Prova finale reti logiche

Matteo Nunziante

Anno 2020-2021

Codice Persona : 10670132
Matricola : 913670

Indice

1 Introduzione

1.1 Descrizione progetto

Lo scopo del progetto è quello di realizzare un'implementazione semplificata del metodo di equalizzazione dell'istogramma di un immagine, ovvero un metodo di elaborazione digitale con cui si può calibrare il contrasto dell'immagine.Si cercherà quindi di distribuire meglio le misure di intensità dei pixel sull'intero intervallo di valori possibili al fine di aumentare il contrasto della figura. Nel particolare l'algoritmo di equalizzazione sarà applicato solo ad immagini in scala di grigi a 256 livelli e trasforma ogni pixel nel seguente modo:

$$DeltaValue = MaxPixelValue – MinPixelValue \quad (1)$$

$$ShiftLevel = (8 – Floor(Log_2(DeltaValue + 1))) \quad (2)$$

$$TempPixel = (CurrentPixelValue - MinPixelValue) << ShiftLevel \quad (3)$$

$$NewPixelValue = Min(255, TempPixel) \quad (4)$$

Dove MaxPixelValue e MinPixelValue sono rispettivamente il valore del pixel maggiore e minore dell'immagine che si sta analizzando, CurrentPixelValue è il valore corrente del pixel da convertire e NewPixelValue è il risultato della conversione.

(1) calcola la differenza tra il pixel maggiore e minore;
(2) è un numero compreso tra 0 e 8, in base al valore del delta;
(3) sottrae al pixel corrente il pixel minimo e il risultato viene shiftato a sinistra;
(4) è il nuovo valore del pixel corrente, ovvero il minimo tra 255,massimo valore disponibile, e TempPixel.

1.1.1 Limite algoritmo

Essendo un algoritmo semplificato non sempre sarà possibile ottenere una buona conversione, infatti basta tenere in considerazione il caso in cui l'immagine fornita abbia un valore DeltaValue massimo, cioè 255.In questo caso si avrà uno ShiftLevel pari a zero, ovvero l'immagine processata sarà identica a quella originale, essendo MinPixelValue uguale a zero.

1.2 Struttura dati

L'immagine da elaborare è memorizzata in una memoria RAM con indirizzamento al byte, in particolare nel byte 0 e nel byte 1 si trovano rispettivamente numero colonne e numero righe dell'immagine da convertire, nei byte successivi sono memorizzati sequenzialmente i pixel dell'immagine.Dimensioni e valori dei pixel hanno dimensione pari a 8 bit.

Il modulo da realizzare legge l'immagine dalla memoria sopra descritta, la elabora e scrive l'immagine equalizzata immediatamente dopo l'immagine originale, ovvero il pixel N-esimo dell'immagine originale verrà scritto in posizione

$$2 + (NumeroRighe \times NumeroColonne) + N \tag{5}$$

1.3 Interfaccia componente

Il componente da descrivere ha la seguente interfaccia:

```
entity project_reti_logiche is
    port (
        i_clk       : in std_logic;
        i_rst       : in std_logic;
        i_start     : in std_logic;
        i_data      : in std_logic_vector(7 downto 0);
        o_address   : out std_logic_vector(15 downto 0);
        o_done      : out std_logic;
        o_en        : out std_logic;
        o_we        : out std_logic;
        o_data      : out std_logic_vector(7 downto 0)
    );
end project_reti_logiche;
```

1.3.1 Descrizione segnali

- i_clock è il segnale di CLOCK in ingresso generato dal TestBench;
- i_rst è il segnale di RESET(attivo alto) che inizializza la macchina pronta per ricevere il primo segnale di start;
- i_start è il segnale di START generato dal TestBench;
- i_data è un segnale che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_address è il segnale di uscita che manda l'indirizzo alla memoria;
- o_done è il segnale in uscita che comunica la fine dell'elaborazione e della scrittura in memoria del risultato;
- o_en è il segnale di ENABLE da mandare alla memoria per poter comunicare;
- o_we è il segnale di WRITE ENABLE da dover mandare alla memoria per leggere(=0) o per scrivere(=1), in coppia con o_en;
- o_data è il segnale di uscita dal componente verso la memoria.

