Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

Progetto Di Reti Logiche

Anno Accademico 2022/2023

Prof. Fabio Salice

POLITECNICO
MILANO 1863

Fiore Donato, codice matricola: 958329

Fornara Alessandro, codice matricola: 955344

Indice

1. INTRODUZIONE

1.1 SCOPO PROGETTO

La specifica del progetto consiste nel leggere un input seriale suddiviso in due parti: i primi due bit identificano uno di quattro canali di uscita (Z0, Z1, Z2 e Z3), mentre i restanti 0-16 bit rappresentano un indirizzo di memoria di una RAM esterna. L'indirizzo contiene il dato, di lunghezza pari a 8 bit, che verrà poi scritto nel canale di uscita precedentemente individuato.
I contenuti dei quattro canali di uscita saranno visibili solo quando il segnale DONE sarà uguale a '1', mentre quando quest'ultimo è 0 le uscite mostreranno 0x00.

1.2 INTERFACCIA DEL COMPONENTE

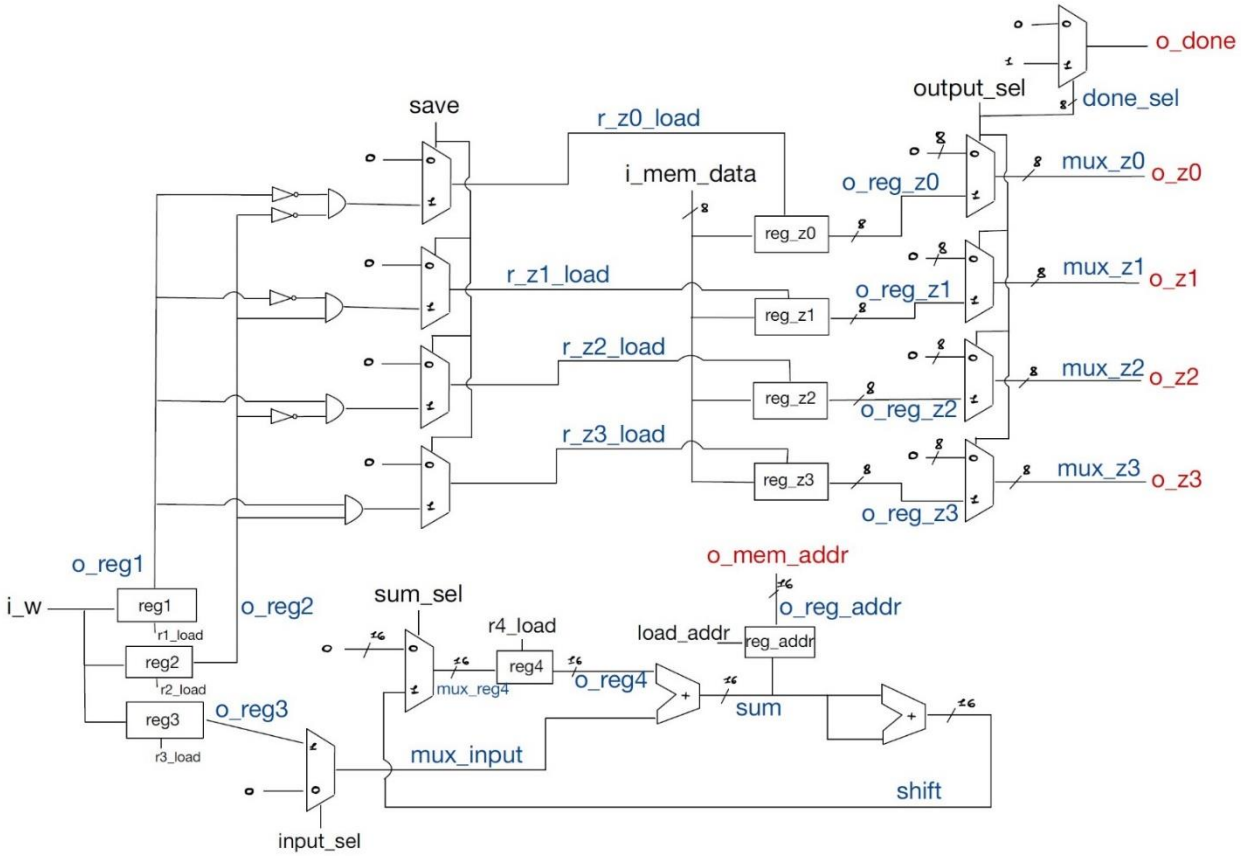
```
entity project_reti_logiche is
    port(
        i_clk : in std_logic;
        i_rst : in std_logic;
        i_start : in std_logic;
        i_w : in std_logic;
        o_z0 : out std_logic_vector(7 downto 0);
        o_z1 : out std_logic_vector(7 downto 0);
        o_z2 : out std_logic_vector(7 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* è il segnale di CLOCK in ingresso generato dal Test Bench;
- *i_rst* è il segnale di RESET che inizializza la macchina pronta per ricevere il 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.

2. ARCHITETTURA

2.1 DATAPATH



2.2 SEGNALI INTERNI

