Prova finale Reti Logiche

Anno accademico 2022/2023

Studente: Davide Spinelli

Indice

1.Introduzione

1.1 Scopo del Progetto

Lo scopo della prova finale è l’implementazione tramite VHDL di un componente hardware in grado di ricevere l’indirizzo di una locazione di memoria e di indirizzarne il contenuto verso una delle quattro uscite, anch'essa ricavata dall’ingresso seriale. Le uscite forniscono i bit della parola in memoria in parallelo.

1.2 Specifiche Generali

All'istante iniziale le uscite del componente hanno i seguenti valori:

- Z0, Z1, Z2, Z3 sono 0000 0000
- DONE è 0

Il componente non parte nella lettura dei dati in input finchè non riceve il primo segnale di reset. A quel punto attende che il segnale START sia alto(=1) e a quel punto inizia a leggere i dati sull'ingresso seriale primario W. Una volta che il segnale START torna basso(=0) la lettura termina e il componente a questo punto richiede alla memoria il dato corrispondente all'indirizzo ricevuto in ingresso e lo memorizza in un registro in modo tale che quando il segnale DONE passa ad alto(=1), per un ciclo di clock, possa dare in uscita i valori letti da memoria. I dati letti in ingresso sono costituiti nel seguente modo:

- 2 bit per segnalare il canale su cui deve essere mandato il dato letto da memoria
- N bit di indirizzo di memoria. N è compreso tra 0 e 16

Nel caso N sia minore di 16 l'indirizzo viene esteso con 0 sul bit più significativo.

1.3 Descrizione del Modulo

1.3.1 Interfaccia del Modulo

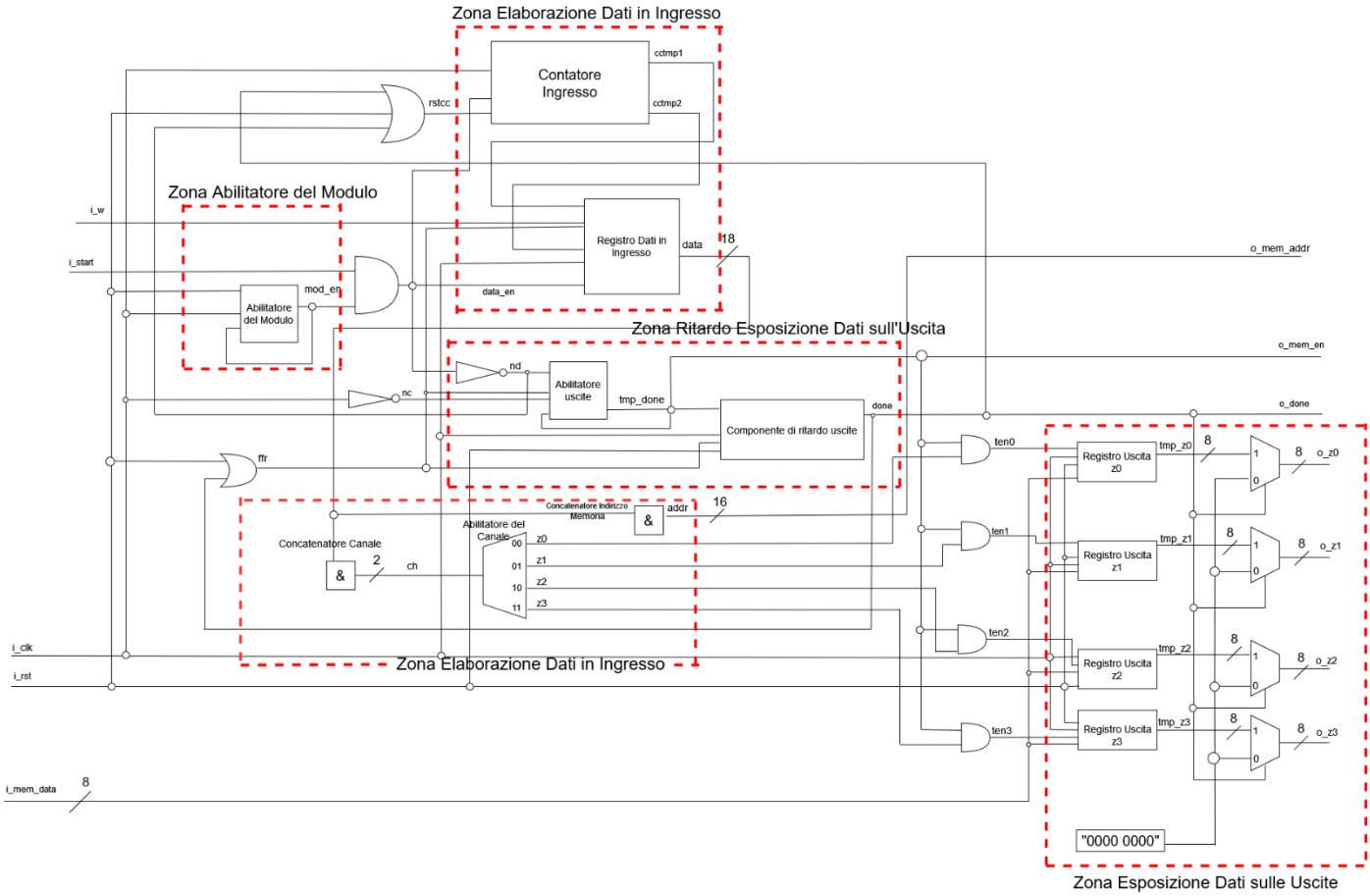
Il componente da descrivere possiede la seguente interfaccia:

```
entity project_reti_logiche is

        port (

                i_clk : in std_logic;

                i_rst : in std_logic;

                i_start : in std_logic;

                i_w : in std_logic;

                o_z0 : out std_logic_vector(7 downto 0);

                o_z1 : out std_logic_vector(7 downto 0);

                o_z2 : out std_logic_vector(7 downto 0);

                o_z3 : out std_logic_vector(7 downto 0);

                o_done : out std_logic;

                o_mem_addr : out std_logic_vector(15 downto 0);

                i_mem_data : in std_logic_vector(7 downto 0);

                o_mem_we : out std_logic;

                o_mem_en : out std_logic;

        )

end project_reti_logiche;
```