1.3.2 Utilizzo segnali

Una volta ricevuto il primo segnale di reset, il modulo inizierà la computazione quando i_start verrà portato a 1 e questo segnale rimarrà tale fino a che il segnale o_done non verrà portato alto, ovvero fino alla fine dell'elaborazione dell'immagine.
Il segnale o_done deve rimanere pari a 1 fino a che i_start non è riportato a 0 e, successivamente, i_start non può essere riportato alto fino a che non è stato abbassato o_done.Se a questo punto viene rialzato il segnale i_start il modulo dovrà ripartire con la nuova codifica.

1.4 Considerazioni di progetto

Il modulo è stato progettato considerando che:

- prima della prima codifica verrà sempre attivato il segnale i_rst mentre, per successive elaborazioni, non si dovrà attendere il reset del modulo;
- il segnale di RESET può essere attivato in qualsiasi istante, anche in modo asincrono rispetto al fronte di clock;
- l'immagine non verrà mai cambiata all'interno della stessa esecuzione, ossia fino a che non venga portato alto il segnale o_done.

1.5 Esempio pratico di funzionamento

Viene riportato di seguito un esempio del funzionamento utilizzando un semplice Test Bench composto da un'immagine di dimensione 2x2 e con valori dei pixel originali pari a 46,131,64 e 89.