```
architecture Behavioral of datapath is
--Segnali Datapath
signal o_reg1 : STD_LOGIC := '0';
signal o_reg2 : STD_LOGIC := '0';
signal o_reg3 : STD_LOGIC := '0';
signal o_reg4 : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
signal mux_input : STD_LOGIC;
signal mux_reg4 : STD_LOGIC_VECTOR (15 downto 0);
signal sum : STD_LOGIC_VECTOR (15 downto 0);
signal shift : STD_LOGIC_VECTOR (15 downto 0);
signal o_reg_addr : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
signal r_z0_load : STD_LOGIC;
signal r_z1_load : STD_LOGIC;
signal r_z2_load : STD_LOGIC;
signal r_z3_load : STD_LOGIC;
signal o_reg_z0 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal o_reg_z1 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal o_reg_z2 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal o_reg_z3 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal mux_z0 : STD_LOGIC_VECTOR ( 7 downto 0);
signal mux_z1 : STD_LOGIC_VECTOR ( 7 downto 0);
signal mux_z2 : STD_LOGIC_VECTOR ( 7 downto 0);
signal mux_z3 : STD_LOGIC_VECTOR ( 7 downto 0);
signal done_sel : STD_LOGIC_VECTOR (7 downto 0);
```

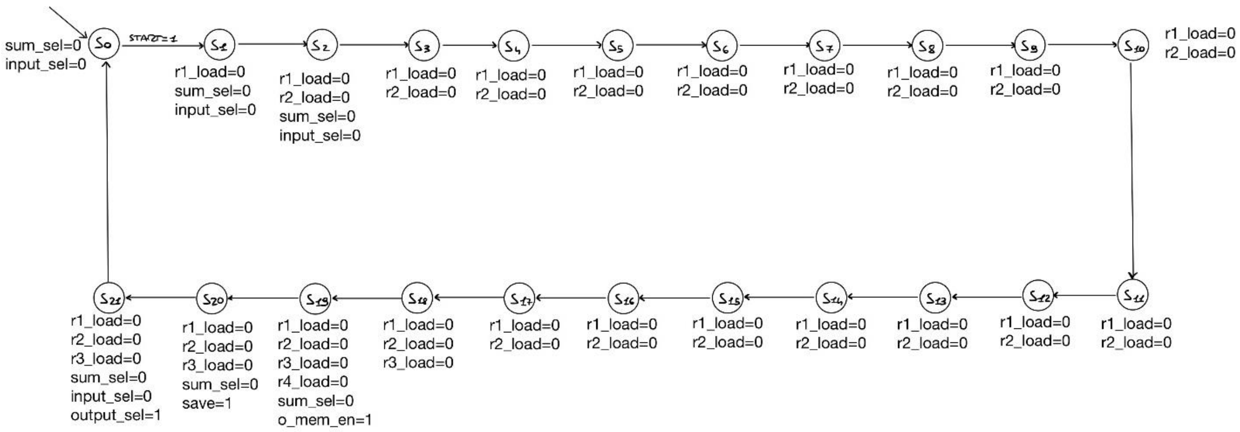
I segnali del tipo *o_reg_x*, *con* $x \in \{1; 2; 3; 4; z0; z1; z2; z3\}$, sono le uscite dei rispettivi registri.

In particolare:

- I registri reg1 e reg2 contengono i primi 2 bit letti da input che indicano il canale di uscita **Z**;
- Il registro reg3 contiene il bit che verrà in seguito sommato per calcolare l'indirizzo finale della memoria;

- Il registro reg4 è inizializzato a 0x0000 e in seguito verrà utilizzato per il calcolo dell'indirizzo di memoria;
- Il segnale *mux_input* è l'uscita del multiplexer controllato da "*input_sel*", può essere uguale al valore di default '0' oppure al bit di ingresso letto contenuto in reg3;
- Il segnale *mux_reg4* è l'uscita del multiplexer controllato da "*sum_sel*", può essere uguale al valore di default '0' oppure all'indirizzo di memoria provvisorio shiftato;
- Il segnale *sum* è l'indirizzo di memoria calcolato ogni ciclo di clock;
- Il segnale *shift* è uguale al segnale sum, ma shiftato di uno a sinistra;
- Il segnale *o_reg_addr* è l'indirizzo di memoria calcolato leggendo l'input;
- I segnali *r_z0_load*, *r_z1_load*, *r_z2_load* e *r_z3_load* sono le uscite dei multiplexer controllati da "*save*" e gestiscono il salvataggio di dati nei rispettivi registri *reg_z0*, *reg_z1*, *reg_z2* e *reg_z3*;
- I segnali *o_reg_z0*, *o_reg_z1*, *o_reg_z2* e *o_reg_z3* sono le eventuali uscite nel caso in cui o_done fosse 1;
- I segnali *mux_z0*, *mux_z1*, *mux_z2* e *mux_z3* sono le uscite dei multiplexer controllati da "*output_sel*" e coincidono rispettivamente con le uscite *o_Z0, o_Z1, o_Z2* e *o_Z3*;
- Il segnale *done_sel* stabilisce il valore dell'output *o_done*.

2.3 MACCHINA A STATI

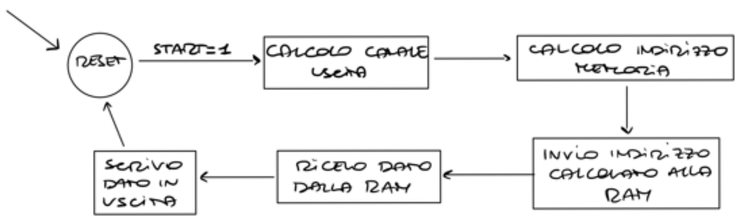


Sono omesse le transizioni verso S0 da tutti gli stati quando "i_rst = 1".

Gli stati S2-->S18 dispongono di una transizione verso lo stato S19 quando i_start diventa 0.

Qui sono riportati i valori di default dei seguenti segnali:

