Progetto Finale Di Reti Logiche

Giuseppe Bocci

A.A. 2022-2023

Contents

1 Introduzione

Il progetto consiste nell'implementatre un modulo HW, descrivendolo in VHDL, che si interfacci con una memoria con indirizzi da 16 bit e dati da 8 bit. Il modulo ha quattro canali di uscita, ognuno da 8 bit, e due ingressi primari da un bit: W e START. W è un segnale, rappresenta i dati di ingesso che sono: 2 bit (sempre dati) che identificano uno dei quattro canali di uscita e da 0 a 16 bit, immediatamente successivi, che rappresentano l'indirizzo di memoria al quale andare a prendere il dato, per poi salvarlo sul canale precedentemente identificato. START se a 1 logico, indica che la sequenza in ingresso su W è valida. START può essere ad 1 logico da 2 a 18 cicli di clock consecutivi, minimo due per leggere i bit di canale).
Alcuni chiarimenti su W:

- Deve essere letto su fronte di salita del clock (da specifiche).
- Se l'indirizzo non è completo (16 bit) i bit rimantenti si considerano come zeri e sono i più significativi. Es: quando START è 1 e W=11111, allora il canale di uscita è il quarto(11=3), e l'indirizzo è 0000000000000111 dove il bit più a sinistra è il MSB.
- Il primo bit delle sequenze in ingresso rappresenta sempre il bit più significativo. Es: quando START è 1 e W = 10110 (il bit più a sinistra è quello fornito per primo), rappresenta il canale 2 (10) e l'indirizzo 0000000000000110 .

Salvato il dato sul canale di uscita selezionato, il modulo deve portare l'uscita DONE a 1 logico per un solo ciclo di clock e contemporaneamente mostra in uscita i dati salvati sui quattro canali. Per tutto il tempo che DONE è a 0, il modulo deve mostrare 00000000 su ogni canale.
Altro segnale in ingresso è il RESET: quando a 1 i dati salvati sui canali di uscita vanno cancellati e si attende che START vada ad 1. É garantito che venga dato il RESET a 1 all'accensione del dispositivo HW.

Come detto, il dispositivo si interfaccia con una memoria, alcune considerazioni:

- La memoria riceve lo stesso segnale di clock del dispositivo HW e funziona sul fronte di salita.
- Il tempo di risposta della memoria alla richiesta di un dato è di due nano secondi.
- L'indirizzo di memoria da dove leggere è formito in ingresso in parallelo alla memoria dal dispositivo HW.
- Il dato letto dalla memoria è fornito in ingresso in parallelo al dispositivo HW dalla memoria.
- La memoria ha altri due ingressi: l'enable che deve essere posto a 1 per effettuare qualsiasi tipo di operazione sulla memoria e il write enable che deve essere posto a 0 per operazioni di lettura e a 1 per operazioni di scrittura sulla memoria. La memoria va in stato di errore se i due ingressi assumono un valore diverso da 0 o 1.

L'entita del modulo HW chiamata project_reti_logiche è data:

```
entity project_reti_logiche is
    port (
        i_clk : in std_logic;  % segnale di clock
        i_rst : in std_logic;  % RESET
        i_start : in std_logic;  % START
        i_w : in std_logic;  % W
        o_z0 : out std_logic_vector(7 downto 0);  % channel di uscita 0
        o_z1 : out std_logic_vector(7 downto 0);  % channel di uscita 1
        o_z2 : out std_logic_vector(7 downto 0);  % channel di uscita 2
        o_z3 : out std_logic_vector(7 downto 0);  % channel di uscita 3
        o_done : out std_logic;  % DONE
        o_mem_addr : out std_logic_vector(15 downto 0);  % indirizzo di memoria
        i_mem_data : in std_logic_vector(7 downto 0); % dato letto dalla memoria
        o_mem_we : out std_logic; % write enable memoria
```

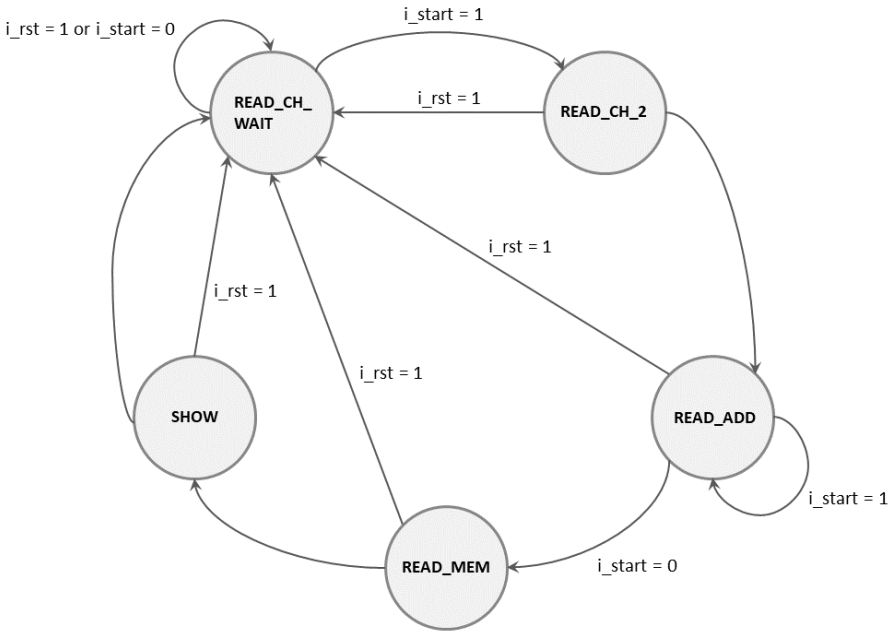
```
        o_mem_en : out std_logic  % enable memoria
    );
end project_reti_logiche;
```

2 Progettazione

Poiché il segnale i_w è seriale e si ha necessità di usare i bit forniti in parallelo, si è optato per dei registri SIPO(Serial Input Parallel Output) alla lettura di i_w. Vista la necessità di far persistere i dati in uscita in parallelo e dato che sono forniti in parallelo, si è optato per dei registri PIPO(Parallel Input Parallel Output) collegati ai canali o_zx in output.
Le fasi di creazione della FSM e progettazione dell'architettura non sono state completamente a sé stanti, ma la prima ha aiutato maggirmente alla formalizzazione del problema.

2.1 Finite State Machine

Facendo queste considerazioni si è giunti alla seguente macchina a stati finiti:



FSM di Moore

Si è optato per una FSM di Moore, perché è più facile individuare stati superflui. Le uscite dovrebbero essere rappresentate dentro gli stati della FSM, ma una questione di spazio sono riportate nella seguente tabella:

Tabella delle uscite	
Stato	Uscite
READ_CH_WAIT	en_c=1, en_a_s=0, o_mem_en=0, save=0, done=0
READ_CH_2	en_c=1, en_a_s=0, o_mem_en=0, save=0, done=0
READ_ADD	en_c=0, en_a_s=1, o_mem_en=1, save=0, done=0
READ_MEM	en_c=0, en_a_s=0, o_mem_en=0, save=1, done=0
SHOW	en_c=0, en_a_s=0, o_mem_en=0, save=0, done=1

Possiamo considerare lo stato READ_CH_WAIT come stato iniziale, in quanto è garantito il segnale di reset all'accensione del modulo. Nella FSM non è specificato in quanto formalmente sbagliato.
Il flusso della FSM è il seguente:

1. READ_CH_WAIT: viene dato i_rst a 1 per iniziare. L'enable del registro SIPO a 2 bit che memorizza il channel(en_c) viene posto a 1 e quindi si abilita la memorizzazione dei bit forniti da i_w. Il registro accetta i bit anche quando i_start=0 o i_rst=1, questo non è un problema perché il registro ha dimensione 2 e i 2 bit del channel sono garantiti, quindi i precedenti bit errati andranno semplicemente persi, infatti questo è anche uno stato di wait, per questo si è voluto evidenziare l'autoanello. Quando i_start=1 il bit assume significato, sul fonte di salita del clock si legge il bit e si passa allo stato successivo.

