PROJECT RETI LOGICHE

LORENZO PRADA 2021

10529212

Sommario

INTRODUZIONE

L'obiettivo della prova finale di reti logiche 2021 è quello di realizzare, utilizzando un'architettura RTL, un equalizzatore di immagini in scala di grigi a 256 livelli. Tali immagini avranno una dimensione compresa tra 0x0 e 128x128 pixel e saranno equalizzate con le equazioni riportate di seguito:

$$delta\_value = max\_pixel\_value - min\_pixel\_value$$

$$shift\_level = (8 - floor(\log_2(delta\_value + 1)))$$

$$temp\_pixel = (current\_pixel\_value - min\_pixel\_value) \ll shift\_level$$

$$new\_pixel\_value = min(255, temp\_pixel)$$

dove *max_pixel_value* e *min_pixel_value* sono il valore massimo e minimo dell'immagine da equalizzare, *current_pixel_value* è il valore del pixel da trasformare, e *new_pixel_value* è il valore del nuovo pixel da scrivere in memoria.

La memoria fornita è pre-allocata e presenta la struttura illustrata in figura 1:

- byte 0 contiene il numero di colonne (*C*) dell'immagine;
- byte 1 contiene il numero di righe (*R*) dell'immagine;
- byte da 2 a *R*C+1* contengono i valori dei singoli pixel;
- byte da *R*C+2* a *2R*C+1* sono inizializzati a 0 e conterranno il valore dell'immagine equalizzata.

L'interfaccia del componente da realizzare ha anch'esso un'interfaccia predefinita illustrata in figura 2.

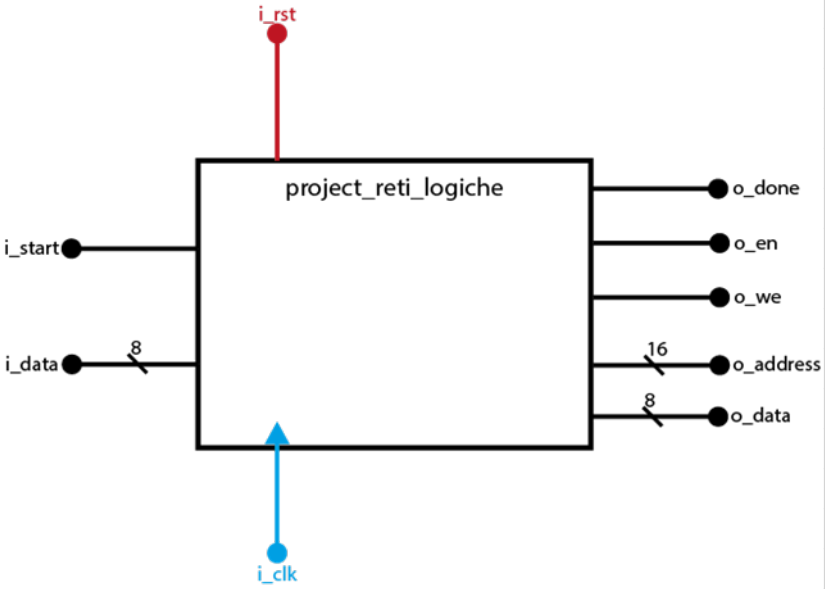


Figura 2: interfaccia del progetto

ADDRESS	SIZE	VALUE	NEW_VALUE
0	4		
1	3		
2		76	
3		131	
4		109	
5		89	
6		46	
7		121	
8		62	
9		59	
10		46	
11		77	
12		68	
13		94	
14			120
15			255
16			255
17			172
18			0
19			255
20			64
21			52
22			0
23			124
24			88
25			192

Figura 1: esempio di memoria con un'immagine 4x3

DESCRIZIONE MACROSCOPICA

In figura 3 è possibile vedere lo schema ad alto livello del componente realizzato. Esso è composto da otto moduli principali, un multiplexer che controlla l'uscita di *o_address*, e una porta logica per il controllo del segnale di *enable* del **counter_16_bit**.