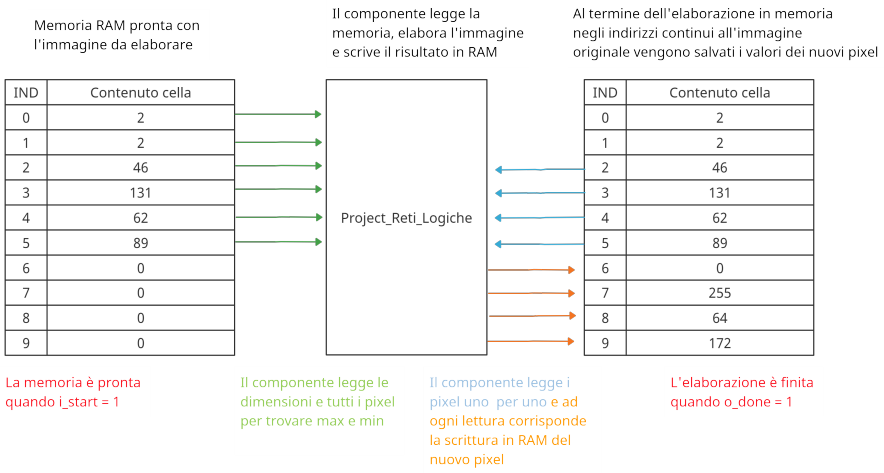

Figura 1: Funzionamento complessivo

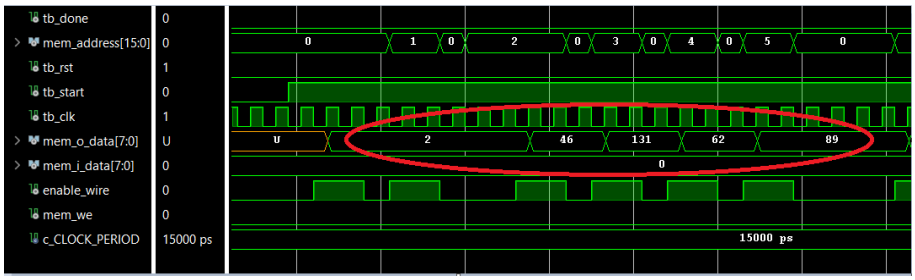

Figura 2: Lettura delle dimensioni e di tutti i pixel

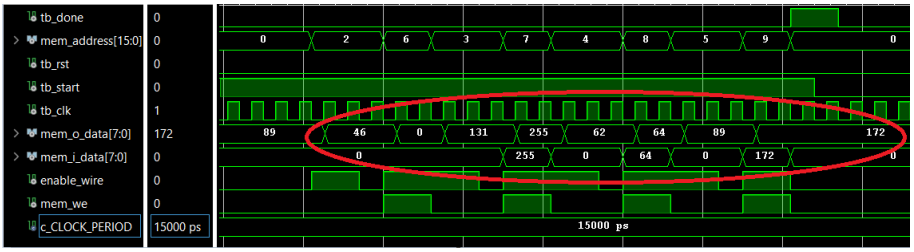

Figura 3: Scrittura dei nuovi pixel

2 Architettura

2.1 Descrizione generale

Per la realizzazione del componente si è deciso di utilizzare un approccio comportamentale(behavioral). L'architettura realizzata consiste in un unico modulo che realizza la macchina a stati finiti, escludendo il modulo della RAM che viene fornito dal TestBench.
Il modulo utilizza al suo interno due processi sincronizzati:"stato" e "combin". Il processo "stato" è sincronizzato sul fronte di salita del clock e si occupa di aggiornare lo stato corrente della macchina.Si è scelto di aggiornare lo stato sul fronte di salita in quanto anche la RAM aggiorna i propri segnali alla salita del clock, dunque si mantiene una coerenza in fase di aggiornamento.
Il processo "combin", sincronizzato sul fronte di discesa del clock, implementa la macchina a stati vera e propria, ovvero quello che la macchina deve fare in base al suo stato corrente.

2.1.1 Ipotesi

Essendo il periodo di clock pari a 100 ns si è ipotizzato che il tempo di transazione del valore dei segnali sia minore di metà periodo di clock. In questo modo il processo "combin" può leggere i valori corretti dei segnali e decidere il

prossimo stato.
In fase di sperimentazione questa ipotesi è stata verificata come corretta.

2.2 Macchina a stati

In figura 4 viene riportato il diagramma della macchina a stati.

2.2.1 Descrizione macchina

La macchina è composta dai seguenti stati:

- IDLE : stato iniziale della macchina nel quale vengono inizializzati tutti i segnali e le variabili utilizzate.La macchina rimane in questo stato fino a quando non riceve il primo segnale di reset(i_rst = 1), il quale fa passare la macchina nello stato di RESET.
- RESET : in questo stato la macchina aspetta il segnale di inizio codifica, ovvero aspetta che il segnale i_start venga messo uguale a 1. Questo vuol dire che l'immagine in memoria è pronta ad essere codificata e la macchina inizia l'elaborazione passando allo stato successivo.
- START_Indirizzo : la macchina si prepara a leggere dalla memoria il numero di colonne(byte 0) e il numero di righe(byte 1) settando o_en = 1, o_we = 0 e o_address pari all'indirizzo 0 nel caso di prima iterazione o pari all'indirizzo 1 nel caso di seconda interazione. Se la macchina non ha ancora letto entrambe le dimensioni allora lo stato successivo sarà START_Wait, altrimenti si prosegue con START_Fine.
- START_Wait : questo stato serve alla macchina per aspettare un ciclo di clock necessario alla memoria RAM per settare correttamente il segnale i_data con l'informazione presente al byte richiesto dal componente.
- START_Read : in questo stato viene effettivamente letta la risposta della RAM alla richiesta e viene memorizzata come numero colonne o numero di righe a seconda dell'indirizzo richiesto.Da questo stato si torna poi a START_Indirizzo.
- START_Fine : qui la macchina salva il numero totale di bit come prodotto dei dati precedentemente letti e si prepara per leggere tutti i pixel dell'immagine per trovare il massimo e minimo valore presente.
- CALCOLA_Indirizzo : fino a quando la macchina non ha letto i valori di tutti i pixel setta o_address con il prossimo indirizzo da leggere e poi passa allo stato CALCOLA_Wait.Se invece ha già letto tutti i pixel passa a CALCOLA_Fine.

