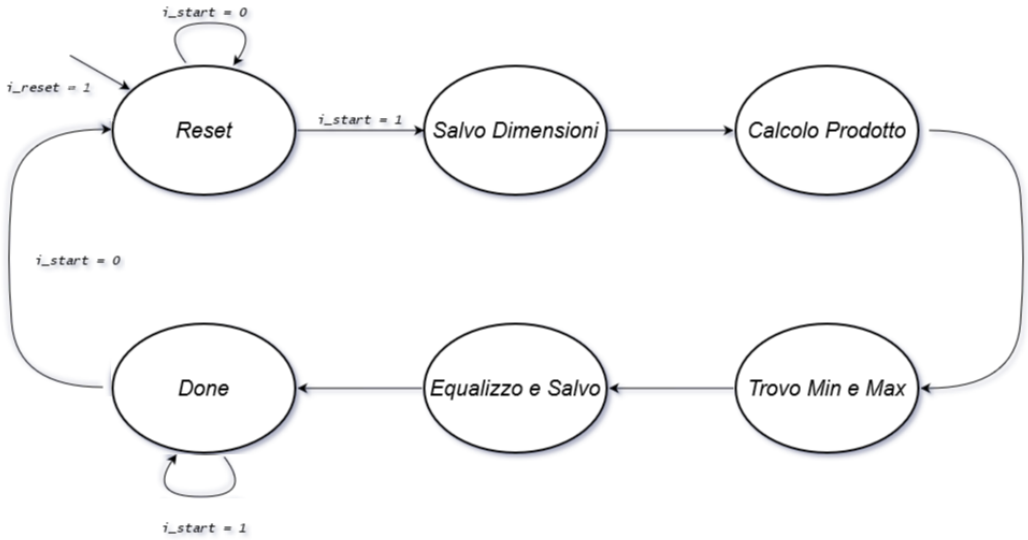


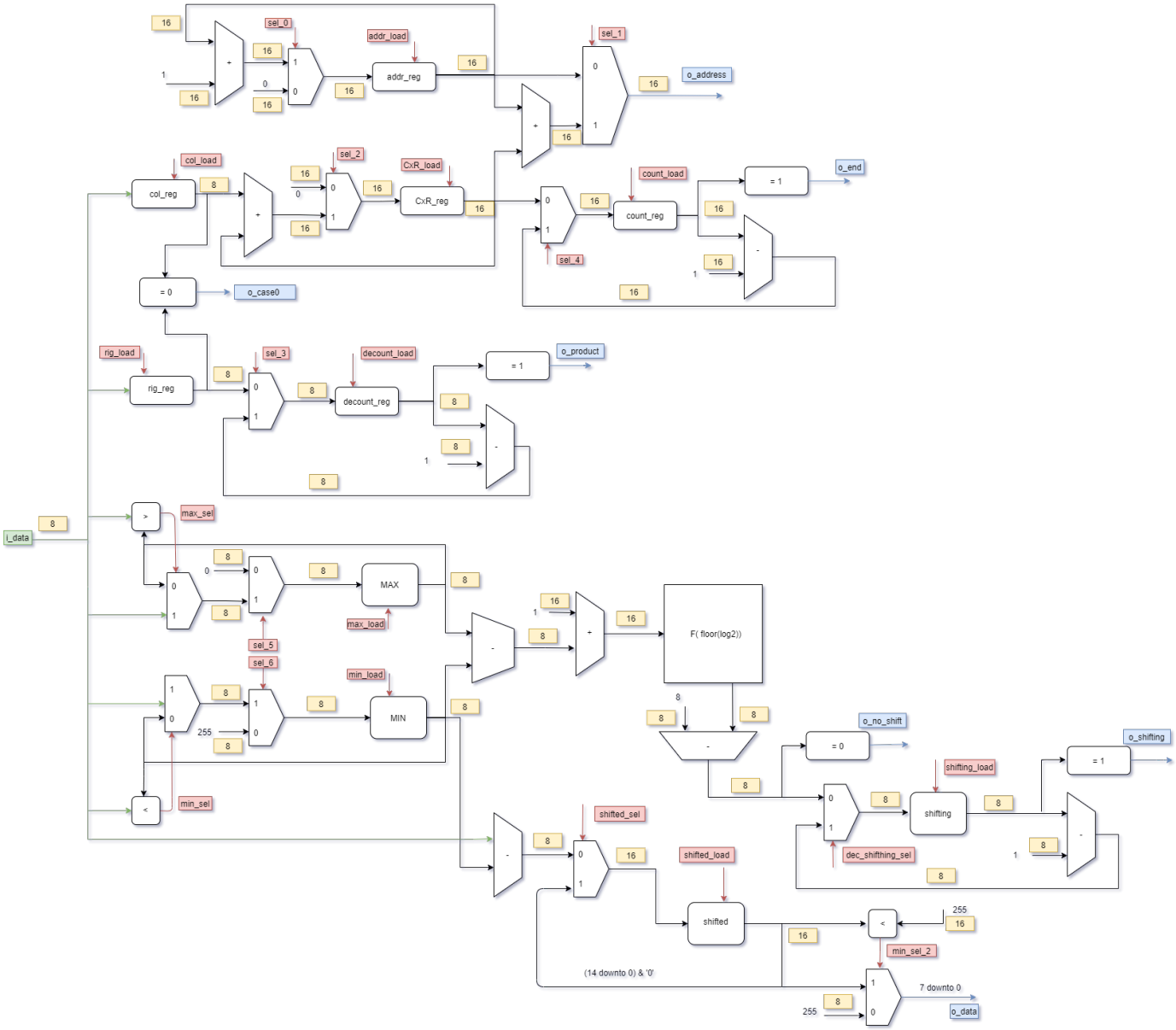
Tabella 1: Stati della FSM

Reset	Stato iniziale e finale della FSM. Si resta in questo stato fino a che il segnale di start non andrà alto .
Dimensioni	In questo stato vengono salvati il Byte 0 contenente il numero di Colonne e il Byte 1 contenente il numero di Righe.
Prodotto	In un registro inizializzato a 0, verrà sommato tante volte il valore del registro Colonne, quante il contenuto del registro Righe. L'uscita da questo ciclo avverrà quando il valore del registro che viene decrementato giungerà ad 1, tramite un opportuno segnale di output: *o_product*.
Min e Max	Vengono passati in ingresso tutti i valori dal Byte 2 al (Byte 2 + Prodotto) e ciascuno viene confrontato con il valore attuale nei registri Min e Max. Al termine della scansione si avranno, nei rispettivi registri, il minimo e massimo valore dei pixel letti.
Equalizzo e salvo	In questa fase, si ricomincia a leggere dal Byte 2, e per ciascun pixel letto, viene fatto uno shifting, calcolato come definito precedentemente, sulla differenza tra il pixel attuale e il Minimo pixel registrato. Il salvataggio avviene nell'indirizzo dato dalla somma tra il pixel attuale e il prodotto Colonna-Riga.
Done	Stato di fine equalizzazione, il segnale o_done andrà altro e si aspetterà che start si abbassi per tornare in Reset ed aspettare un'altra immagine o terminare.

2.2 Casi Limite