In particolare:

- i_clk rappresenta il segnale di CLOCK in ingresso generato dal Test Bench;
- i_rst rappresenta il segnale di RESET in ingresso dal Test Bench, inizializza la macchina in attesa del primo segnale di START;
- i_start è il segnale di START generato dal Test Bench;
- i_w è il segnale W precedentemente descritto e generato dal Test Bench;
- o_z0, o_z1, o_z2, o_z3 sono i quattro canali di uscita;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione;
- o_mem_addr è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- i_mem_data è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_mem_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_mem_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0.

1.3.2 Descrizione del Modulo



Nella mia implementazione delle specifiche si possono individuare cinque aree di interesse:

- abilitazione del modulo;
- lettura dati in ingresso;
- elaborazione dei dati in ingresso;
- ritardo all'esposizione dei dati sulle uscite;
- esposizione dei dati elaborati sulle uscite;

La sezione di abilitazione del modulo si occupa del riconoscimento del primo segnale di reset che porta il modulo nello stato base da cui poi potrà iniziare a leggere i dati dagli ingressi.

La sezione relativa alla lettura dei dati in ingresso si occupa di abilitare il registro adibito al salvataggio dei dati in ingresso solo quando il segnale START è alto e il modulo ha ricevuto almeno un segnale di reset. Il salvataggio dei dati

sfrutta un registro che salva i bit di indirizzo dei canali d'uscita nei primi due bit partendo dal MSB mentre i bit relativi agli indirizzi di memoria vengono salvati nei 16 bit restanti sfruttando un simil shift-registry partendo dal LSB. Questo permette di avere sia il canale di uscita che l'indirizzo di memoria a cui accedere subito dopo che la lettura dall'ingresso finisce poichè l'indirizzo di memoria sarà già esteso a 16 bit qualsiasi sia la lunghezza che è stata ricevuta dall'ingresso.

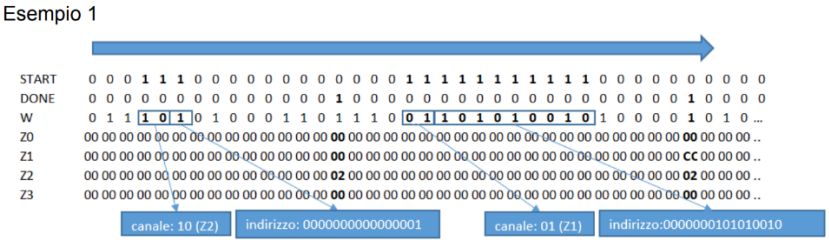
La parte relativa all'elaborazione dei dati abilita il registro temporaneo relativo all'uscita segnalata durante la lettura dei dati in ingresso e aspetta che il dato venga letto da memoria e arrivi all'entrata dei registri temporanei (nel mio caso ho dato tempo quattro cicli di clock).

La sezione relativa al ritardo dell'esposizione dei dati sulle uscite una volta che il segnale di abilitazione della lettura torna basso manda un segnale ai vari componenti coinvolti con l'elaborazione dei dati e ritarda l'alzamento del segnale di done di quattro cicli di clock per permettere la corretta elaborazione. Una volta scaduti i quattro cicli di clock fa alzare il segnale done abilitando le uscite e resettando se stesso(per avere done alto solo un ciclo di clock) e le componenti relative alla lettura dei dati da input.

La sezione di esposizione dei dati elaborati sulle uscite infine aspetta che il segnale done sia alto per mandare sulle relative uscite i dati presenti nei registri temporanei. Quando done è basso presenta sulle uscite dei vettori di bit nulli.