```
r1_load<='1';
r2_load<='1';
r3_load<='1';
r4_load<='1';
sum_sel<='1';
output_sel<='0';
input_sel<='1';
save<='0';
load_addr<='1';
o_mem_en<='0';
o_mem_we<='0';
```



Versione semplificata della macchina a stati.

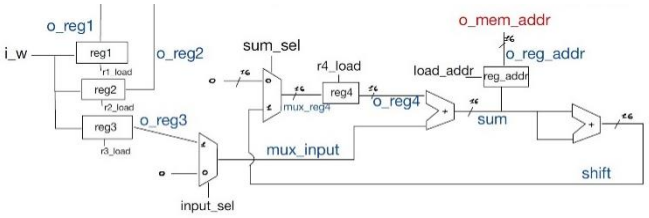
Descrizione macchina a stati:

- Lo stato **S0** è lo stato di "*<u>reset</u>*" della macchina e rappresenta lo stato iniziale per ogni lettura di indirizzo;

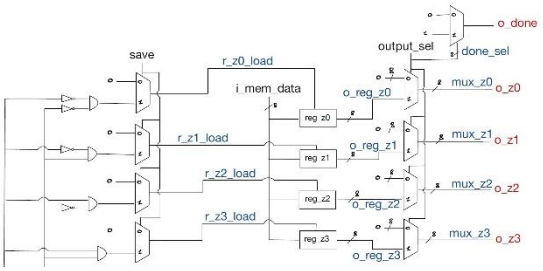
- Negli stati **S1** e **S2** avviene la lettura dei bit che saranno salvati nei registri reg1 e reg2. Essi indicano il canale di uscita, *"Calcolo canale uscita"*;
- Negli stati **S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17 e S18** avviene il calcolo dell'indirizzo di memoria dal quale leggeremo il dato che verrà scritto nel canale di uscita individuato negli stati precedenti, *"Calcolo indirizzo memoria"*;
- Nello stato **S19** l'indirizzo calcolato viene dato in ingresso alla RAM, *"Calcolo indirizzo memoria"*;
- Nello stato **S20** la RAM ci restituisce il dato che si trova nell'indirizzo specificato, *"Ricevo dato dalla RAM"*;
- Nello stato **S21** il dato ricevuto dalla RAM viene salvato e scritto nel canale di uscita. In contemporanea l'uscita *o_done* assume valore 1 per un ciclo di clock, *"Scrivo dato in uscita"*.

2.4 SPIEGAZIONE GENERALE DEL CIRCUITO