Per un corretto funzionamento del componente si è dovuto tener conto anche di alcuni "casi limite". Immagini che possiedono numeri precisi di pixel, che comporterebbero calcoli e transizioni inutili al componente. Casi come immagini da 0 pixel, che posseggono il valore Zero nel registro delle Colonne o delle Righe, viene dunque posto un segnale d'eccezione, o_case0, alto che porta direttamente nello stato Done, pronti a terminare o aspettare una nuova immagine. Immagini che possiedono un valore di shifting pari a Zero, derivanti da immagini aventi pixel massimo a 255 e minimo a 0, consentono di evitare il passaggio di shifting e procedendo direttamente al salvataggio dell'immagine.

2.3 Datapath



3 Sintesi

3.1 Timing Report

Tramite Report di Timing, si può analizzare la velocità della computazione del componente, rispetto un constrain di clock fornito. Nonostante la specifica richiedesse un constrain massimo di 100ns, nel report fornito se ne è usato uno da 10ns e si è comunque riusciti a stare di molto sotto esso.

Tabella 2: Timing Report Post Sintesi

Slack	Data Path Delay	Requirement
5.368ns	4.481ns	10.00ns

Tabella 3: Dettaglio del ritardo Data Path

Ritardo	logic	logic %	route	route %
4.481ns	2.132ns	47.579	2.349	52.421

Il valore fornito esprime quanto siamo vicini/lontani dal ciclo di clock provvisto dal constrain. Il valore comunicato dal Data Path delay, sul tale circuito e con la FPGA selezionata, ci determina l'intevallo tra l'arrivo del fronte di salita del clock e la commutazione di tutte le porte logiche del nostro componente. Nel restante tempo, identificato nella tabella come SLACK, tutti i segnali sono in attesa. Da tali dati possiamo ricavare anche la frequenza massima di funzionamento del componente, che sarà: $f_{max} = 1/(T_{ck} - WNS) \approx 215,889MHz$. Dove il tempo di clock posto è di 10ns e il WNS (Worst Negative Slack) pari a 5.368ns, dato ricavato dal report_timing.