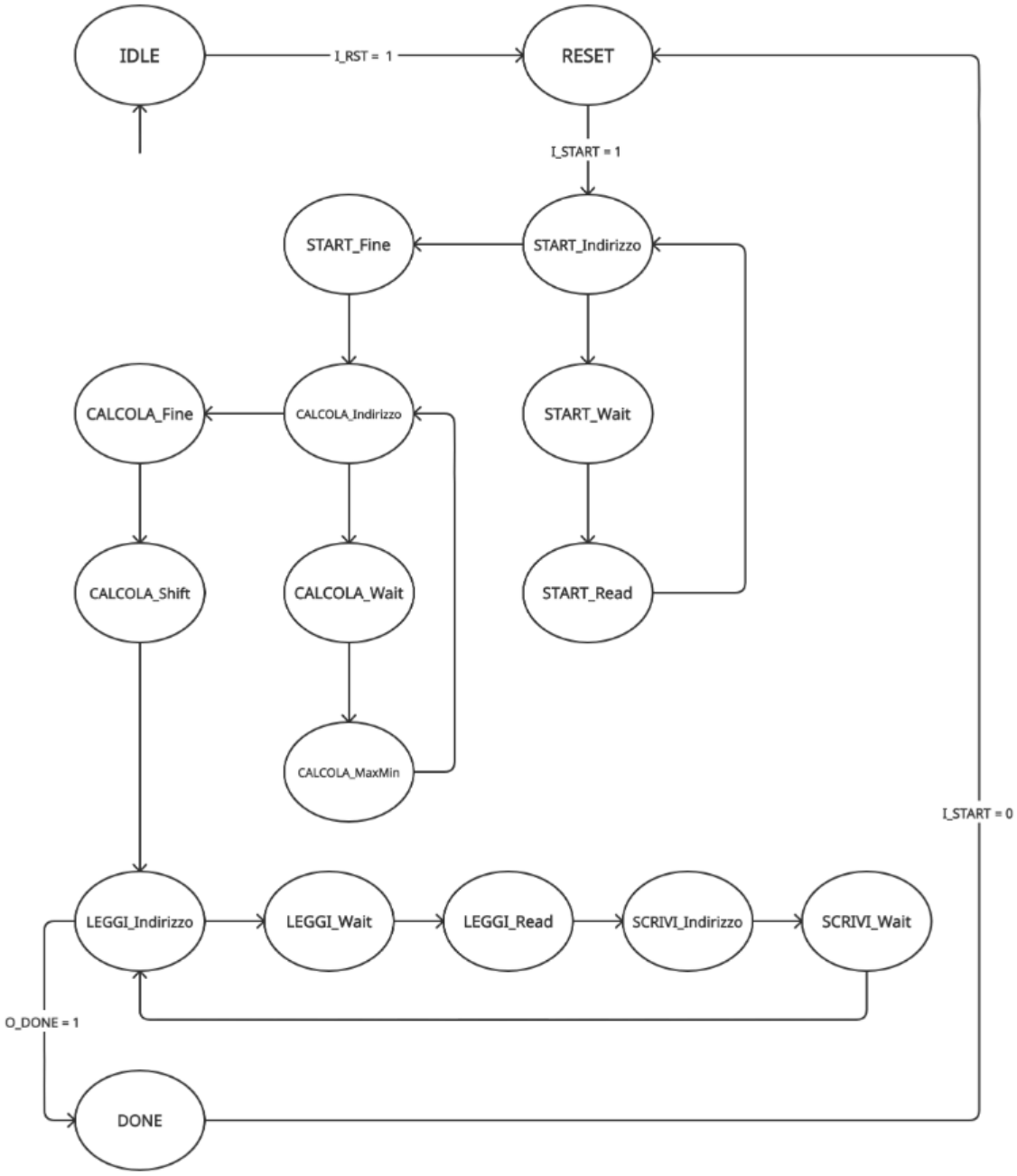

Figura 4: Diagramma degli stati del componente

- CALCOLA_Wait : analogo a START_Wait con la sola differenza nello stato prossimo che sarà CALCOLA_MaxMin.
- CALCOLA_MaxMin : la macchina legge il valore richiesto e lo confronta con il massimo e minimo trovati fino a quel momento,aggiornandoli se necessario.
- CALCOLA_Fine : è finita la lettura e valutazione di tutti i pixel e dal prossimo stato inizia l'algoritmo di elaborazione dell'immagine vero e proprio.
- CALCOLA_Shift : si calcola il delta e, tramite un semplice controllo a soglia, si determina il valore dello shift, quindi si realizzano le formule (1) e (2).
- LEGGI_Indirizzo : la macchina deve leggere di nuovo dalla RAM tutti i pixel per convertirli uno ad uno e, fino a quando non li ha elaborati tutti, setta il segnale o_address all'indirizzo successivo e passa allo stato LEGGI_Wait.Se invece li ha già convertiti tutti vuol dire che la computazione dell'immagine è terminata, dunque pone ad un valore alto il segnale o_done e passa allo stato DONE.
- LEGGI_Wait : si aspetta il ciclo di clock necessario alla RAM come nei casi precedenti.
- LEGGI_Read : si legge il valore effettivo del pixel richiesto e lo si converte applicando l'algoritmo descritto nell'introduzione per il calcolo del nuovo pixel da inserire, in particolare formule (3) e (4).
- SCRIVI_Indirizzo : si prepara la memoria per scrivere, quindi o_en = 1 e o_we = 1, inserendo in o_address l'indirizzo nel quale scrivere il pixel N-esimo, come descritto nella formula (5).
- SCRIVI_Wait : si aspetta un ciclo di clock per attendere che il nuovo valore venga correttamente memorizzato in memoria;dopo questo stato si torna a LEGGI_Indirizzo.
- DONE : in questo stato la macchina ha terminato l'elaborazione dell'immagine e aspetta che il segnale i_start venga abbassato.Quando questo accadrà passa allo stato successivo, ovvero torna allo stato RESET in quanto deve essere pronta per una nuova esecuzione appena il segnale i_start diventerà di nuovo alto, senza aspettare il RESET.