Gli otto componenti principali sono elencati di seguito e saranno descritti nella sezione dedicata:

- activator;
- chunk_maker;
- initializer;
- counter_16_bit;
- address_requester;
- big_brain;
- max_min;
- shift_pixel.

Tutti i componenti sono stati descritti con una descrizione di tipo structural, tuttavia **initializer** è stato successivamente descritto anche con una descrizione di tipo FSM. Nel modulo principale, durante la dichiarazione della entity, è stato specificato di utilizzare la descrizione FSM per la sintesi.

Il processo di equalizzazione è stato separato in due processi. Il primo è quello di acquisizione dei primi due byte di memoria, necessari per calcolare le dimensioni dell'immagine (d'ora in poi CHUNK); il secondo processo invece si occupa di leggere la memoria, ricavare il valore massimo e minimo, e infine effettuare una seconda lettura dell'immagine alternando la lettura di ogni pixel alla sua scrittura equalizzata in memoria.

Tutti i componenti sono sequenziali e lavorano sul fronte di salita del clock, ad esclusione dello **shift_pixel**, che è un blocco di logica combinatoria.

Tutti i componenti che sono sequenziali sono sensibili ad un reset asincrono che riporta il componente in uno stato di partenza. Gli stessi componenti hanno anche una porta di reset sincrono (chiamata *soft_rst*), utilizzata per il loro reset dopo che un'immagine è stata equalizzata, in caso dovesse esserne fornita un'altra prima del termine della computazione.