1.3.3 Esempio di Funzionamento

Esempio 1

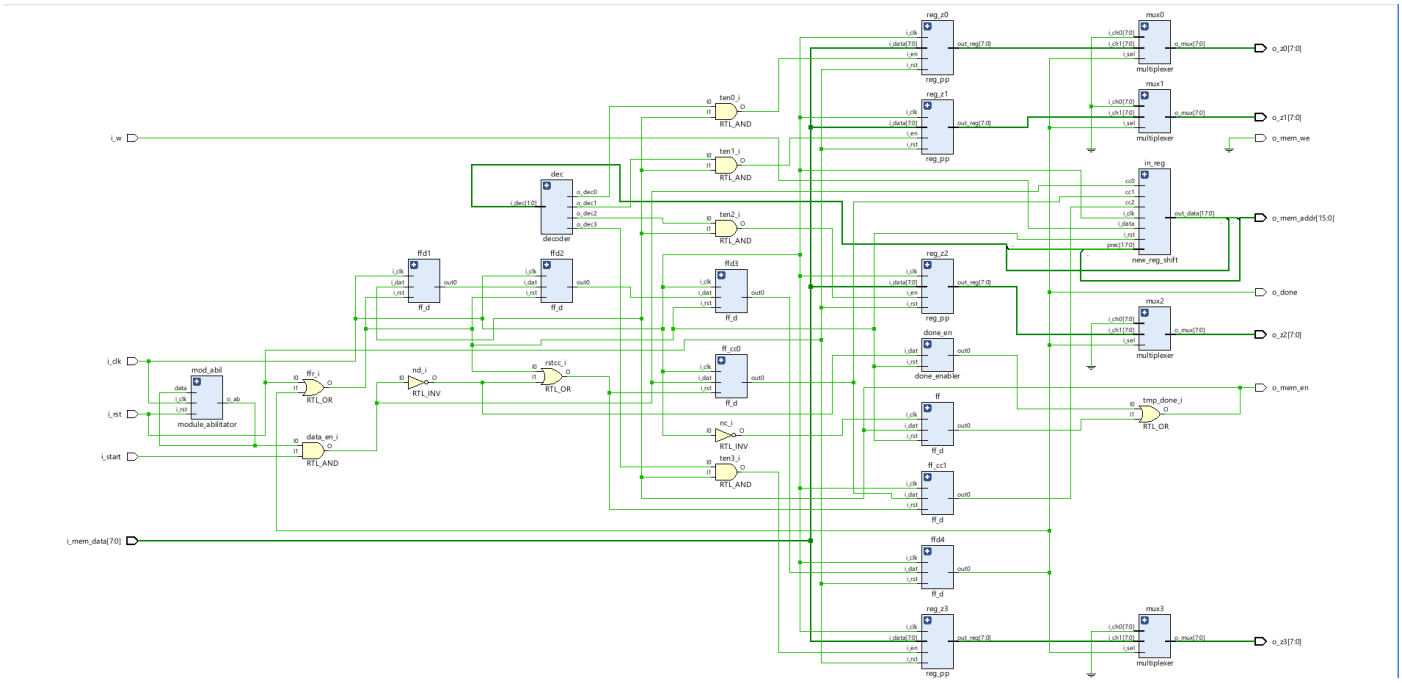


		reset	00	00	00	00	[LSB … MBS | LSB MSB]
CANALE	INDIRIZZO	DATO	Z0	Z1	Z2	Z3	W utile
02	0001	**02**	00	00	02	00	**101**
01	0152	**CC**	00	CC	02	00	**01001010110**

L'esempio nell'immagine appena mostrata ci presenta il funzionamento del modulo dopo il ricevimento del primo segnale di RESET. Possiamo vedere come i dati presenti sul segnale W sono ignorati fintanto che il segnale START è basso(=0). Nel momento in cui il segnale START passa ad alto(=1) procediamo a leggere i bit dal segnale W: i primi due rappresenteranno il canale su cui andremo a mostrare i dati mentre gli altri, fino ad un massimo di 16, indicheranno l'indirizzo di memoria in cui leggere il dato. Una volta che il segnale START torna basso(=0) smettiamo di leggere i dati dal segnale W. Attendiamo un numero di cicli di clock inferiore a 20 per vedere il segnale DONE andare ad alto(=1) per un solo ciclo di clock e mostrare sui canali di uscita i dati letti da memoria.

2.0 Architettura

2.1 Schema Dettagliato del Modulo



2.2 Elenco dei Segnali Utilizzati

Segnali a un bit

- mod_en: rappresenta l'effettiva comparsa di un segnale RESET alto e quindi l'abilitazione del modulo;
- data_en: rappresenta l'effettiva possibilità di leggere dati dal segnale W, è un and tra i segnali START e mod_en;
- tmp_done: rappresenta l'effettiva fine della fase di lettura dei dati e l'inizio della fase di ricezione dei dati dalla memoria e loro esposizione sulle uscite, ottenuto da un or tra i segnali t1 e t2;
- z0, z1, z2, z3: rappresentano la scelta della rispettiva uscita da aggiornare con il nuovo dato letto da memoria;
- ten0, ten1, ten2, ten3: rappresentano l'abilitazione del rispettivo registro di memoria per la rispettiva uscita, sono un and tra i segnali tmp_done e z0, z1, z2 o z3;
- tmp1, tmp2, tmp3: rappresentano segnali con cui il segnale done viene ritardato prima di essere mandato in uscita;
- done: rappresenta l'attivazione delle uscite con la relativa esposizione dei relativi dati, inoltre rappresenta un reset del sistema per l'attesa di un nuovo ciclo di lettura dati;
- cctmp1: rappresenta la lettura del primo bit di canale;
- cctmp2: rappresenta la lettura del secondo bit di canale e il passaggio alla lettura dei bit di indirizzo di memoria;
- nd: utile per segnalare la fine della lettura dei dati da input, è rappresentato dalla negazione del segnale data_en;
- rstcc: rappresenta l'effettivo segnale di reset dei flip-flop che tengono memoria della lettura dei bit di canale, è costituito da un or tra i segnali RESET, done e nd;
- ffr: effettivo segnale di reset per il registro che salva i dati arrivati da W e per il sistema di ritardamento dell'esposizione dei dati sulle uscite, è costituito da un or tra i segnali done e RESET;
- nc: negato del CLOCK, utile per avere il valore di tmp_done mezzo ciclo di clock prima
- t1,t2: segnali utili per il calcolo di tmp_done, derivano rispettivamente dall'uscita del modulo done_enabler e un flip-flop di tipo D utile per salvarsi il valore di tmp_done;