2.2.2 Note generali sul funzionamento della macchina

Alcuni dettagli generali:

- Il segnale di fine computazione o_done è sempre pari a 0 tranne quando la macchina ha finito l'elaborazione, in particolare viene posto a 1 dallo stato LEGGI_Indirizzo e poi verrà mantenuto tale dallo stato DONE, per poi essere abbassato nuovamente quando i_start viene posto a 0, cioè quando torna allo stato RESET.

- La macchina a stati utilizzata ovviamente non è minima infatti, effettuando delle ottimizzazioni, sarebbe possibile aggregare più stati, per esempio tutti gli stati che terminano con X_Wait. Essendo il numero di FF e di LUT utilizzato molto minore di quelli disponibili complessivamente ed essendo ampiamente rispettato il vincolo sul tempo di clock di 100 ns, si è deciso di utilizzare la macchina non ottimizzata in favore di una maggior leggibilità del codice e stati più semplici.

- Il reset è asincrono, ovvero in qualsiasi momento viene posto a 1 il segnale i_rst il processo "combin" si attiva e pone a 1 il segnale interno resetAttivato, che verrà utilizzato nel processo "stato" al prossimo fronte di salita per un aggiornamento sincrono dello stato. Per semplicità nella figura 4 non è stato riportato, ma l'aggiornamento dovuto al reset è indipendente dallo stato corrente della macchina, ovvero in qualunque stato essa sia al fronte di salita successivo lo stato corrente sarà RESET.Inoltre, per come è stata progettata la macchina, se a seguito del reset il segnale i_start viene lasciato pari ad 1 verrà iniziata una nuova elaborazione della stessa immagine.Se invece c'è la necessità di cambiare i dati in memoria allora occorre porre a 0 i_start quando il reset viene attivato.

3 Simulazioni

Per testare il componente sono stati utilizzati diversi Test Benches, ognuno rivolto a verificare il funzionamento nei casi limite o in condizioni generali e casuali.
Nel particolare sono state effettuate le seguenti prove:

- Immagine vuota, ovvero 0 pixel, per verificare che la macchina non scriva nulla in memoria. Come si può notare nell'immagine 5 riportata di seguito, dopo la lettura del numero di colonne e righe(0 e 2 rispettivamente), la macchina non fa nulla se non far susseguire gli stati fino allo stato DONE.

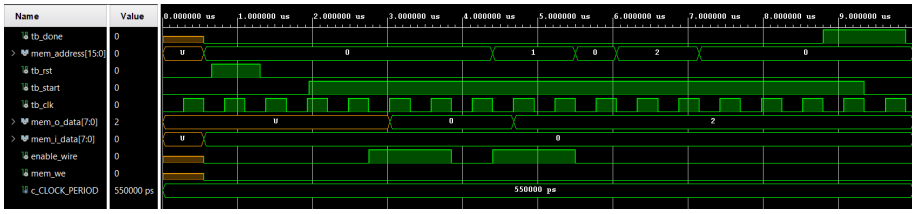

Figura 5: Test immagine vuota

- Immagine di dimensione massima, ovvero 128x128 pixel(caso limite opposto al precedente).

- Immagine con valori dei pixel tutti pari a 0.Caso delta uguale a zero, quindi shift massimo. Si verifica che la macchina riscrivi ancora zero come nuovi valori. Si riporta il comportamento osservato.