2. READ_CH_2: permette la lettura del secondo e ultimo bit del channel sul fronte di salita e contemporaneamente si passa sempre allo stato successivo.

3. READ_ADD: porta l'enable del registro SIPO dell'address (en_a_s) a 1. L'indirizzo viene letto fintanto che il segnale i_start è a 1. Quando i_start è 0 si passa allo stato successivo. In questo stato avviene anche la lettura del dato dalla memoria (o_mem_en = 1), nello specifico avviene ogni ciclo di clock sul fronte di salita che determina il passaggio allo stato successivo o la permanenza nello stato corrente. Nel caso in cui il segnale i_start sia a 0 al terzo ciclo di clock complessivo, e quindi non debba essere letto l'indirizzo da i_w, non è un problema perché, come si vedrà più chiaramente in seguito, l'enable del registro SIPO a 16bit consiste nell'and tra il e_a _s e il segnale di i_start. Seppur non necessario, si è voluto evidenziare l'autoanello con i_start=1.

4. READ_MEM: in questo stato il livello logico del segnale o_mem_en è indifferente, ma è stato scelto di porlo a 0, altrimenti ci sarebbe stata una lettura superflua dello stesso dato. Si è comunque deciso di chiamare questo stato "leggi memoria", anche se l'effettiva lettura avviene quando si entra in questo stato. Il nome deriva dalla sua importanza in tale lettura, infatti senza di esso non si potrebbe getire il delay di risposta della memoria, che se pur piccolo(2ns) è presente(in seguito verranno fatti alcuni esempi). In questo stato il segnale di save viene posto ad 1, in modo tale da poter salvare il dato nei registri PIPO in uscita sul fronte di salita del clock successivo, che determina anche il passaggio allo stato successivo.

5. SHOW: in questo stato il segnale o_done viene posto ad 1 e i dati salvati mostrati in uscita, allo stesso tempo vengono azzerati i registri SIPO. Qusto stato può essere definito "l'ultimo" perché sul fronte di salita successivo si ritorna allo stato iniziale.

Note:

- Ogni stato ha un arco che contesnte il ritorno allo stato iniziale quando i_rst=1 in modo da effettivamente reinizializzare il dispositivo

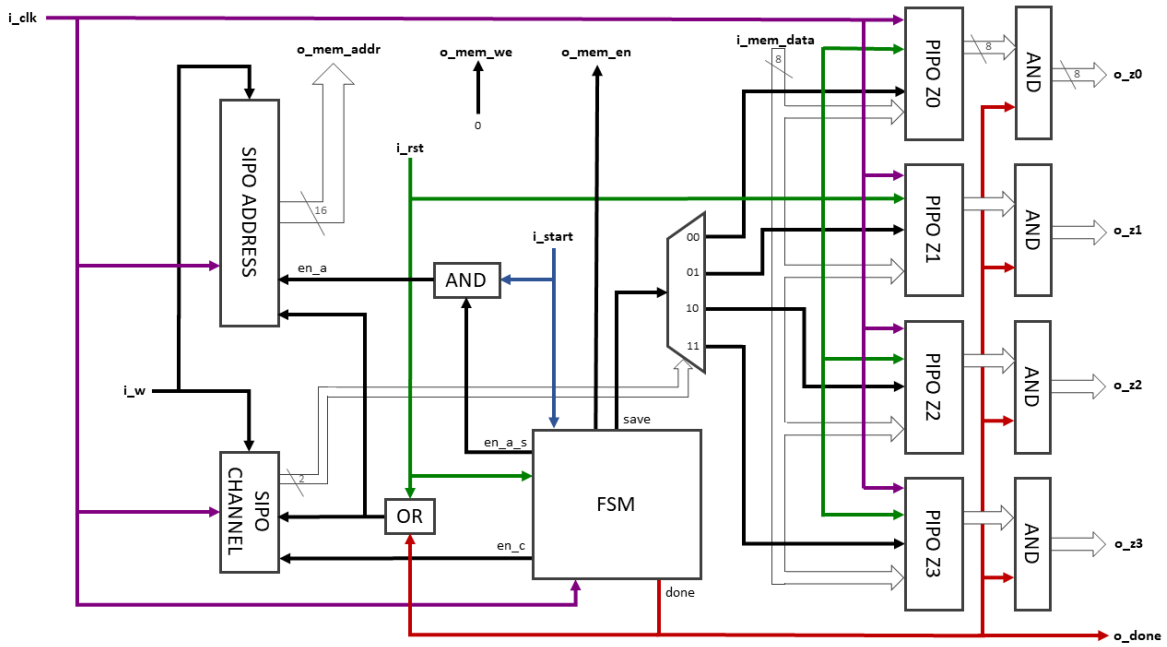
- Si sarebbe potuto usare un solo segnale per save e o_mem _en in quanto non infuliscono rispettivamente gli stati READ_ADD e READ_MEM se posti a 1, ma si è deciso di non farlo per evidenziare la differenza tra i due stati. Si ricorda inoltre che per far funzionare la memoria il segnale o_mem_en può assumere solo i valori logici 0 e 1, non, ad esempio, '-'.

- Gli enable sono necessari per entrambi i registri SIPO in quanto permettono di isolare la sequenza di bit a loro utile, infatti se non fosse presente un segnale di enable non sarebbe possibile leggere con semplicità l'indirizzo di memoria e non sarebbe possibile "conservare" il channel giusto anche durante gli stati successivi al READ_CH_2.

- Nel caso in cui il seganle di start tornasse alto dopo essere andato a 0 in READ_ADD, semplicemente si procederebbe con gli stati successivi fino a READ_CH_WAIT ingorando il segnale perché considerato un errore.

2.2 Architettura

Nella seguente figura è riportato lo schema a blocchi dell'architettura realizzata. I segnali riportati con lo stesso nome utilizzato in VHDL, alcuni di essi hanno bus coloroato per una più facile lettura della figura.



Schema a blocchi

Qui è possibile apprezzare meglio l'enable del registro SIPO ADDRESS descritto parlando dello stato READ_ADD.

Come detto precedentemente si è optato per dei registri SIPO per la memorizzazione dei dati forniti in seriale. I registri sono dotati di reset controllato, come descritto in SHOW, dal segnale done e ovviamente dal i_rst, infatti i due sono in OR logico.
La pulizia del registro SIPO ADDRESS è necessaria perché consente il riempimento, come da specifica, con zeri nei bit più significativi dell'indirizzo della sequenza valida successiva, in quanto già presenti nel registro: Es. con i_w = 01101 e i_start=11111, inizialmente SIPO_ADD=0000000000000000 per passare, al fonte di salita utile alla lettura dell'indirizzo, a SIPO_ADD=0000000000000001, al secondo SIPO_ADD=0000000000000010 e al terzo SIPO_ADD=0000000000000101, questo consente di avere l'indirizzo esatto già pronto senza bisogno di modifiche successive.