Segnali a 18 bit

- data: rappresenta i dati letti dal segnale W, sia l'uscita da aggiornare che l'indirizzo di memoria;

Segnali a 2 bit

- ch: rappresenta l'uscita su cui deve essere aggiornato il dato;

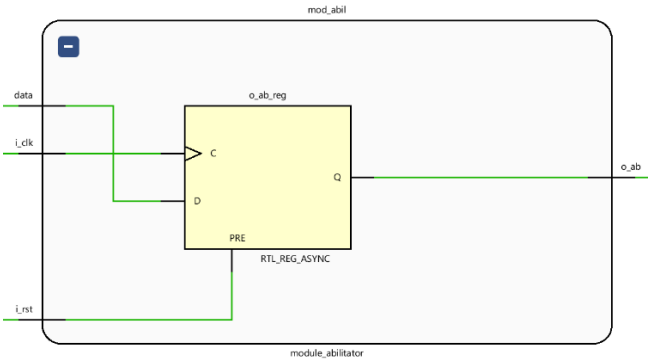
Segnali a 8 bit

- tmp_z0, tmp_z1, tmp_z2, tmp_z3: rappresentano i segnali che portano i dati letti da memoria dai registri temporanei all'uscita corrispondente;

Inoltre viene usata una costante nv che rappresenta un vettore di otto bit tutti a zero utile nella fase di esposizione dei dati sulle uscite.

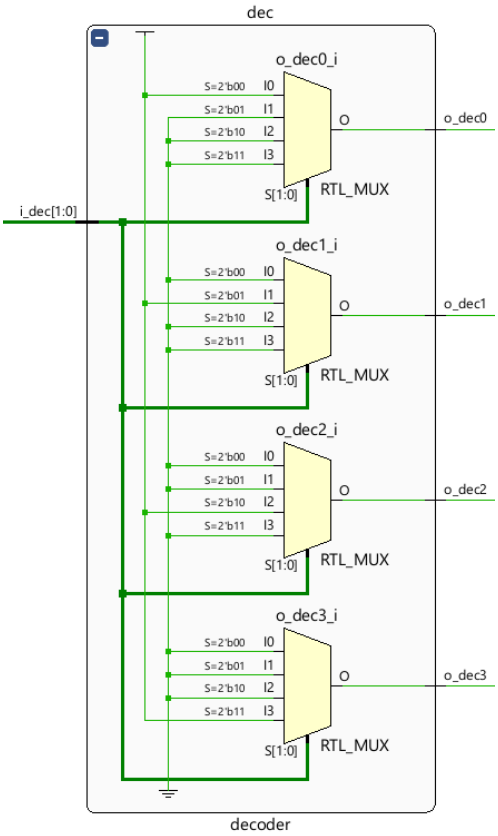
2.3 Elenco dei Componenti Utilizzati

module_abilitator



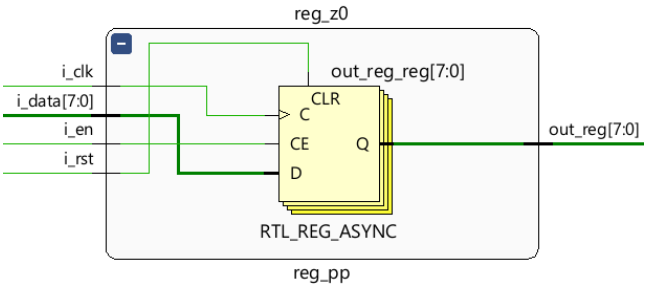
Consiste nel componente che permette il riconoscimento del primo segnale di RESET e conseguentemente abilita il modulo a poter iniziare a leggere i dati da input quando riceverà un segnale di START alto. Comportamento: se riceve in ingresso un segnale di reset alto l'uscita sarà 1 altrimenti sul fronte di salita del segnale di clock porta in uscita il dato precedente.

Decoder



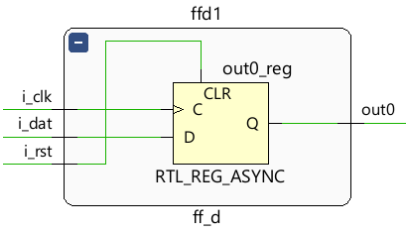
Consiste in un componente che implementa un decoder a quattro uscite. Riceve in ingresso un vettore di due bit e con esso attiva una delle quattro uscite. Comportamento: dato un segnale in ingresso porta ad 1 l'uscita corrispondente mentre porta a 0 tutte le altre uscite.