Nello stato di reset **S0**, i registri reg1, reg2, reg3, reg4 e reg_addr hanno il proprio segnale di load impostato a 1 e tutti i multiplexer hanno come uscita il valore '0' oppure 0x00. Nel momento in cui *i_start* diventa 1, il bit letto viene salvato in reg1 e *r1_load* viene immediatamente posto a '0', in modo tale da non perdere quel bit. La stessa cosa avverrà nel ciclo successivo in cui verrà letto il secondo bit che sarà salvato in reg2 con *r2_load* che diventerà '0'.



Poiché l'input è seriale, la prima parte del circuito si occupa di salvare correttamente l'indirizzo di memoria in un registro. Supponiamo che *i_start* rimanga alto e venga dato in ingresso qualche bit dell'indirizzo: il terzo bit letto (primo bit dell'indirizzo) viene salvato in reg3 e i segnali *sum_sel* e *input_sel* vengono posti a '1'. In questo modo, mux*_input* diventerà uguale a *o_reg3* e sarà sommato al contenuto di *o_reg4*, pari a 0x00. Il risultato *sum* viene sommato a se stesso per effettuare uno shift a sinistra dell'indirizzo e, nel ciclo di clock successivo, sarà salvato in reg_addr. La stessa cosa vale per il segnale shift che passerà per il multiplexer (*sum_sel* è '1') e sarà salvato in reg4 al prossimo ciclo di clock. Una volta letto il quarto bit in ingresso, quest'ultimo sarà sommato con il valore salvato in reg4, ovvero il valore di shift nel ciclo di clock precedente. Da qui in poi il processo verrà iterato fino a quando start non diventerà 0; il risultato finale si troverà in reg_addr.