Per quanto riguarda il registro SIPO_CHANNEL a 2bit non ci sarebbe bisogno di azzerarlo, perché viene sempre sovrascritto in quanto i due bit di channel sono garantiti, ma è stato implementato comunque un clear in modo da poterlo riusare in futuri progetti e perché la rete di clear non influenza le prestanzioni in quanto comunque presente nel registro SIPO_ADDRESS.

Si noti come l'uscita o_mem_we è posta sempre 0 logico perché non è mai necessaria una scrittura della memoria e se o_mem_en si trovasse ad 1 logico non sarebbe possibile la lettura della stessa.

Il salvataggio del dato nel giusto registro PIPO avviene sfruttando l'enable dei registri: viene usato un demux 1:4 che permette di mandare il segnale di enable (save), al registro selezionato dal SIPO_CHANNEL. Ovviamente non è possibile avere sempre a 1 l'ingresso del demux altrimenti ci sarebbe in ogni stato della FSM un registro che cambia il suo valore in uno sbagliato e non si avrebbe più presistenza dei dati letti da memoria.

Come soluzione al dover mostrare in uscita i dati memorizzati solo quando o_done è a 1 è stato effettuato un AND tra ogni bit savlato nei registri PIPO e il segnale di done stesso.

Si noti come tutti i moduli abbino lo stesso ciclo di clock (memoria inclusa) e funzionino tutti sul fronte di salita.

Si è deciso di descivere in VHDL tutti i vari moduli come componenti a sé stanti, in modo da aumentare

la modularità e la riusabilità del codice. Mentre la FSM, in quanto specifica di questo progetto, è stata descritta direttamente nell'architettura di *project_reti_logiche*.

2.2.1 SIPO Registers

Entrambi i registri SIPO sono stati descritti in VHDL allo stesso modo, l'unica cosa che cambia è il numero di bit dei registri.
Le enità ad essi associati sono:

```
entity SIPO_register2 is
        port(
            clk_i: in std_logic;
            rst_i: in std_logic;
            in_i: in std_logic;
            en_i: in std_logic;
            Q_o: out std_logic_vector(1 downto 0)
        );
end SIPO_register2;

entity SIPO_register16 is
        port(
            clk_i: in std_logic;
            rst_i: in std_logic;
            in_i: in std_logic;
            en_i: in std_logic;
            Q_o: out std_logic_vector(15 downto 0)
        );
end SIPO_register16;
```

Dove:

- clk_i rappresnta il segnale di clock da fornire in ingresso
- rst_i rappresenta il segnale di clear/reset da dare in ingresso
- in_i rappresneta il dato in ingresso
- en_i rappresnta il segnale di enable da fornire in ingresso
- Q_o è il vettore di segnali rappresentante il dato a 16 bit salvato e in uscita

L'architettura è descritta da un solo processo per ogni registro. É stato necessario un segnale vettore Q di appoggio per "rappresentare" i vari Filp Flop. Per implementare un corretto scorrimento dei bit, considerando che il primo bit della sequenza è il più significativo e che si usa una rappresentazione *downto*, sul fronte di salita di ogni ciclo di clock, se l'enable è a 1, il bit viene memorizzato in posizione 0 e i restanti shiftati di una posizione.
Si riporta il codice del *SIPO_register16*:

```
if rst_i = '1' then
    Q <= "0000000000000000";
elsif clk_i'event and clk_i = '1' and en_i = '1' then
    Q(15 downto 1) <= Q(14 downto 0);
    Q(0) <= in_i;
end if;
Q_o <= Q;
```

Si noti che la quarta e quinta riga si possono invertire senza alcun problema.

Ci si aspetta che i registri vengano sintetizzati usando 18 Filp Flop (16+2).

2.2.2 PIPO Registers

Nel componente *PIPO_8* si è deciso di includere l'operazione logica di AND con il done, la fine di un possibile riutilizzo del codice.
L'entità è la seguente:

```
entity PIPO_register8 is
        port(
            en_i: in std_logic;
            rst_i: in std_logic;
            clk_i: in std_logic;
            show_i: in std_logic;
            word_i: in std_logic_vector(7 downto 0);
            word_o: out std_logic_vector(7 downto 0)
        );
end PIPO_register_8;
```

Dove:

- clk_i rappresnta il segnale di clock da fornire in ingresso
- rst_i rappresenta il segnale di clear/reset da fornire in ingresso
- en_i rappresnta il segnale di enable da fornire in ingresso
- word_i è il vettore di segnali che rappresenta il dato a 8 bit in ingresso che verrà salvato sul fronte di salita successivo
- word_o è il vettore di segnali che rappresenta il dato a 8 bit salvato e in uscita
- show_i rappresenta il segnale che determina quando mostrare in uscita(su word_o) il dato salvato o se porre word_o a tutti zeri. Quando show_i è a 1 logico il dato viene mostrato.

Il registro è descritto da due processi, uno sequenziale che descrive il registro vero e proprio e uno combinatorio(dove nella toggle list del process sono inclusi tutti i segnali coinvolti nella rete) che descrive l'AND per mostrare o no il dato in uscita.
Nell'architettura di *project_reti_logiche* i registri sono creati con un for generativo di seguito riportato:

```
genZ: for i in 0 to 3 generate
    Z_regiters: PIPO_register8 port map(
        en_i => enables_Z(i),
        rst_i => i_rst,
        clk_i => i_clk,
        show_i => done,
        word_i => i_mem_data,
        word_o => z_vector(((i+1)*8 - 1) downto i*8)
    );
end generate;
```

dove z_vector è un vettore di 32 segnali logici e enables_Z di 4.

Ci si aspetta che i registri vengano sintetizzati usando 32 Filp Flop (8*4).

2.2.3 Demultiplexer 1:4

Il demultiplexer 1:4 è stato impiegato per abilitare uno solo dei quattro registri PIPO a memorizzare il dato letto dalla memoria, questo grazie al segnale di save proveniente dalla FSM. Ai registri PIPO corrispondono le seguenti codifiche: 00 → Z0, 01 → Z1, 10 → Z2 e 11 → Z3.
Il demux è descritto da un process "combinatorio" e ha come caso di base y=0000.
La sua entità è la seguente:

```
entity demux4 is
        port(
            sel : in std_logic_vector(1 downto 0);
            x : in std_logic;
            y : out std_logic_vector(3 downto 0)
        );
end demux4;
```

Dove:

- sel è il vettore composto dai segnali di selezione in ingresso (2 bit)
- x rappresnta il segnale in ingresso da indirizzare verso l'uscita selezionata
- y è il vettore dei quattro segnali logici in uscita

2.2.4 Architettura di project_reti_logiche

L'architettura di *project_reti_logiche* è stata chiamata *project_reti_logiche_arch* ed è una architettura mista.
L'architettura raccoglie tutti i componenti descritti precedentemente e li interconnette come presentato nello schema a blocchi. Per farlo sono stati necessari dei segnali di appoggio come: done, cls, i segnali dell'FSM ed altri.
La piccola rete combinatoria è stata descritta in dataflow: l'AND tra i_rst e done, l'OR tra i_start e en_a_s e i collegamenti tra z_vector e le uscite o_z0 o_z1 o_z2 o_z3.