Reg_pp



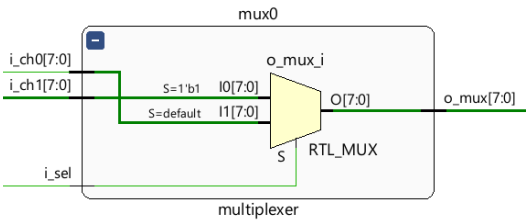
Consiste in un componente che implementa un registro parallelo/parallelo che si aggiorna sostituendo il valore salvato all'interno con il dato in ingresso ogni qualvolta che il segnale abilitatore è alto. Comportamento: se il segnale di reset è 1 il dato contenuto nel registro diventa un vettore di 0 altrimenti se è sul fronte di salita del clock controlla che il segnale abilitatore sia 1 e nel caso aggiorna l'uscita con il valore corrispondente al dato in ingresso.

ff_d



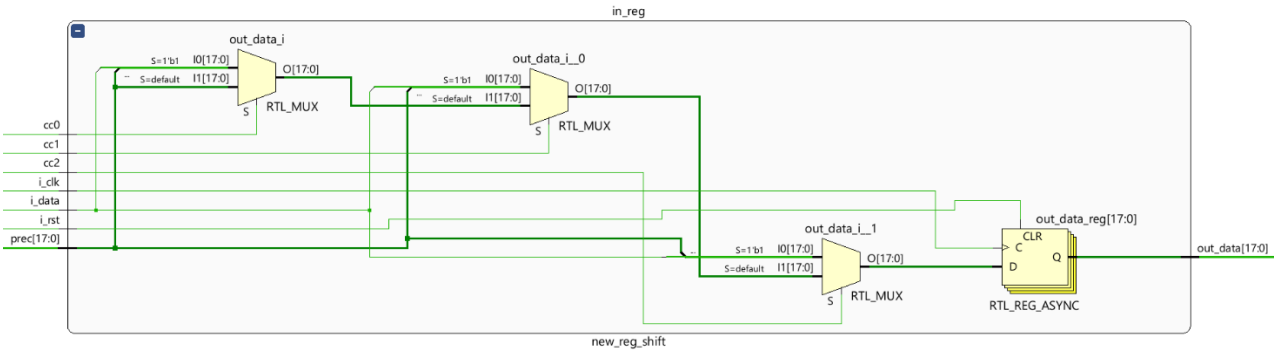
Implementa un flip-flop di tipo D. Comportamento: se il segnale di reset è pari a 1 l'uscita viene impostata a 0 altrimenti se è sul fronte di salita del clock l'uscita è pari al valore precedente.

multiplexer



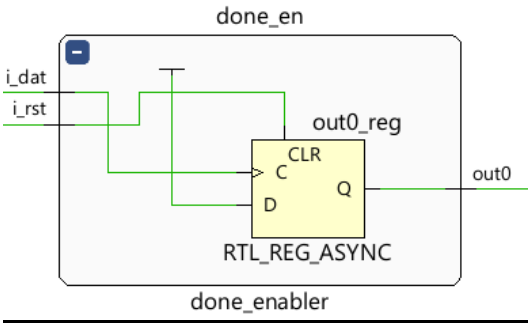
Consiste in un componente implementante un multiplexer a 8 bit. Comportamento: l'uscita è pari all'ingresso 0 se il segnale di selezione è pari a 0, altrimenti è pari all'ingresso 1.

new_reg_shift



Consiste in un modulo che implementa un registro a 18 bit in cui i primi due bit dal MSB saranno utilizzati per memorizzare il canale su cui andrà scritto il dato, mentre i restanti sedici saranno popolati tramite una logica a shift-registry partendo dal LSB. Il componente presenta due ingressi che gli indicheranno in che punto della lettura si trova(lettura dei bit di indirizzo di canale o lettura di bit di indirizzo memoria). Comportamento: se il segnale di reset è 1 allora l'uscita corrisponde a un vettore di 0 altrimenti se si trova sul fronte di salita del clock controlla in che stato della lettura si trova. Se l'ingresso corrispondente alla lettura del primo bit in ingresso è 1 allora il MSB dell'uscita corrisponderà al bit letto in ingresso e il resto rimarrà invariato, altrimenti se l'ingresso corrispondente alla lettura del secondo bit in ingresso è 1 allora il secondo MSB dell'uscita corrisponderà al bit letto in ingresso e il resto rimarrà invariato altrimenti se l'ingresso corrispondente alla lettura dei bit di indirizzo memoria è a 1 allora i primi due bit dal MSB dell'uscita corrispondono ai primi due bit dal MSB precedenti mentre i restanti bit shiftano di uno a sinistra e LSB diventa pari al bit in ingresso. Infine se nessuna di queste condizioni è soddisfatta allora l'uscita rimane invariata dal valore precedente.

done_enabler



Consiste in un componente che attende l'abbassamento del segnale data_en(il negato viene fornito come dato in ingresso) e fa partire un segnale per indicare che devono essere mostrati i dati sulle uscite. Comportamento: se RESET diventa 1 espone sull'uscita 0 mentre se si trova sul fronte di salita di i_dat espone sull'uscita 1.