3.2 Area utilizzata

Attraverso un "Report Utilization", viene fornita l'area occupata del design sintetizzato, i dati sono riportati come segue:

Tabella 4: Report d'utilizzo per l'area occupata

Risorsa	Utilizzo	Disponibile	Utilizzo in %
Look Up Table	174	134600	0.13
Flip Flop	130	269200	0.05

3.3 Post Sintesi

Sono stati risolti tutti i warning generati dal tool di sintesi. Nel post-sintesi è stato risolto anche un warning dato da un inferred latch, causato dalla non totale copertura della cassistica dovuta alla funzione che calcola lo shifting. Nel componente fornito, non sono presenti né latch nella Report Utilization, né warning nel Project Recap.

4 Simulazioni

Completata la trasposizione del Data Path e della FSM nel tool Vivado, si è cominciato a testare il comportamento del componente sintetizzato, tramite appositi Test Bench, forniti e creati. Lo scopo era quello di controllare il corretto funzionamento di tutte le funzionalità del componente attraverso diversi casi.

4.1 Test Bench 0 (fornito)

Il Test Bench fornito consiste in un'immagine molto semplice, la cui equalizzazione deve dare un risultato quanto segue:

Indirizzo Letura	Contenuto	Valore	Equalizzazione	Indirizzo Scrittura
0	#colonne	2	-	-
1	#righe	2	-	-
2	Pixel 1	46	0	6
3	Pixel 2	131	255	7
4	Pixel 3	62	64	8
5	Pixel 4	89	172	9

4.1.1 Elaborazione

Figura 2: TestBench 0, waveform dei segnali in post-synthesis simulation

Prenderò come esempio un Test Bench fornito dal prof. William Fornaciari per procedere a una dettagliata analisi del comportamento del componente, descrivendo passo per passo ciò che accade in ciascuno stato. Con la Figura 3 in sovraimpressione andiamo ad esaminare cosa accade durante la computazione:

- Qui ci troviamo nello stato di Default S0, stato in cui torneremo se, durante la computazione, ricevessi un segnale di Reset. Restiamo in questo stato fino a che non salirà ad 1 il valore di start.
- Nello stato S1 procedo ad inizializzare i valori dei registri contenenti l'inidirizzo attuale, il valore massimo e il valore minimo. Tramite opportuni Multiplexer si è settato i primi due valori a 0 e il terzo a 255.

- Nello stato S2, mi preparo a leggere da o_address il valore fornitogli dal registro address ponendo o_en ad 1, che contiene ancora 0. Nello stesso ciclo di clock pongo anche il valore del multiplexer sel_0 ad 1 così da incrementare di 1 in ogni ciclo di clock il valore contenuto nel registro address. Pongo anche ad 1 il load del registro adibito allo storing del prodotto, così da inizializzarlo a 0.

- Qui, salito nuovamente il fronte di clock, avrò in i_data il Byte 0, che salverò nel registro relativo delle colonne ponendo il suo segnale di load ad 1. Nel frattempo continuo a incrementare il registro address e a fornire il valore in lettura tramite o_en ad 1. Effettuerò un controllo su tale valore e, se questo fosse pari a 0, un segnale farà terminare direttamente il processo, passando allo stato finale.

- Si arresta la lettura di address, dato che si è arrivati al Byte 2, contenente il primo pixel da equalizzare. In i_data si avrà il Byte 1, che porta con sé il numero di Righe dell'immagine, il quale verrà salvato nel registro adibito, ponendo rig_load ad 1. Lo stesso controllo dello stato precedente sul valore appena immagazinato viene fatto sulle righe.

- Questo stato viene utilizzato per immagazinare il valore del registro contenente il numero di righe in un registro ausiliario, adibito al "conto alla rovescia".