L'FSM invece è descritta in behavioral da due processi: uno per la funzione delta e lo stato corrente e uno per la funzione lamda della FSM di Moore.
Il processo *FSM_delta* descrive i cambi di stato e memorizza lo stato corrente in un segnale di tipo S così definito:

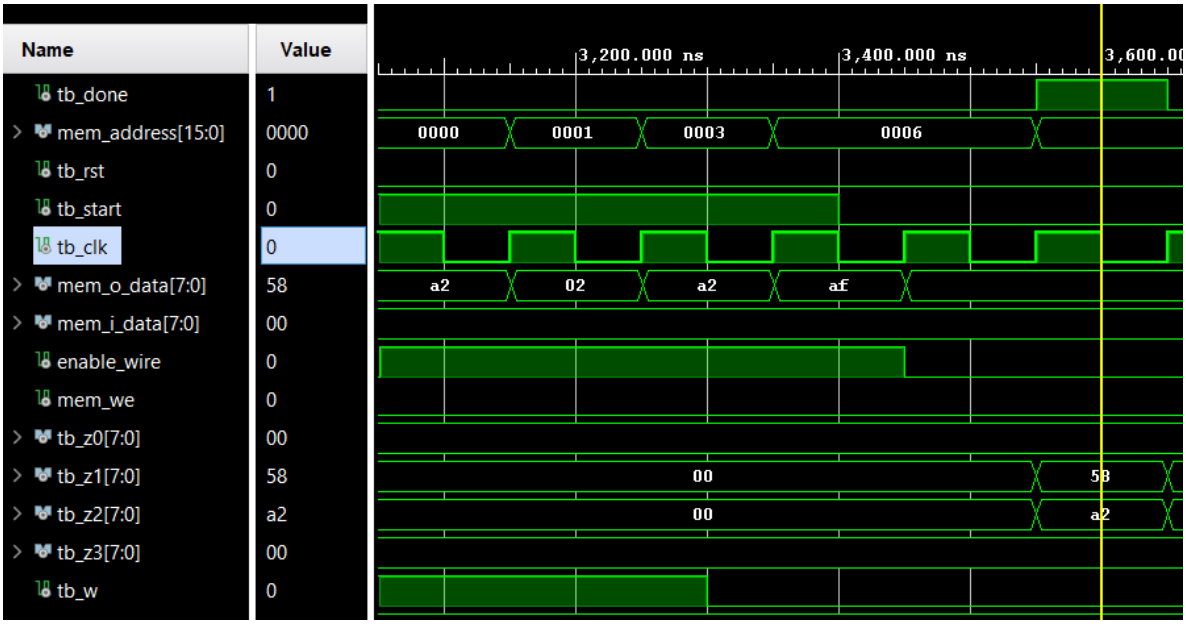
```
type S is (READ_CH_WAIT, READ_CH_2, READ_ADD, READ_MEM, SHOW);
```

I cambi di stato avvengono solo sul fronte di salita del clock, tranne per i cambi di stati legati al reset che possono avvenire in qualsiasi momento (non specificato dalle specifiche del progetto).
Invece il processo *FSM_lambda* descrive le uscite in base allo stato corrente. Il caso base è tutte le uscite a 0 logico.

Ci si aspetta che per memorizzare gli stati, in quanto 5, la FSM venga sintetizzata usando 3 Filp Flop ($2^3 = 8$).

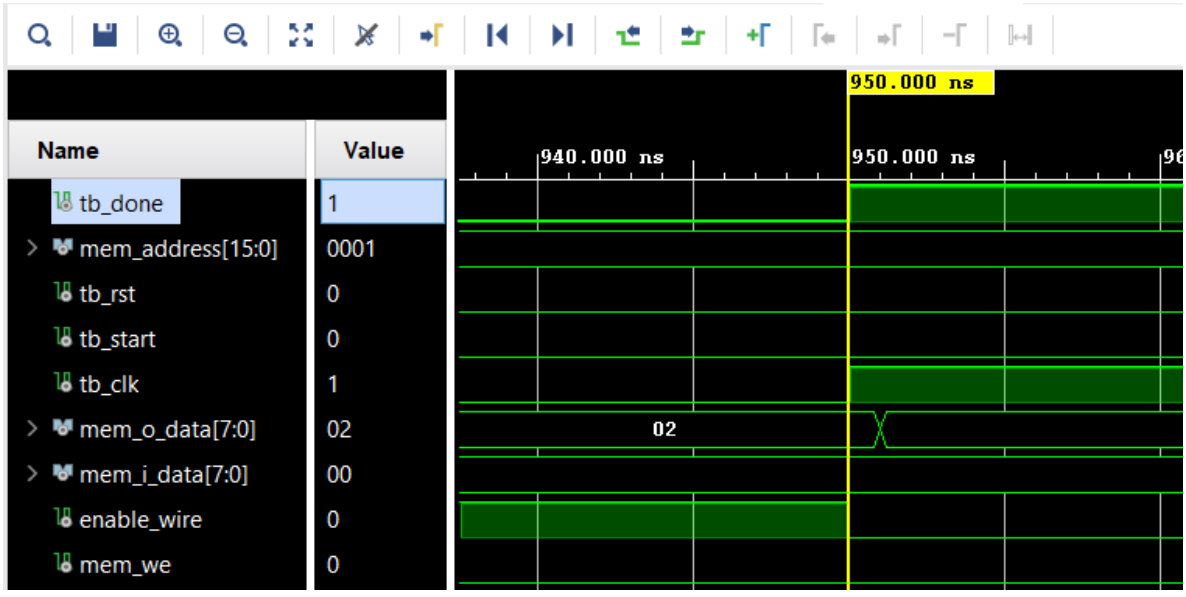
2.3 Motivazione scelte progettuali e soluzioni alternative

Durante la progettazione si è cercato di ridurre il tempo di risposta il più possibile, arrivando ad impiegare solo due cicli di clock dall'ultima acquisizione di i_w. Lo si mostra in figura:



La strategia adottata porta a richiedere dati superflui alla memoria, ma permette di risparmiare uno stato nella FSM e quindi un ciclo di clock.

Per abbassare maggiormente il tempo di risposta bisogna ridurre il numero di stati della FSM dopo che i_start è passato a 0 e prima che o_done passi a 1, in quanto è necessario sia leggere fintanto che i_start è a 1, sia mostrare il risultato in un ciclo di clock. Quindi l'unico stato candidato all'eliminazione è READ_MEM.
Putroppo però esso è necessario, da questo il suo nome come detto prima, perché la memoria risponde con un ritardo di 2ns e anche se si faccia salvare il dato nei registri PIPO alla fine dello stato(fronte di salita) di SHOW e si usasse una rete combinatoria per mandare l'output giusto sull'uscita selezionata (si ricorda che done a 1 pulisce il SIPO_CHANNEL, ma non è necessario farlo), comunque non si rispetterebbe la specifica di avere l'ucita giusta per tutto il tempo che o_done è a 1.
Si può notare il ritardo e la mancata specifica nella seguente immagine:



Probabilmente solo se la memoria lavorasse sul fronte di discesa sarebbe possibile, ma questa opzione non è stata presa in considerazione.