3.0 Risultati Sperimentali

3.1 Sintesi

Report Timing

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.173 ns	Worst Hold Slack (WHS):	0.216 ns	Worst Pulse Width Slack (WPWS):	4.500 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:	98	Total Number of Endpoints:	98	Total Number of Endpoints:	58

Report Utilization(Slice Logic)

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	33	0	0	134600	0.02
LUT as Logic	33	0	0	134600	0.02
LUT as Memory	0	0	0	46200	0.00
Slice Registers	58	0	0	269200	0.02
Register as Flip Flop	58	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Dai report di sintesi di timing e utilization si può intuire come il modulo rimanga nei limiti di tempo considerati e di come non implementi nessuna logica tramite latch.

3.2 Simulazioni

Di seguito riporto le mie deduzioni riguardanti il testing del modulo attraverso i testbench forniti e testbench aggiunti al fine di controllare casi non coperti dai primi.

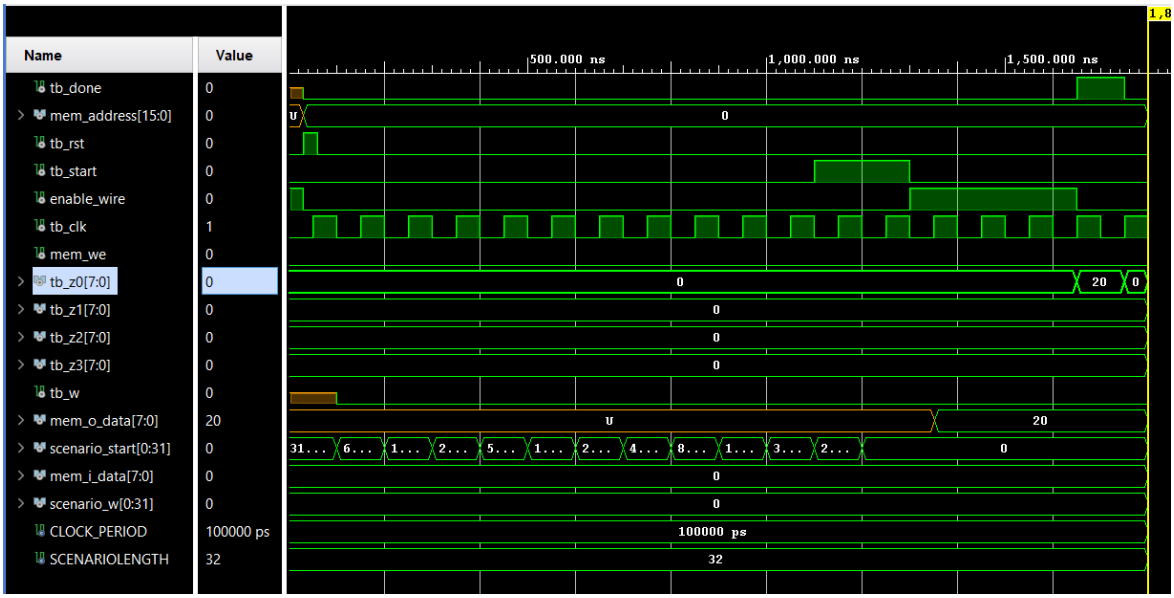
3.2.1 Testbench Forniti

I testbench forniti vanno a coprire i seguenti casi:

- il corretto funzionamento del modulo, cioè lettura dei dati da W solo dopo l'arrivo di un segnale di reset;
- segnale di start attivo per due cicli di clock(indirizzo di memoria da leggere deve essere 0000 0000 0000 0000);
- il corretto funzionamento di un secondo segnale di reset dopo il reset iniziale;
- la scrittura dei dati su tutti i canali di uscita;
- la corretta sostituzione di un dato su un canale di uscita;
- la lettura di un reset lungo un solo ciclo di clock;
- segnale di reset dato solo all'inizio della simulazione;
- segnale di start attivo per diciotto cicli di clock(indirizzo di memoria non richiede un'estensione);
- corretto funzionamento anche quando vengono ripetuti degli input dopo un segnale di reset;
- corretto funzionamento quando tutti i bit letti dall'ingresso W sono uguali;
- testa la corretta inizializzazione di tutti i segnali e di tutte le uscite relative alla memoria da cui bisogna leggere i dati;