- Da questo stato comincerà un ciclo. Mentre il registro descritto nel punto precedente sottrarrà a sé stesso 1 per ciascuna iterazione eseguita, il registro adibito al prodotto sommerà a sè stesso il valore del registro Colonne. Uscendo da questo stato e entrando nel successivo, si avrà nel registro CxR, il prodotto Colonne-Righe. La condizione d'uscita da tale stato si avrà quando un segnale o_product sarà posto ad 1. Ciò avverrà quando il valore del registro decount, adibito al conteggio, sarà pari ad 1.

- In questo stato "di transizione" pongo il segnale di enable ad 1, così da portare il Byte 2 del registro address su i_data nel prossimo ciclo. Viene posto il load di un registro Count ad 1, così da immagazinare il valore del prodotto appena calcolato e procedere nello stato in cui comincerò a confrontare il valore dei pixel per cercare minimo e massimo. Il registro Count ha la medesima funzione del registro Decount precedentemente specificato, sarà utile per contare il numero di pixel da leggere.

- Lo stato 8 è un semplice stato di preparazione al confronto che avverrà sotto forma di ciclo dallo stato successivo.

- Qui, avendo in lettura già il primo pixel, si inizia il confronto con i valori inizializzati in Max e Min, mediante due comparator e due multiplexer. Il registro address incrementerà di 1 il proprio valore fino ad arrivare a 2 + prodotto Colonne-Righe, conteggio che verrà effettuato dall'apposito registro count e il suo multiplexer sel_4. L'uscita da questo ciclo è sancita dal segnale di output o_end.

- Con la lettura dell'ultimo pixel e confrontato con Max e Min, il load di address viene posto ad 1 così che, nel prossimo ciclo, tale registro sarà re-inizializzato a 0. Ciò viene fatto perché si necessita di una seconda passata per equalizzare ciascun bit.

- Il valore di address continua da incermentare e il registro Count viene re-inizializzato allo stesso del prodotto, semplicementre ponendo nuovamente il valore di count_load ad 1.

- Lo stato S12 penserà all'ultimo incremento, tale da ottenere nuovamente il primo pixel in o_address.

- Questo stato permetterà di leggere il valore del registro in i_data tramite il segnale di enable. Posto il segnale load di un registro adibito al conteggio dello shifting ad 1, permette al componente di salvare quel preciso valore. Il valore appena salvato, deriva da una serie di operazioni: per prima cosa viene calcolata la differenza tra minimo e massimo valore del pixel, a tale differenza viene sommato 1, calcolata la parte intera inferiore del logaritmo in base due della somma e, infine, tale valore viene sottratto ad 8. Se il risultato equivale a 0, viene creata un'eccezione chiamata o_no_shift che comunica al componente che l'immagine dovrà essere salvata così come viene portata in ingresso, ovvero saltando il ciclo dello stato S15.

- Qui, si inizializza il valore del registro adibito allo shifting, chiamato Shifted. Il suo valore viene dettato da un multiplexer cui, se fornito il segnale 0, porterà nel registro il valore della differenza tra il pixel corrente e il minimo valore registrato. Altrimenti, se il segnale fosse 1, torna in ingresso lo stesso valore di Shifted traslato a sinistra di un posto.

- Ci troviamo in S15, qui avviene lo shifting descritto nello stato precedente. Il multiplexer pilotato da shifted_sel viene posto ad 1 e lo shifting verrà eseguito tante volte quante il valore contenuto dal registro shifting.

- Usciti dal ciclo, sono pronto a scrivere il valore ottenuto nel relativo indirizzo, portato in o_address dal valore del multiplexer 1 che, avendo ora come segnale di selezione 1, fornisce in uscita la somma del valore di address attuale e il prodotto colonne-righe. Avendo finito il ciclo di equalizzazione del primo pixel si procede a decrementare il registro Count, settando il segnale del suo multiplexer ad 1. La permanenza in questo ciclo verrà dettata da un segnale o_end, lo stesso che usammo per contare il numero di pixel nell'immagine.