Il processo appena descritto avviene tra gli stati **S3** e **S18**, ma se l'ingresso dovesse fornire meno di 16 bit per l'indirizzo di memoria, la macchina a stati si sposterà su **S19** indipendentemente dallo stato in cui si trova. Successivamente il segnale *enable* della memoria viene posto a 1 e nel ciclo di clock successivo, il circuito riceverà dalla RAM il dato. A quel punto il segnale *save* sarà impostato a 1 e i multiplexer controllati da quest'ultimo avranno come uscita le seguenti funzioni:

- r_z0_load <= not(o_reg1) and not(o_reg2);
- r_z1_load <= not(o_reg1) and o_reg2;
- r_z2_load <= o_reg1 and not(o_reg2);
- r_z3_load <= o_reg1 and o_reg2;

Grazie a questi segnali solo uno tra reg_z0, reg_z1, reg_z2 e reg_z3 avrà load=1 e sarà il registro rappresentato dal valore *"o_reg1" "o_reg2"* (00 -> *r_z0_load*='1', 01 -> *r_z1_load*='1', ...). Il dato

sarà quindi salvato in un registro nel ciclo successivo e *output_sel* sarà posto a '1', rendendo visibili in uscita i contenuti dei registri e portando *o_done* a '1' grazie ad un multiplexer.
Infine la macchina a stati tornerà nello stato di reset **S0** portando così *output_sel* a '0', e di conseguenza *o_done* a '0' e le quattro uscite a 0x00. In questo stato il circuito rimane in attesa che il segnale *i_start* venga posto nuovamente a 1 per ricominciare il suo ciclo.

2.5 IL PROBLEMA DEL "*DONE*"