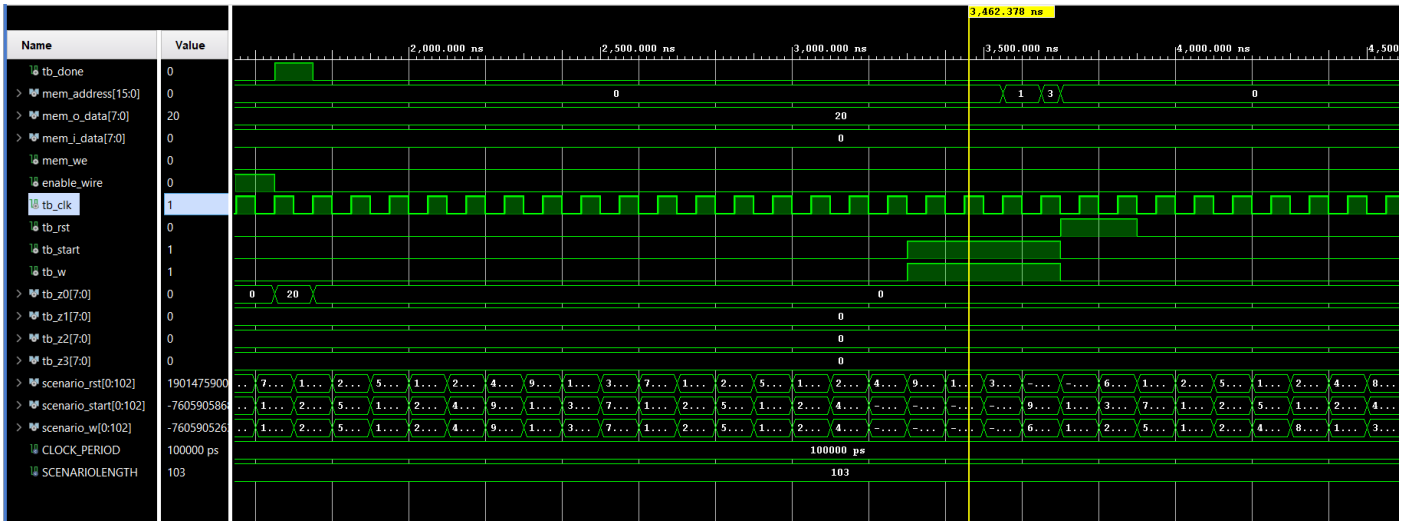
3.2.2 Casi Non Coperti dai Testbench Forniti

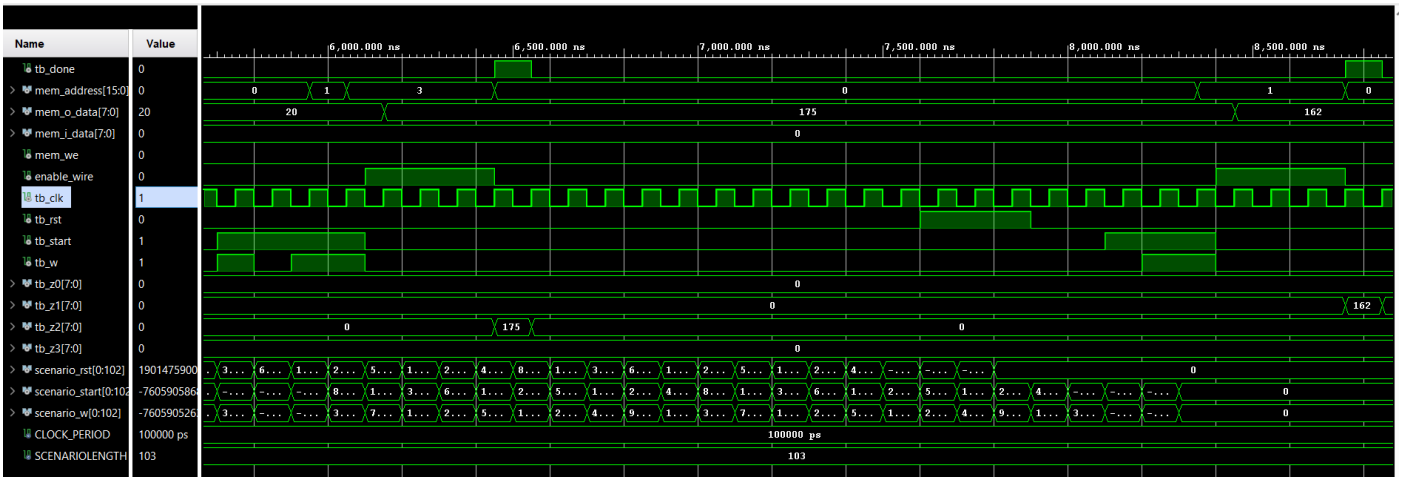
Reset asincrono



Dal grafico delle forme d'onda si può notare come il modulo sintetizzato all'arrivo di un reset asincrono e lungo meno di un ciclo di clock risponda correttamente e porti il modulo nello stato di partenza.