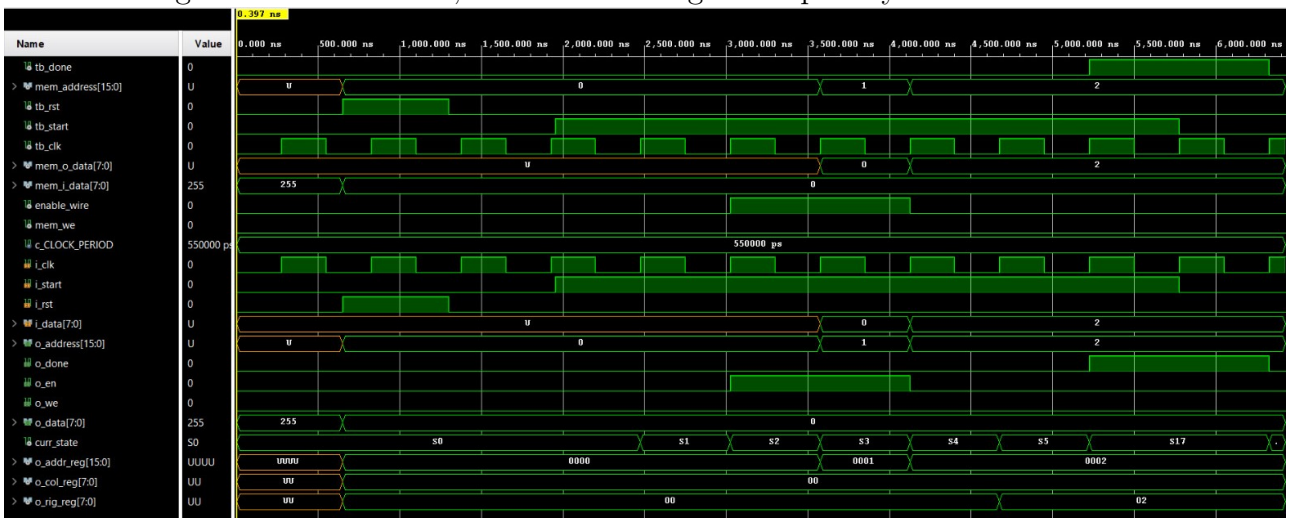
- Lo stato S17 porrà il segnale di output done ad 1 per segnalare la fine dell'equalizzazione. Il segnale di start portato basso, comunica al sistema la fine della procedura e il ritorno nello stato iniziale S0.

4.2 Test Bench 1, 0 Pixel

Nel Test 1 vengono forniti 3 valori: il primo di questi è proprio il valore 0. Nonostante vengano forniti altri valori, si noti come dopo lo stato S5, l' immediato successivo è S17, stato di fine computazione. Come specificato nel funzionamento prima, il passaggio avviene tramite il segnale o_col_case0 nel caso di colonna o o_rig_case0 nel caso di riga.

4.2.1 Elaborazione

Figura 3: TestBench 1, waveform dei segnali in post-synthesis simulation

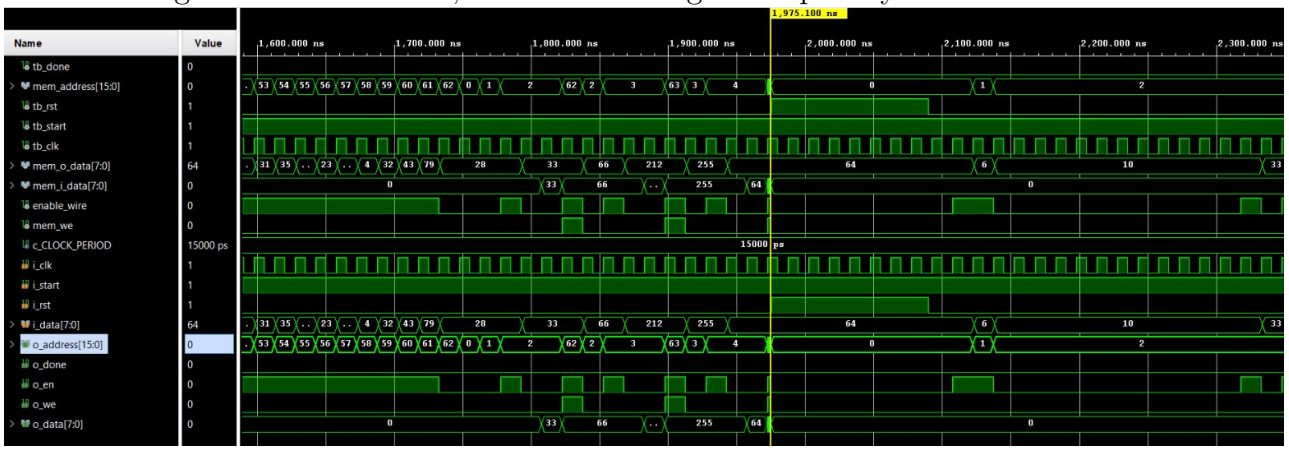


4.3 Test Bench 2, Reset

In questo secondo Test, ingrandendo su una precisa parte della waveform, notiamo che il segnale di reset viene portato alto e, con esso, mem_data e mem_address tornano ad avere valore 0, proprio come ci si aspetterebbe dalla ricezione di un segnale reset. Tornati nello stato iniziale S0, si è pronti a terminare o ricevere un' altra immagine. Si ricorda inoltre che il segnale reset è inserito anche nella sensitivity list di ciascun registro del componente, quindi anche ogni valore contenuto da ciascun registro torna a 0.