Un'altra possibile ottimizzazione probbe essere ottenuta riducendo il numero di stati a quattro, in modo

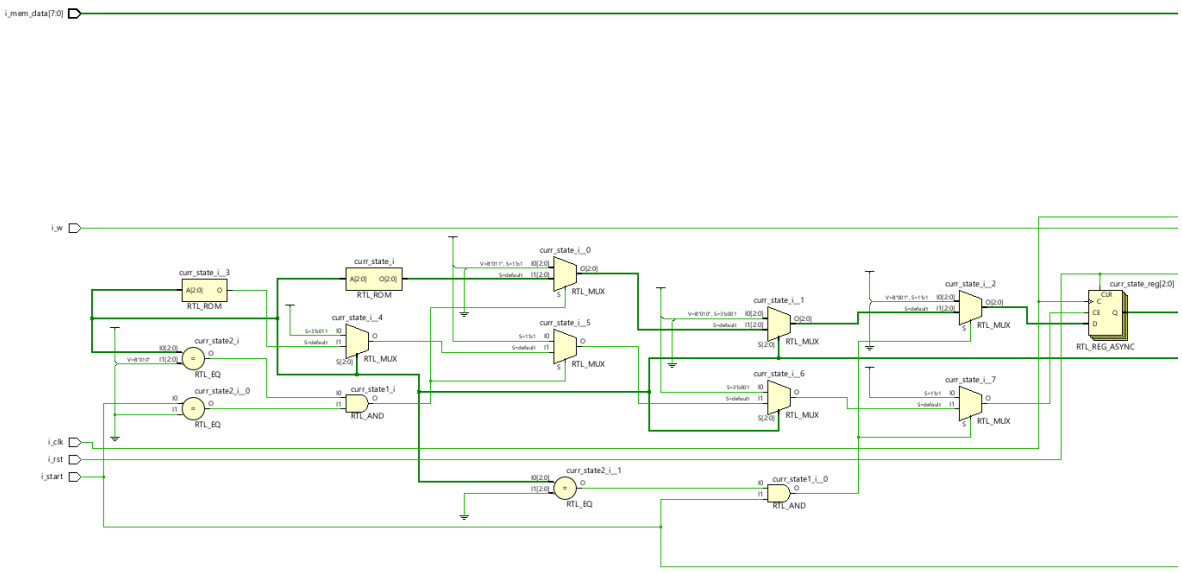
da utilizzare due Flip Flop al posto di tre. Come si è spiegato non è possile rimuovere READ_ADD, READ_MEM e SHOW.
Si potrebbe pensare di eliminare READ_CH_2 in quanto avente le stesse uscite di READ_CH_WAIT, ma READ_CH_2 è usato per contare i bit di channel, toglierlo significherebbe dover contare in altri modi come, ad esempio, aggiungendo un contatore, questo implicherebbe l'uso di almeno un Flip Flop per tenere traccia del conteggio rendendo la modifica inutile, se non controproducente.

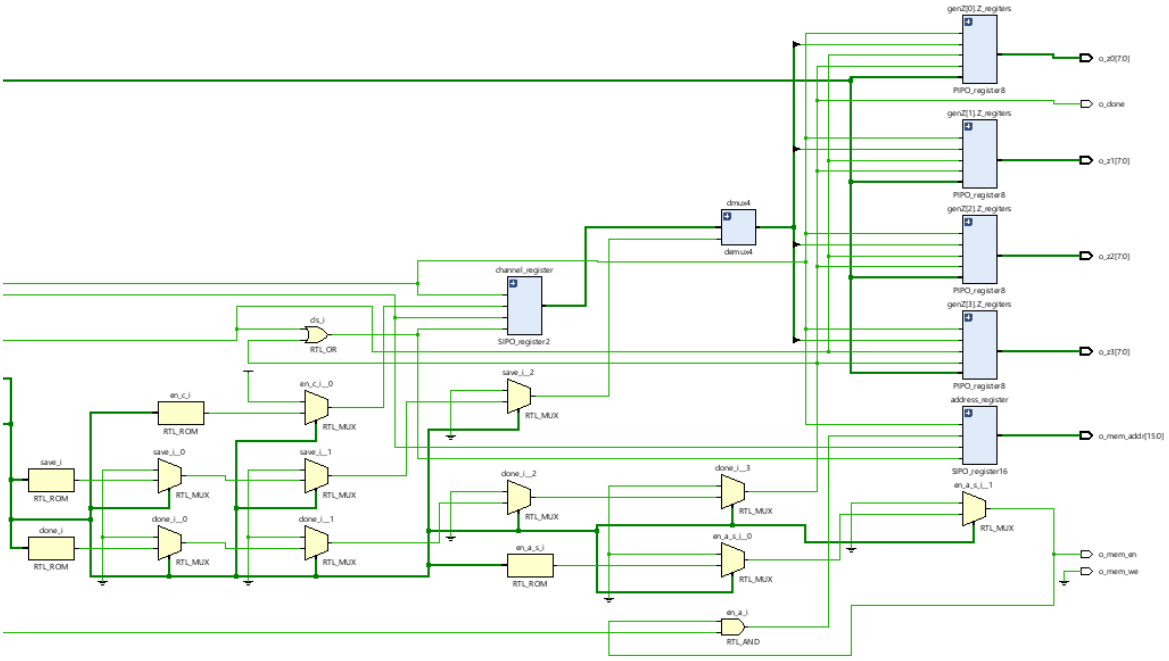
3 Risultati sperimentali

3.1 Sintesi

La sintesi del circuito da parte di VIVADO ha prodotto un circuito costituito da 53 FLIP FLOP come da aspettative: 18 per i registri SIPO, 32 per i registri PIPO e 3 per la FSM. Si noti l'assenza di LATCH.
Lo slack time è di 97.006ns, molto ampio considerando un clock da 100ns. La rete combinatoria più lenta, probabilmente, è quella relativa al segnale di uscita della FSM che pilota il DEMUX, quindi ad essa è dovuto questo tempo.

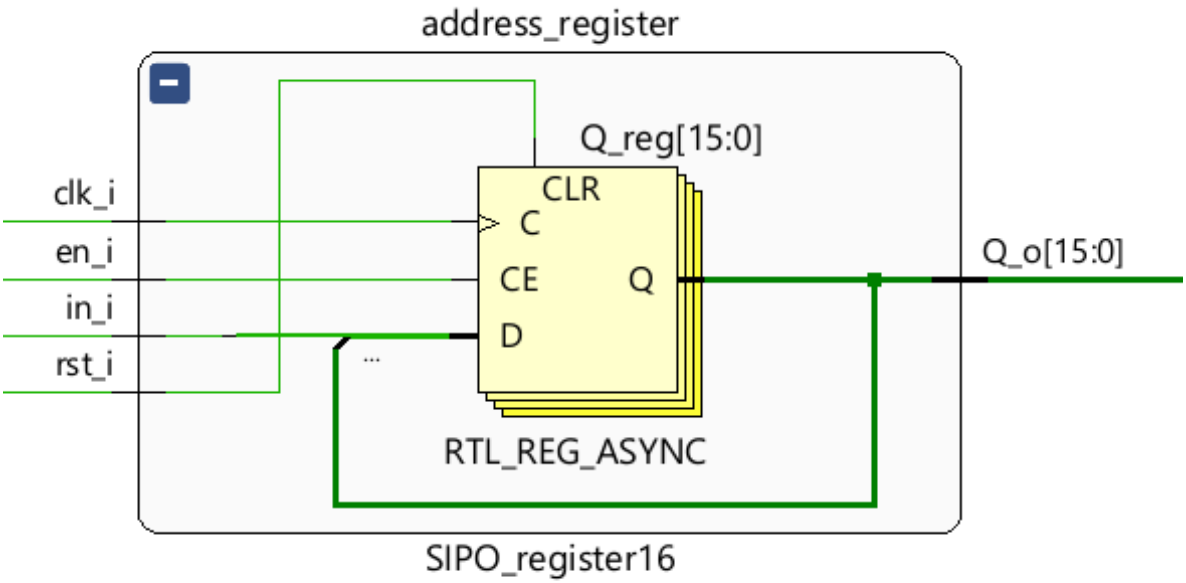
Nelle successive figure è riportata la sintesi dell'intero circuto:





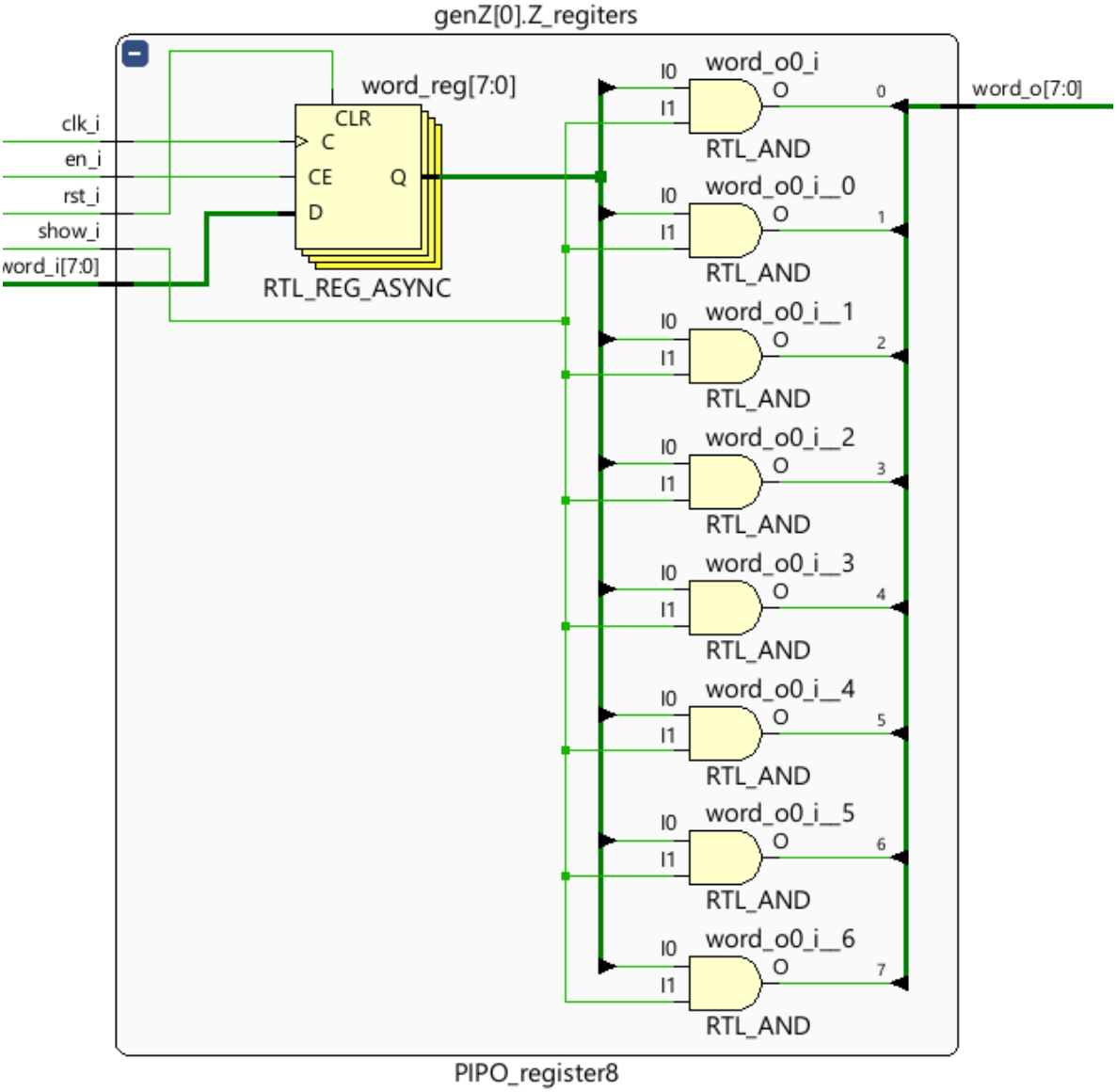
3.1.1 Sintesi SIPO Registers

Di seguito si riporta la sintesi del registro SIPO da 16bit, in qunto quella del registo SIPO da 2bit è molto simile.



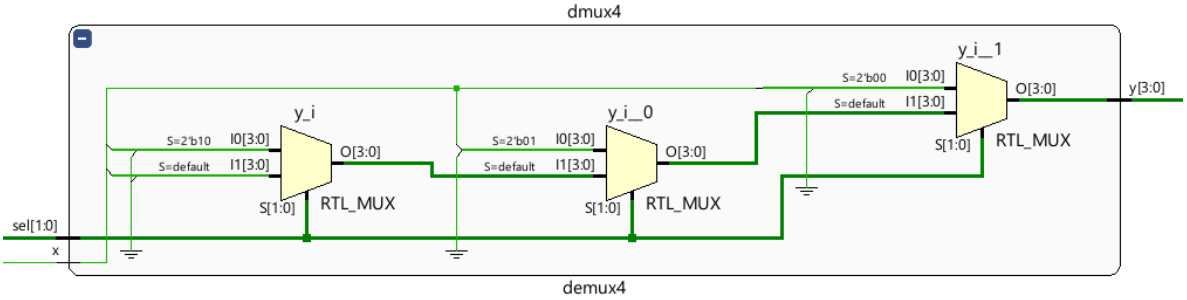
Si può notare come siano stati utilizzati sedici Flip Flop D collegati in cascata e con clock, enable e clr condiviso.