Come si può notare dal disegno e dal codice, il segnale *done_sel*, direttamente collegato ad *output_sel*, ha una lunghezza di 8 bit e non di 1. Durante le prime fasi di progettazione abbiamo riscontrato un problema riguardo il segnale o_done. Quando il segnale *o_done* tornava al valore '0', i segnali in uscita (*o_z0, o_z1, o_z2 e o_z3*) non venivano posti immediatamente al valore 0x00. Abbiamo ipotizzato che il motivo fosse legato al tempo, ovvero che un segnale da 8 bit fosse più lento di un segnale da 1 bit nel subire un cambiamento.
La prima soluzione che abbiamo trovato è stata quella di utilizzare delle porte logiche OR tra le uscite dei multiplexer per controllare il multiplexer di *o_done*, in modo tale che quest'ultimo fosse posto a '0' solo quando il risultato di tutti gli OR fosse 0x00. Grazie a questa soluzione eravamo riusciti a risolvere solo parzialmente il problema, in quanto avevamo inavvertitamente creato un bug, ovvero che, se il primo dato ricevuto dalla RAM fosse stato 0x00, il segnale *o_done* non sarebbe diventato '1' nel momento giusto. Alla fine ci siamo resi conto che potevamo semplicemente estendere *output_sel* con 7 zeri nelle posizioni più significative. In questo modo, al cambiamento di *output_sel* (da 1 bit) corrisponde un cambiamento di *done_sel* (da 8 bit), risolvendo così il problema.

3. RISULTATI SPERIMENTALI

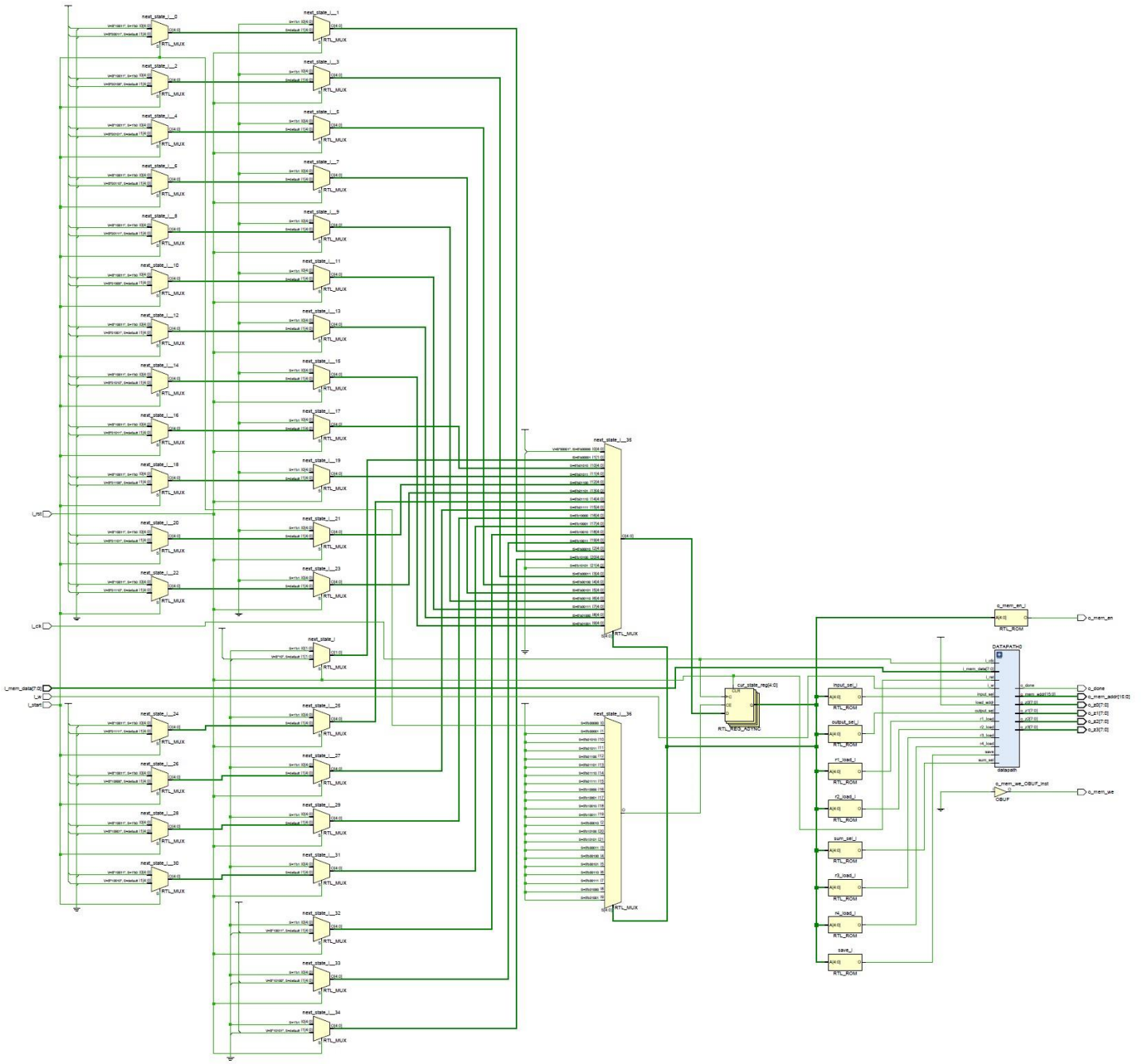
3.1 COMPONENTI CIRCUITO SINTETIZZATO

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	85	0	134600	0.06
LUT as Logic	85	0	134600	0.06
LUT as Memory	0	0	46200	0.00