4.3.1 Elaborazione

Figura 4: TestBench 2, waveform dei segnali in post-synthesis simulation

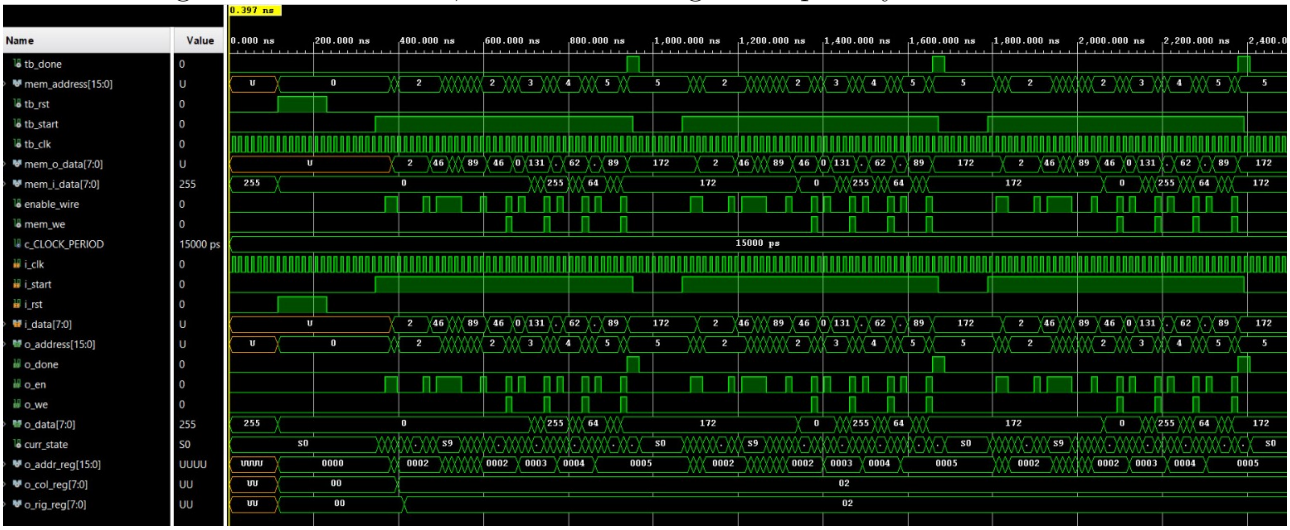


4.4 Test Bench 3, 3 Immagini

Nel terzo ed ultimo Test, vengono proposte 3 immagini identiche, con quindi 3 segnali di start. Si nota infatti che in corrispondenza di quando questi scendono a zero, solo quando il segnale done viene portato alto, lo stato della FSM corrisponde a S0, iniziale. La macchina aspetta di ricevere un altro segnale di start prima di ripartire per la successiva computazione. Ovviamente essendo le tre immagini identiche, i valori e i segnali della waveform sono perfettamente uguali.

4.4.1 Elaborazione

Figura 5: TestBench 3, waveform dei segnali in post-synthesis simulation



5 Conclusione

Il progetto presentato si è rilevato interessante dal punto di vista formativo. Ha fatto da ponte tra capacità organizzative e programmazione. Non solo si è dovuto creare un programma funzionante dal punto di vista applicativo con rispetto della specifica, ma innanzitutto progettare il relativo modello logico che ne facesse da spina dorsale. Partendo dal datapath si è progettata la relativa FSM per poi tradurre tutto tramite il tool Vivado. La qui presente relazione vuole fornire un componente:

- funzionante in pre e post-sintesi senza la presenza di warning o latch;
- che sia ragionevolmente ottimizzato, si è cercato di abbassare il tempo di computazione sotto un limite ragionevole, la Frequenza massima impostabile risultante è di 215Mhz;
- con un utilizzo di LUT pari al 0.13% e di Flip Flop allo 0.05%.