3.1.2 Sintesi PIPO Registers



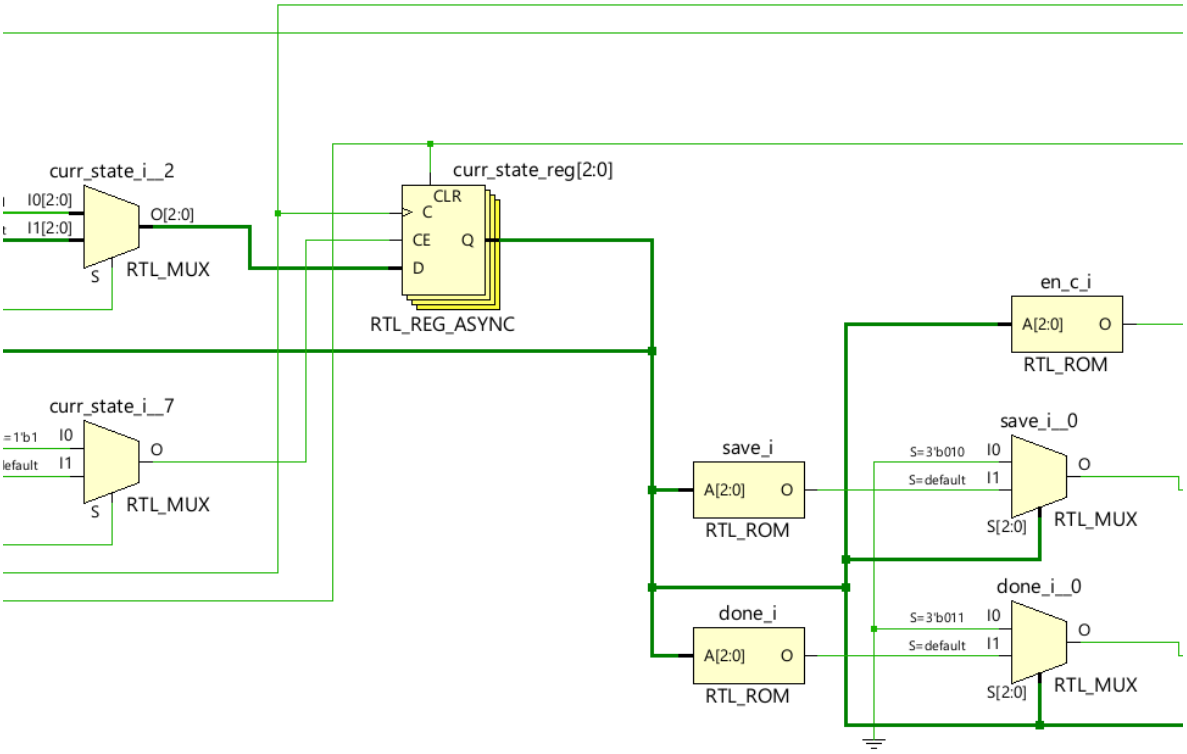
Si può notare come siano stati utilizzati otto Filp Flop D con clock, enable e clr condiviso ma ognuno con il proprio ingresso distinto, per poi mettere in AND ogni uscita di ogni FF con show_i.

3.1.3 Sintesi Demux



Il demux è stato sintetizzato con tre mux, si noti la presenza del caso di default.

3.1.4 Sintesi FSM



La FSM come ci si aspettava è stata implementata con tre FF per tenere traccia del current state e la rete combinatoria di output non è collegata agli ingressi, in quanto in una FSM di Moore le uscite dipendono solo dallo stato corrente, al contrario di una FSM di Mealy.
Si noti anche come l'uso della descrizione Behavioral con il costrutto if-elsif porti alla sintesi di mux.

3.2 Simulazioni

Le simulazioni sono state effettuate tenendo in considerazione il testbench fornito dai professori, pertanto i segnali di ingresso iniziano e terminano sul fronte di discesa del clock, questo anche per avere una più facile lettura da parte dell'essere umano. Il clock period usato è da 100ns come da specifica.
Si ricorda che la lettura di W, e di conseguenza anche di START, deve avvenire sul fronte di salita del clock come da specifica.
In ogni simulazione la memoria è riempita con valori unici per gli indirizzi usati, in modo da essere sicuri che il test bench non passi per una mera casualità.
Inoltre, ogni volta che il DONE passa ad 1, vengono controllati tutti i registri e non solo quelli cambiati, questo per accertarsi della persistenza dei dati e che non ci siano errori con i canali e con il circuito di gestione degli stessi(SIPO_CH + DEMUX).

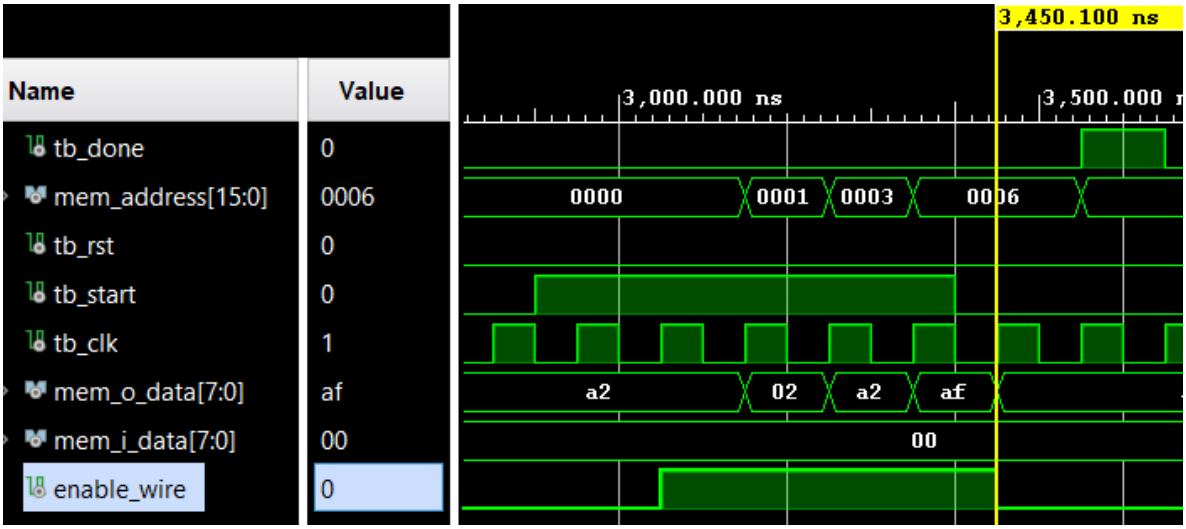
3.2.1 Caso generico

Il primo caso analizzato è un caso generico. In questo testbench sono stati inclusi divesi casi consecutivi generati casualmente.
In questo testbench il segnale W era a 0 quando START era a 0.
In questo testbench è stato incluso anche il testbench fronito dai professori.

L'obiettivo è controllare se di norma il modulo HW funzioni, le transizioni della FSM funzionino escludendo quelle del RESET e il riempimento dell'indirizzo con gli zeri funzioni.

3.2.2 Tutti 0 e tutti 1

Gli estremi di un insieme vengono considerati casi critici, da qui la necessità di analizzare i casi in cui la sequenza valida sull'ingresso W è costituita da bit tutti a 0 o tutti a 1.
Il testbench richiede anche il dover ignorare W sia in fase di WAIT (dopo che DONE è tornato a 0), sia agli estremi della sequenza dei casi limite. Ignorare gli estremi della sequenza significa essere sicuri dello stato nel quale la FSM si trova: es. con START=0011100 W=1100011, si ottiene come channel 00 e come idirizzo 0000000000000000 signifca essere sicuri che alla fine del primo 0 ci si trovi nello stato READ_CH_2, alla fine del secondo 0 ci si trovi nello stato READ_ADD e alla fine del terzo 0 nello stato READ_MEM. É stato anche usato l'enable della memoria per controllare se ci si trovasse correttamente nello stato READ_ADD, infatti può essere controllato l'inizio dopo due cicli di clock e la fine dopo un ciclo di clock, questo perché c'è un leggero ritardo dovuto alla rete combinatora delle uscite della FSM:



I casi critici di questo tipo sono i seguenti:

```
% START
"00001111111111111111110000000"
% W
"11110000000000000000001101000"

% START
"00000001111111111111111110000"
% W
"11010001111111111111111110000"

% START
"00001111000011110000"
% W
"00001111001100001111"

% START
"00001111111111111111110000000"
% W
"11110011111111111111110001000"

% START
"00001111111111111111110000000"
% W
"00001100000000000000001111000"
```

3.2.3 Enable SIPO ADDRESS

Questa categoia di casi di test mira a verificare che l'enable dei SIPO Address sia effettivamente funzionante e mira a completare il testbench precedente considerando come critico anche il caso in cui la sequenza valida sia costituita da soli 2bit.

```
% START
"0000110000000"
% W
"1011001111111"

% START
"0000110000000"
% W
"1011011111111"

% START
"0000110000"
% W
"0000101100"

% START
"00110000"
% W
"00110000"
```

Ogni canale è stato controllato su 2bit di sequenza.