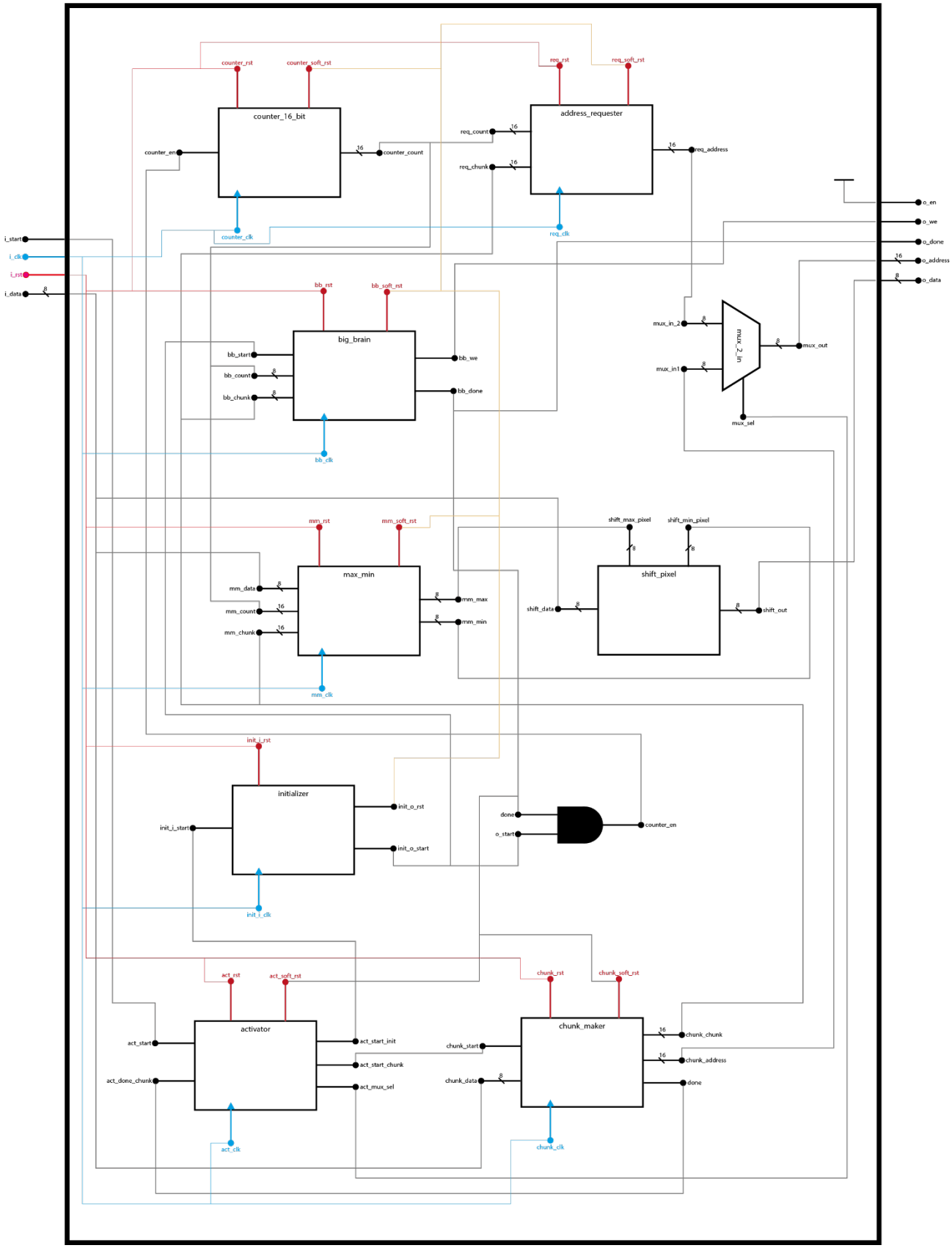


Figura 3: visione macroscopica del progetto

ARCHITETTURA

activator

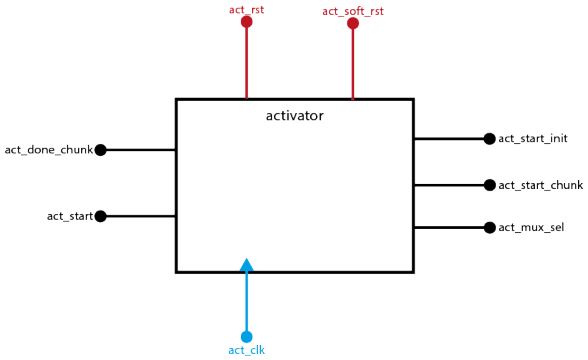


Figura 4: interfaccia di activator

activator è il primo componente in cui scorre corrente quando il processo inizia. Esso si occupa di avviare nell'ordine corretto le due fasi descritte precedentemente di calcolo del CHUNK ed equalizzazione dell'immagine. In principio, tramite dei segnali di controllo, attiva il **chunk_maker**. Questo procede con la sua computazione e risponde con un segnale di *done*. Quando **activator** riceve questo segnale disattiva **chunk_maker** per attivare **initializer**. In ciascuna fase si occupa anche di selezionare l'uscita corretta del multiplexer, in modo da mandare in uscita a *o_address* il segnale fornito dal componente attivo in quel momento.

chunk_maker

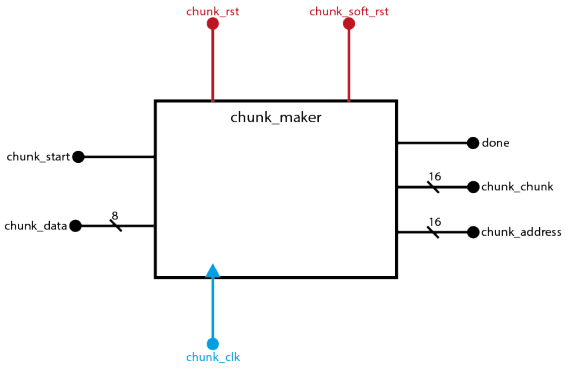


Figura 5: interfaccia di chunk_maker

chunk_maker è il componente che si occupa di calcolare il CHUNK di memoria da leggere, equalizzare e scrivere. Esso si occupa di richiedere il byte 0 e il byte 1 alla memoria, per poi moltiplicarli tra loro e fornire il segnale CHUNK a **max_min**, **big_brain**, e **address_requester**.