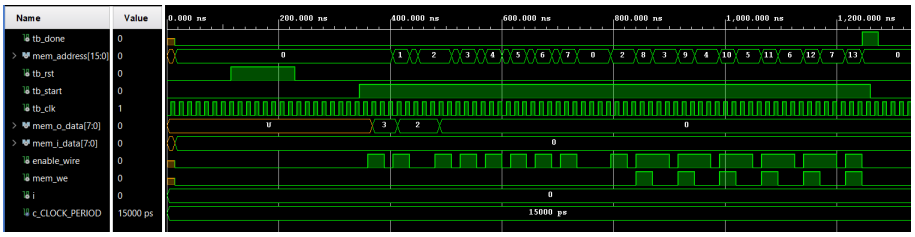

Figura 6: Test immagine tutti i pixel a zero

- Immagine con valori dei pixel tutti pari a 255. Caso duale al precedente di shift massimo.Si verifica che i valori 255 saranno effettivamente riportati a zero a seguito di uno shift pari a 8.

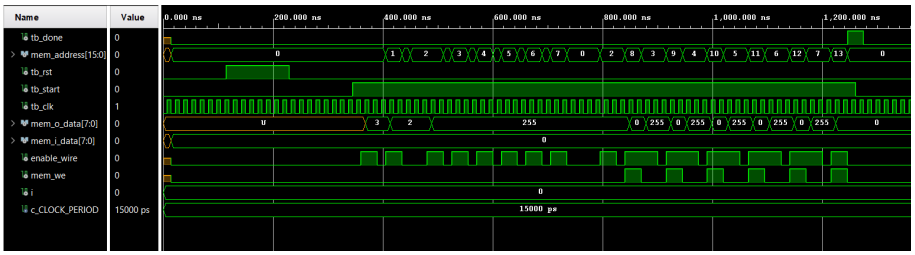

Figura 7: Test immagine con tutti i pixel di valore massimo

- Immagine con valori solo pari a 0 e 255 per verificare il caso di delta massimo, ovvero di shift nullo.In questo caso si nota come ogni valore letto venga riscritto uguale.

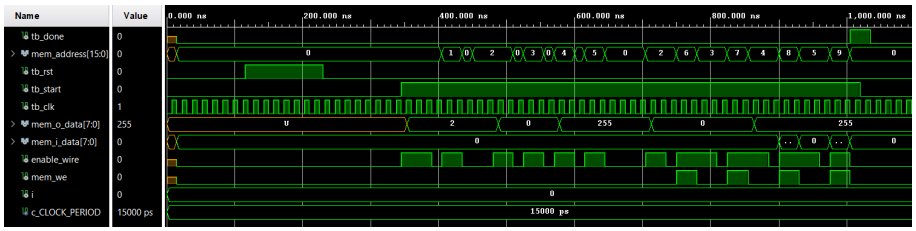

Figura 8: Immagine con delta massimo

- Test del reset asincrono. Si può osservare, grazie ai segnali o_en e o_we pari a zero entrambi, come, a seguito di i_rst = 1, la macchina torni allo stato di RESET, cominciando la nuova elaborazione quando il segnale i_start viene posto uguale a 1.Se il segnale i_start fosse lasciato uguale a 1 la macchina comincerebbe di nuovo la conversione.

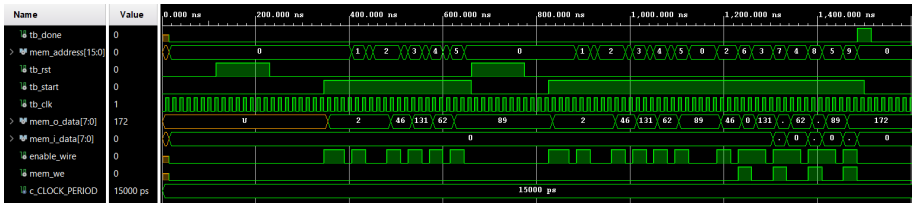

Figura 9: Verifica reset asincrono

- Test con più immagini, ovvero si verifica il comportamento della macchina nel caso di conversioni successive.Come si può osservare in figura 10 la macchina, dopo aver terminato l'elaborazione della prima immagine, aspetta che venga riportato alto il segnale i_start, potendo iniziare così la conversione della nuova immagine(nell'esempio la nuova immagine ha dimensioni 4x1).