3.2.4 Address testbench

Una possibilità potrebbe essere quella di aver analizzato solo casi di indirizzi palindromi o che lo scorrimento dei bit non funzioni correttamente, questo si risolve testanto pochi casi.

```
% START
"0011110000"
% W
"0011011100"

% START
"00111111111111111110000"
% W
"00001000000000000000000"

% START
"00111111111111111110000"
% W
"00101010010000000111100"
```

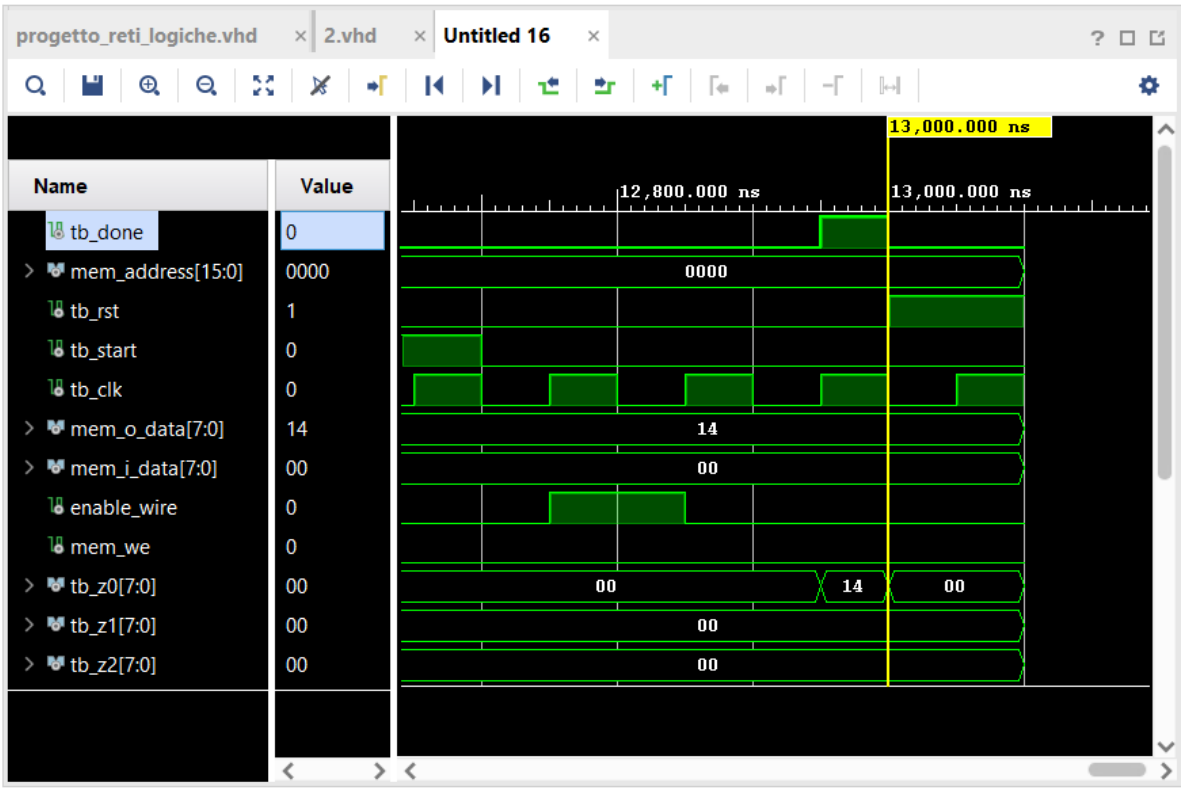
Il primo caso di test garantisce che l'indirizzo non sia palindromo e sia effettivamente letto come 0000000000000010.
Il secondo caso di test assicura che il primo bit sia effettivamente il più significativo e lo scorrimento del primo bit avvenga per tutti e sedici i Flip Flop. É assicurato che il bit giusto scorra perché è l'unico 1 della sequenza.
Il terzo caso di test unisce i primi due.

3.2.5 RESET testbench

Il segnale di RESET deve avere la precedenza su tutti gli altri segnali.
Per questo motivo ci si vuole accertare che il reset sia eseguito correttamente e per ogni stato, quindi

mischiando i precedenti casi critici e i casi generici, si è effettuato il reset sia nella fase di WAIT che in ogni possibile stato, verificando che tutti i canali tornino a 0, tranne uno che viene modificato(a giro) alla memorizzazione successiva.

Un comportamento interessante, ma voluto, avviene quando viene portato DONE a 1, ma a metà clock il segnale di RESET viene posto ad 1. Questo porta DONE a 0 perché avviene un cambio di stato e le uscite tornano a 0, sie per il cambio di stato, ma anche perché i registri PIPO vengono azzerati.



3.2.6 Segnali in successione

L'ultima categoria di casi critici è quella in cui i segnali vengono forniti ad 1 logico in immediata successione.
In particolare quando il segnale di START viene posto ad 1 immediatamente dopo che il segnale di RESET viene posto a 0 e quando il segnale di START viene posto ad 1 immediatamente dopo che il segnale di DONE è stato posto a 0.
Il primo caso in realtà potrebbe ricadere nella categoria precedente perché il segnale di reset ha priorità su tutto.
Il secondo caso vuole verificare che la specifica di poter dare una nuova sequenza subito dopo aver posto il segnale di DONE a 0 sia rispettata.
Un altro possibile problema associato potrebbe essere il corretto conteggio del 2bit di channel in quanto il primo stato è anche stato di WAIT.

4 Conclusioni

Il modulo HW è stato progettato rispettando tutte le specifiche, le quali sono state testate esaustivamente dai testbench precedentemente descritti.
La sintesi ha rispettato le aspettative nell'impiego di Flip Flop e l'assenza di Leach. Si è potuto notare la descrizione Behavioral in quanto le scelte nelle reti combinatorie sono state sintetizzate con dei MUX. Lo slack time è molto ampio, circa 97% del tempo di clock prestabilito e quindi nessuna preoccupazione viene sollevata dai ritardi delle reti combinatorie.

Il tempo di risposta del modulo è di due cicli di clock, il quale presumibilmente è il più basso raggiungibile in qunato limitato dal tempo di risposta della memoria.
Il numero di Filp Flop utilizzati è di 53 e si ritiene sia il minimo possibile, in quanto 50 sono necessari per la memorizzazzione dei dati in ingresso e in uscita, mentre gli altri 3 servono a sintetizzare la FSM con 5 stati.

Si spera che il lavoro svolto sia di Vostro gradimento e ci si scusa per la lunghezza della relazione dovuta anche al gran numero di immagini presenti.