initializer

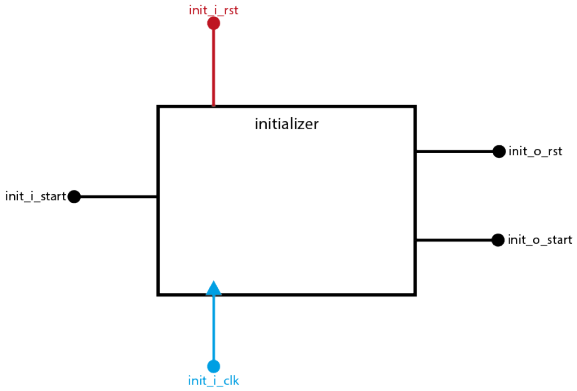


Figura 6: interfaccia di initializer

initializer si occupa di preparare la macchina per seconda fase di computazione e in caso di computazioni successive. Quando il segnale di *start* viene alzato, prima di avviare la seconda fase, viene dato un segnale di *soft_reset* ai componenti cui interessa la seconda fase. Successivamente viene attivato il **counter_16_bit**, e in cascata si attiveranno tutti gli altri componenti, poiché sensibili al valore *count*.

initializer è stato descritto con una descrizione structural come tutti gli altri componenti, tuttavia per fini didattici è stato realizzato anche con una descrizione FSM. Nel top module, quando il componente viene istanziato, viene specificato di utilizzare la descrizione FSM.

counter_16_bit

counter_16_bit è un contatore a 16 bit con reset asincrono e un segnale di *enable*. Ad ogni fronte di salita del clock, se enable=1 il valore del contatore viene incrementato. Se il contatore dovesse raggiungere il valore limite 2^{16}-1, il valore successivo sarà 0.

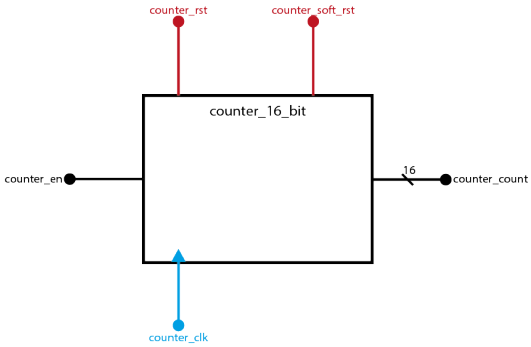


Figura 8: interfaccia di counter_16_bit

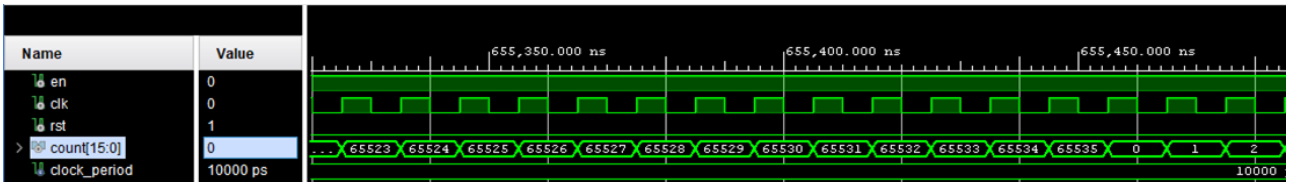


Figura 7: forma d'onda del segnale count_count quando raggiunge il valore massimo

address_requester

address_requester si occupa di richiedere gli indirizzi corretti alla memoria. In un primo momento dovrà richiedere gli indirizzi da 2 a CHUNK+1, in un secondo momento dovrà invece alternare una richiesta di un pixel dell'immagine compreso tra 2 e CHUNK+1 a una richiesta di un byte compreso tra CHUNK+2 e 2*CHUNK+1 per scrivere il risultato del pixel equalizzato.

In figura 10 e 11 sono riportati due esempi di come varia il valore di *req_address* in funzione del valore di *count*.