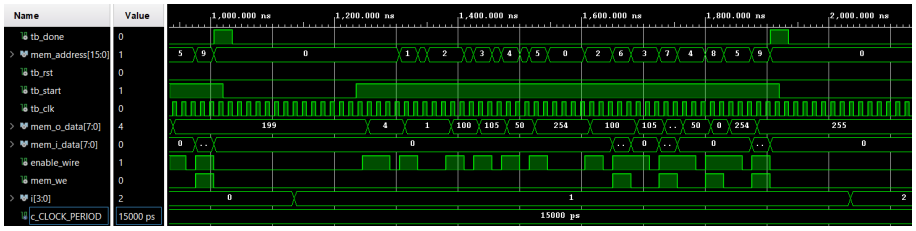

Figura 10: Test con più immagini

- Test generici creati da un generatore con immagini in numero e dimensioni variabili.

4 Risultati sperimentali

Per la realizzazione del progetto si è scelto come dispositivo FPGA la scheda xc7a200tfbg484-1 in quanto, ai fini del progetto, di dimensioni e risorse utilizzate molto limitate, non è stato necessario utilizzare schede più grandi e complesse.

4.1 Sintesi

Il report di sintesi non ha riportato errori o warning significativi. È presente infatti un solo warning relativo al segnale "resetAttivato" usato nel processo "stato" e non inserito nella sensitivity list, ma questa è stata una decisione progettuale, in quanto si vuole aggiornare lo stato della macchina solo al fronte di salita del clock e non al variare del segnale in questione.

4.1.1 Risorse utilizzate

Vengono riportate le risorse utilizzate in termini di numero di FF e LUT(figura 11) e percentuale complessiva(figura 12).

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!								329	114	0.0	0	3
✓ impl_1	constrs_1	route_design Complete!	44.281	0.000	0.163	0.000	0.000	0.132	0	317	114	0.0	0	3

Figura 11: FF e LUT utilizzate

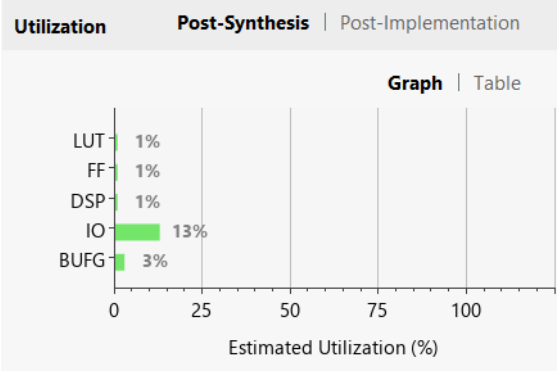

Figura 12: percentuale di risorse utilizzate

4.2 Timing report

L'unico vincolo richiesto nella progettazione è stato il funzionamento della macchina con un periodo di clock di almeno 100 ns, che viene ampiamente rispettato. Infatti, provando a ridurre il periodo di clock, il vincolo di timing non viene rispettato se il periodo è minore di 13ns.

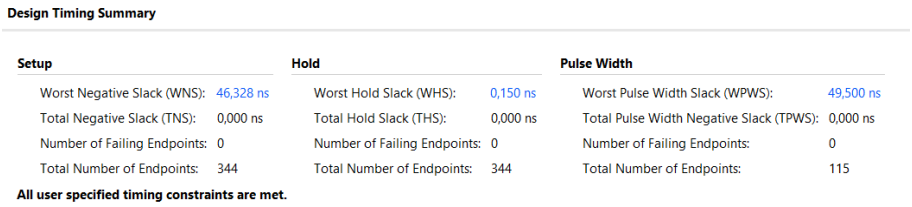

Figura 13: Timing result

5 Conclusione

L'architettura realizzata implementa correttamente la funzionalità voluta, infatti la verifica sul comportamento è stata effettuata sia con test casuali creati tramite un generatore, sia con test mirati a delle situazioni specifiche, e.g. i casi limite. Inoltre viene rispettato il vincolo sul periodo di clock della macchina. Si può dunque affermare che il componente realizzato funzioni correttamente, infatti supera tutti i test in Behavioral Simulation, in Post-Synthetis Functional e in Post-Synthetis Timing e, anche se non richiesto, supera correttamente anche Post-Implementation Functional e Timing.