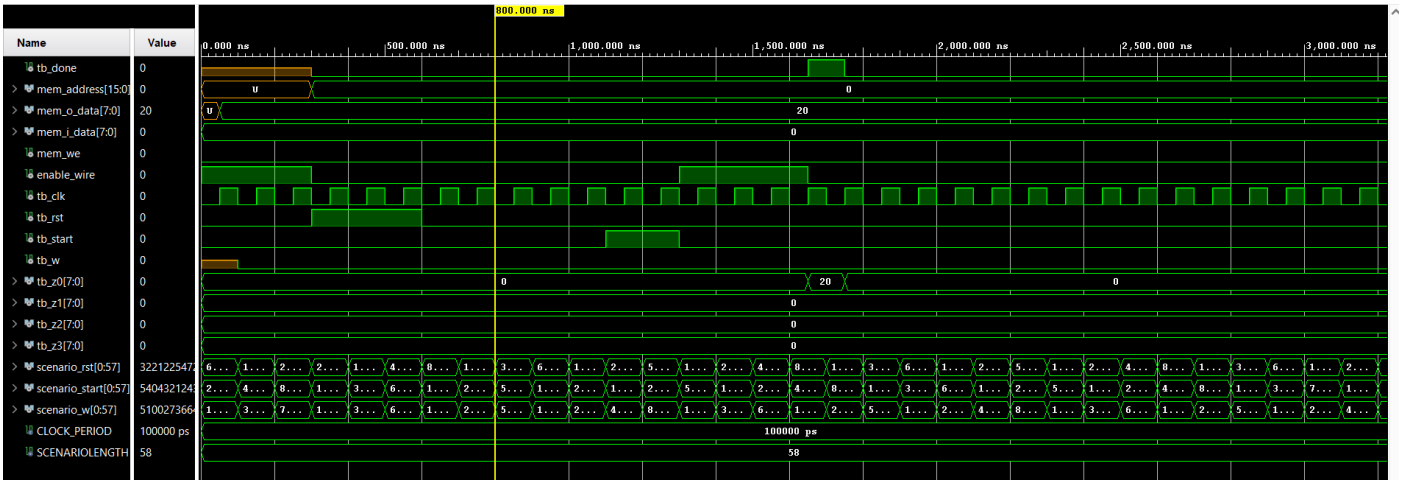
Reset alto durante la fase di esposizione dati sulle uscite

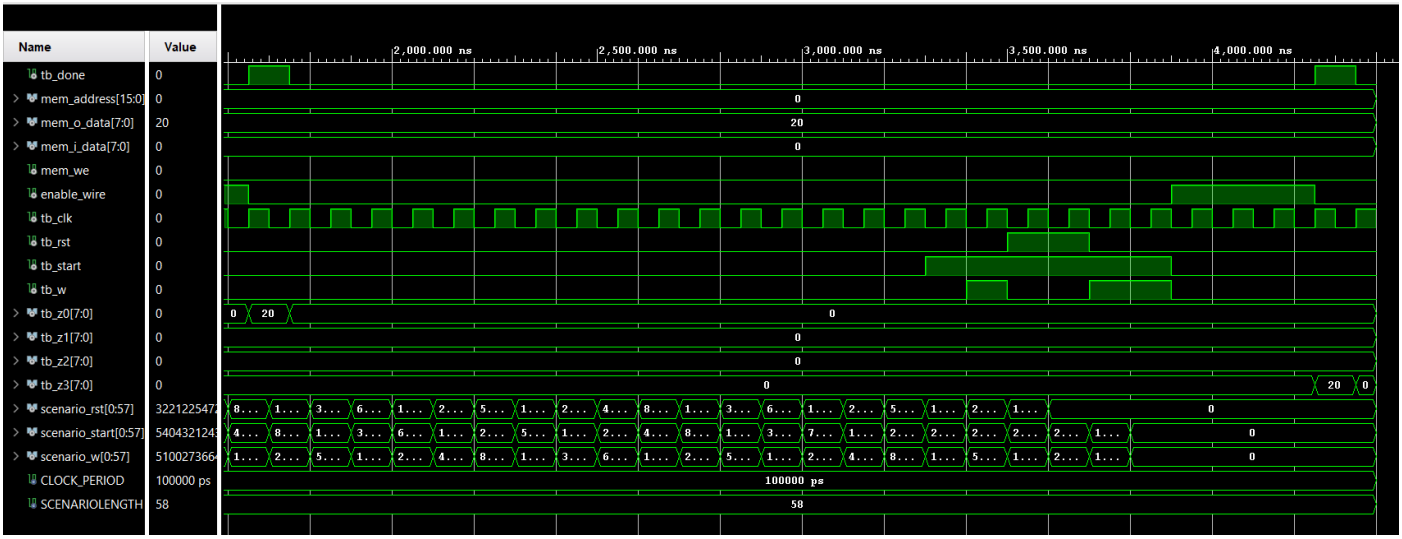




Dai grafici delle forme d'onda si può notare come il modulo sintetizzato all'arrivo di un segnale di reset tra la fine della lettura dei dati dall'ingresso e l'esposizione dei dati letti da memoria sull'uscita resetta i dati saltati e salta completamente la fase di esposizione dati mentre se arriva dopo l'esposizione dei dati resetta semplicemente i dati tenuti in memoria.

Reset durante start attivo





Dai grafici delle forme d'onda si può notare come all'arrivo di un segnale di reset nel mentre è attivo il segnale di start comporta un reset delle uscite, come mostrato dal fatto che subito dopo al momento dell'esposizione dei dati viene esposto solo il nuovo dato letto dall'ingresso.

4.0 Conclusioni

Visti i risultati sperimentali la sintesi del modulo rispetta correttamente le specifiche fornite.

Un’implementazione alternativa della specifica ha previsto l'utilizzo di una macchina a stati finiti in cui i moduli di module_enabler e done_enabler sono stati gestiti tramite passaggi tra stati e non tramite componenti specifici.

L’implementazione del modulo tramite macchina a stati finiti ha utilizzato molti più elementi di memoria di quelli usati dall’implementazione iniziale evidenziando quindi come il primo approccio consenta un risparmio di blocchi hardware interni all’FPGA.