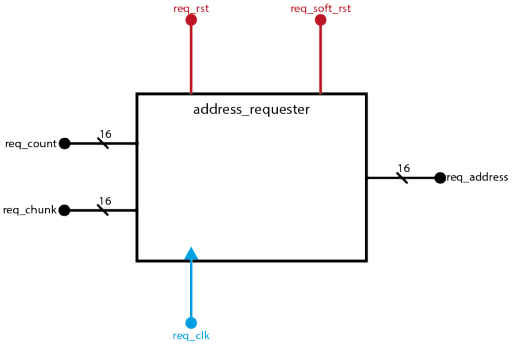


Figura 9: interfaccia di address_requester

COUNTER	O_ADDRESS			
0	0			
1	1			
2		2		
3		3		
4		4		
5		5		
6		6		
7			2	
8				7
9			3	
10				8
11			4	
12				9
13			5	
14				10
15			6	
16				11

Figura 11: andamento di req_address quando CHUNK = 5

COUNTER	O_ADDRESS			
0	0			
1	1			
2		2		
3		3		
4		4		
5		5		
6		6		
7		7		
8		8		
9		9		
10			2	
11				10
12			3	
13				11
14			4	
15				12
16			5	
17				13
18			6	
19				14
20			7	
21				15
22			8	
23				16
24			9	
25				17

Figura 10: andamento di req_address quando CHUNK = 8

max_min

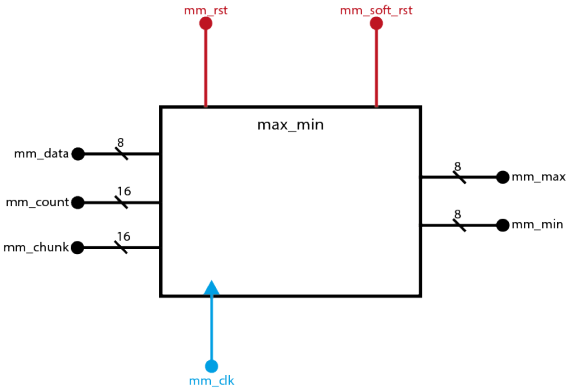


Figura 12: interfaccia di max_min

max_min è l'unico componente che è stato descritto come composizione di componenti al di fuori del modulo principale. Esso comprende due componenti **check_max** e **check_min** e due registri $MAX e $MIN controllati dai rispettivi componenti.

I dati provenienti dalla memoria entrano in entrambi i registri e in entrambi i componenti, tuttavia verranno memorizzati in un registro solo **se check_max o check_min** lo attiverà.

check_max attiverà $MAX solo se il dato sarà maggiore del contenuto del registro e analogamente **check_min** solo se sarà minore.

Per evitare che il contenuto dei registri non venga aggiornato correttamente – soprattutto quello di $MIN in quanto inizializzato a 0 – entrambi i componenti attiveranno i rispettivi registri in corrispondenza del primo dato proveniente dalla memoria.

shift_pixel

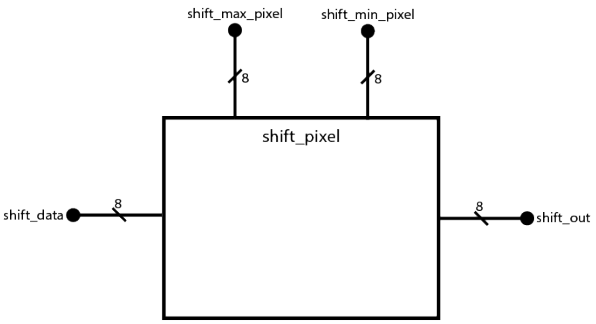


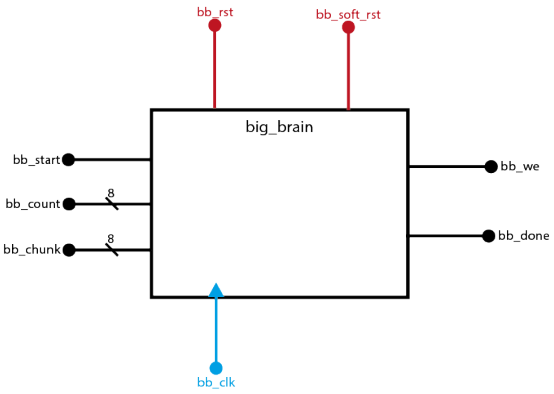
Figura 14: interfaccia di shift_pixel

shift_pixel è l'unico blocco di logica combinatoria della macchina. Esso, a fronte di valori stabili forniti da $MAX e $MIN, è in grado di calcolare lo SHIFT_LEVEL tramite dei controlli a soglia mostrati in figura 14, e successivamente effettuare uno shift logico a sinistra, un numero di volte pari a SHIFT_LEVEL.

FROM	TO	SHIFT
0	0	8
1	2	7
3	6	6
7	14	5
15	30	4
31	62	3
63	126	2
127	254	1

Figura 13: valore di shift_level in funzione di delta_value

big_brain



big_brain si occupa della gestione dei segnali *o_done* e *o_we*. Questi valori possono essere calcolati univocamente in funzione di *count* e CHUNK.

Il segnale di *write_enable* indica alla memoria che il dato in *o_data* è da scrivere all'indirizzo *o_address*, mentre il done, come da specifica, si alza al termine della computazione per poi abbassarsi quando si abbassa il segnale *i_start*.

RISULTATI SPERIMENTALI

Il componente così descritto è stato sintetizzato e sottoposto a vari test con verifica automatica dei risultati.

Ulteriori dettagli sulla sintesi sono mostrati nelle figure successive:

Name	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (285)	BUFGCTRL (32)
project_reti_logiche	329	132	38	1
act (activator)	9	2	0	0
brian (big_brain)	10	2	0	0
chunk_calculator (chunk_maker)	128	45	0	0
counter (counter_16_bit)	98	16	0	0
edges (max_min)	29	16	0	0
init (initializer_fsm)	39	3	0	0
requester (address_requester)	1	48	0	0
shifter (shift_pixel)	38	0	0	0

Figura 15: report di utilizzo

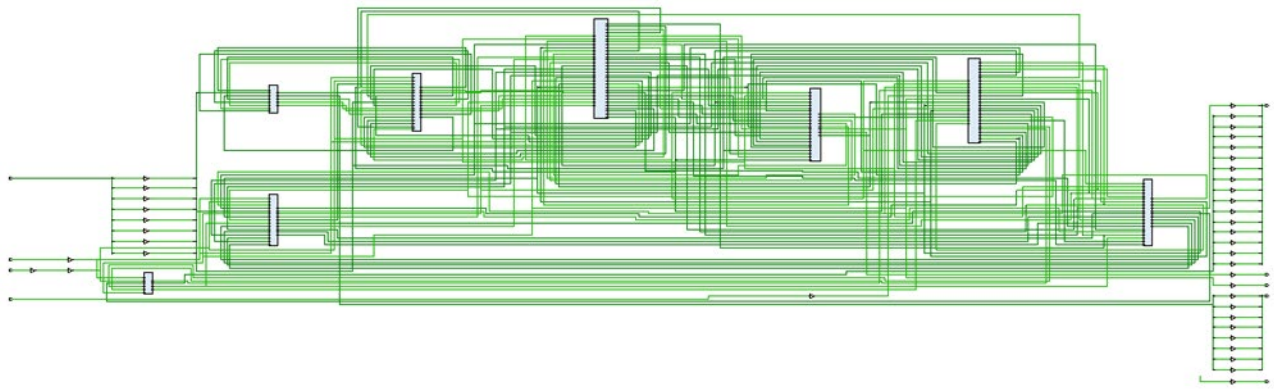


Figura 16: design sintetizzato

I punti più critici, che hanno causato l'introduzione del componente **activator**, sono state le immagini con un CHUNK di dimensione 0,1,2 e 3. Si verificava infatti che il CHUNK non era ancora stato calcolato quando si iniziava la fase di lettura dell'immagine, rendendo impossibile decidere in maniera deterministica quanti pixel era necessario leggere.

Altri aspetti a cui è stata dedicata attenzione sono stati gli algoritmi per calcolare l'uscita del *o_we* e del *o_address*, la cui gestione in fase iniziale di sviluppo si è rivelata più impegnativa del previsto.

Di seguito sono riportati i risultati dei test più significativi che sono stati effettuati post sintesi funzionale.

In figura 17 si può vedere dal segnale *i_start* e *o_done* un test che sottopone il componente a 10 immagini consecutive.

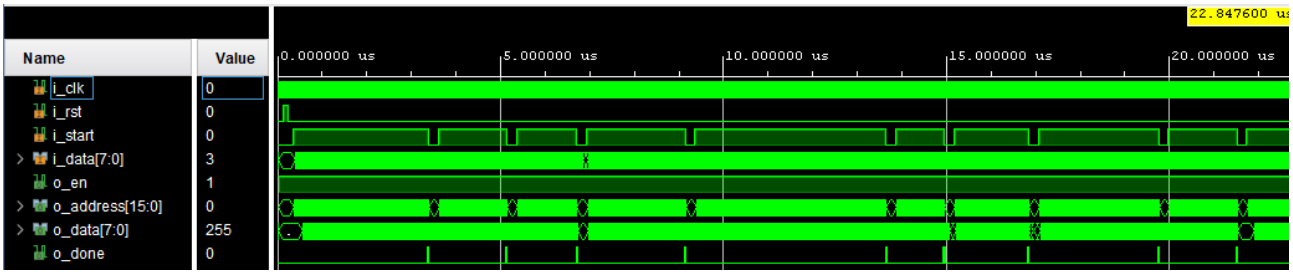


Figura 16: 10 immagini consecutive

In figura 17 si può vedere il comportamento corretto della macchina nei casi in cui il CHUNK sia uno dei valori critici descritti precedentemente.

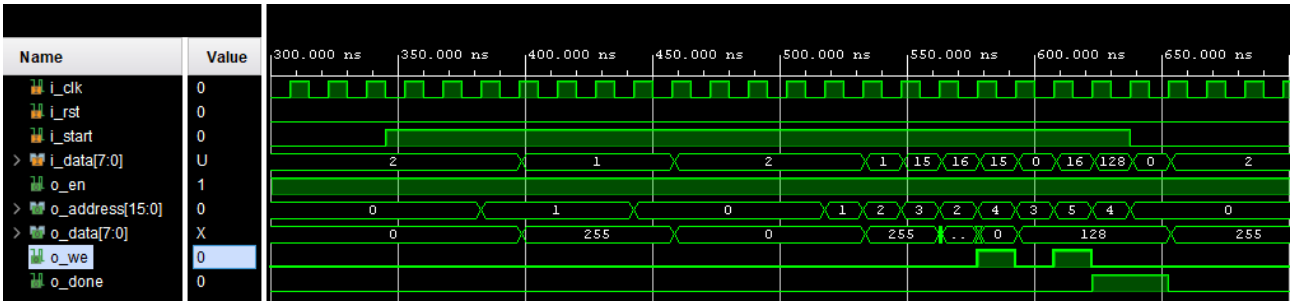


Figura 17: CHUNK = 2

Oltre ai test su situazioni individuate come critiche, sono stati effettuati anche altri test su condizioni che, pur non individuate come possibili fonti di errori, presentavano comunque delle particolarità.

Questi test hanno verificato il corretto funzionamento nei seguenti casi:

- immagine di 128x128;
- segnale di reset asincrono alzato durante una computazione;
- immagini con pixel tutti uguali;
- immagini con pixel tutti diversi;
- immagini con pixel tutti 0;
- immagini con pixel tutti 1;
- immagini con pixel tutti 255;
- immagini che contenevano il valore massimo e minimo nel primo e ultimo pixel;
- immagini che contenevano il valore massimo e minimo nell'ultimo e nel primo;
- i test forniti da esempio nelle specifiche del progetto;
- un test per ogni possibile *delta_value*;

Tutti i test sono stati superati, ad eccezione dell'ultimo test fornito nelle specifiche, tuttavia si è rivelato errato il test fornito, in quanto a fronte di un valore massimo di 255 e di un valore minimo di 0, *shift_level* vale 0, dunque la memoria deve essere riscritta esattamente come la si legge. Nel test fornito è presente il valore 69 nel byte 5 mentre è presente il valore 68 nel byte 17.