Slice Registers	88	0	269200	0.03
Register as Flip Flop	88	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

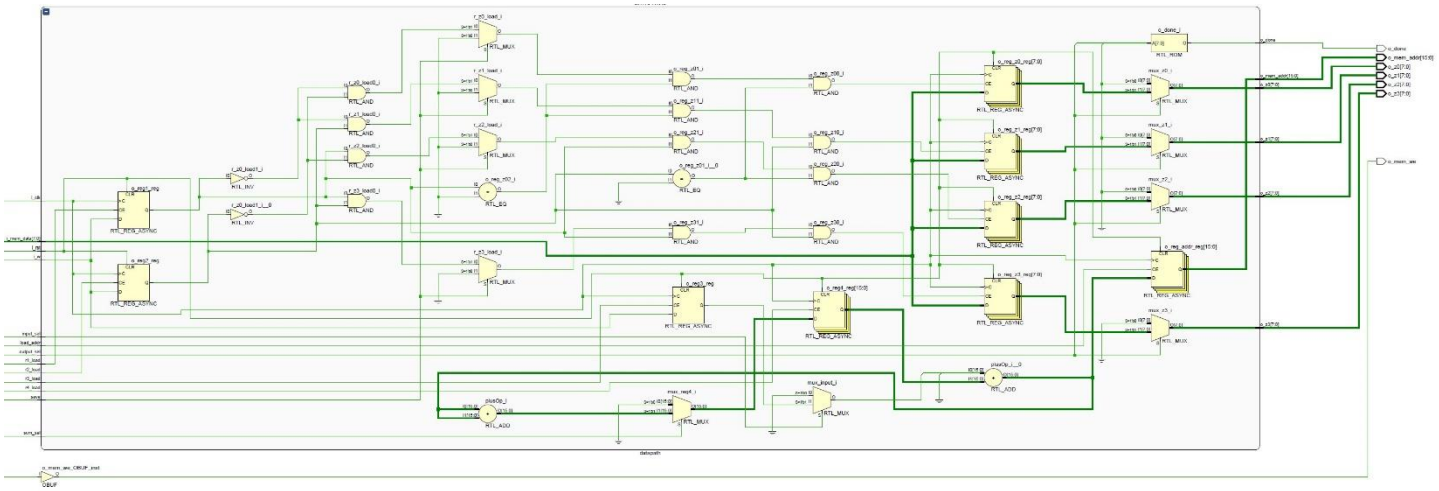
Il circuito è sintetizzato tramite l'utilizzo di 85 LUT, 88 Flip Flop e 0 Latch.

In elettronica digitale, il latch (letteralmente "serratura", "chiavistello") è un circuito elettronico bistabile, caratterizzato quindi da almeno due stati stabili, in grado di memorizzare un bit di informazione nei sistemi a logica sequenziale asincrona.
Il latch è un elemento logico che può campionare e contenere (*sample & hold*) un valore binario, molto simile ad un flip-flop. Diversamente dal flip-flop, che è edge-triggered, il latch è level-triggered.
I latch si generano quando si crea un processo combinatorio o un'assegnazione condizionale o un blocco combinatorio con un'uscita che non è assegnata in tutte le possibili condizioni di input. Questo crea ciò che è noto come *assegnazione incompleta* da parte degli strumenti di sintesi. L'assegnazione dell'output non è completa in tutte le possibilità di input.

3.2 SCHEMA CIRCUITO SINTETIZZATO



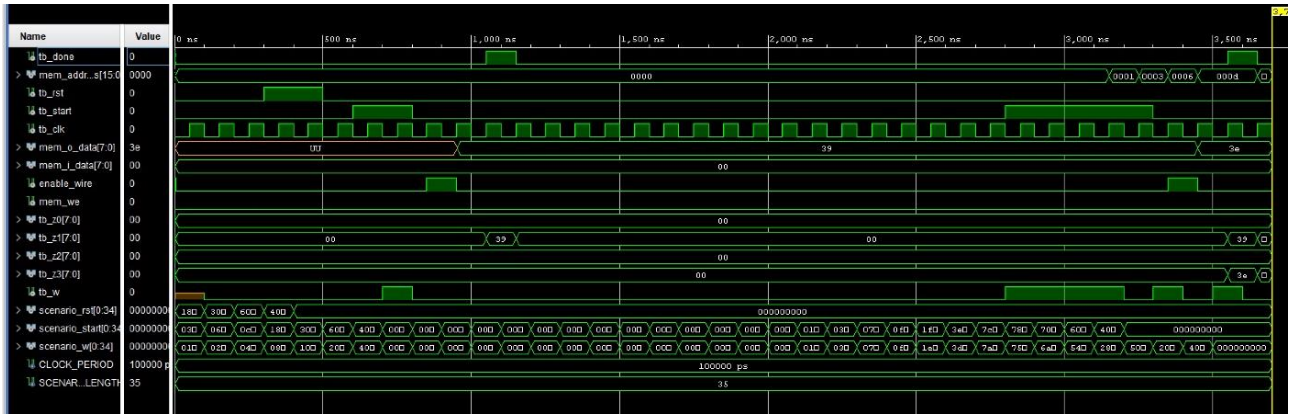
Datapath sintetizzato (zoom del rettangolo azzurro nel disegno qui sopra)



3.3 CASI DI TEST

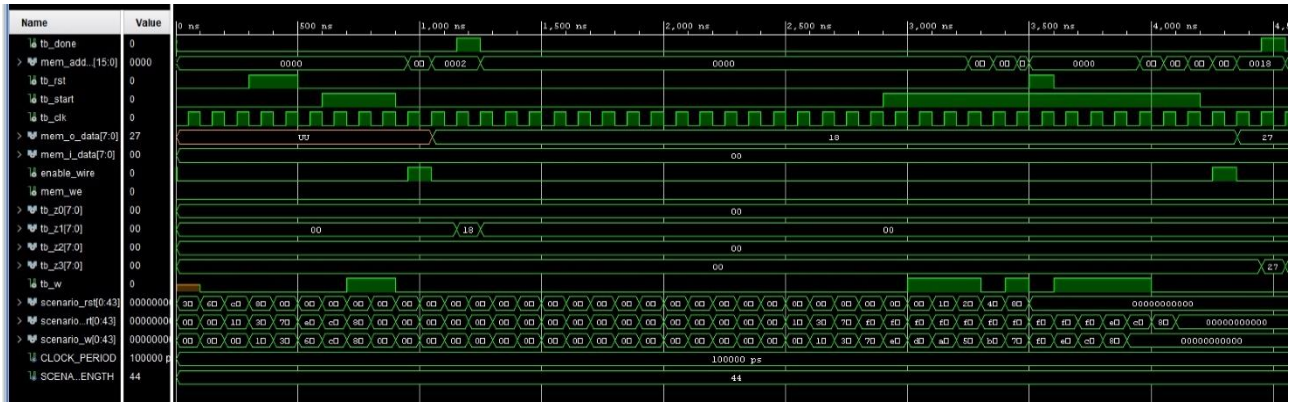
Qui sotto sono riportati i risultati in Post-Synthesis Functional Simulation di alcuni test creati da noi per verificare il corretto funzionamento del circuito.

a. 0 bit di indirizzo



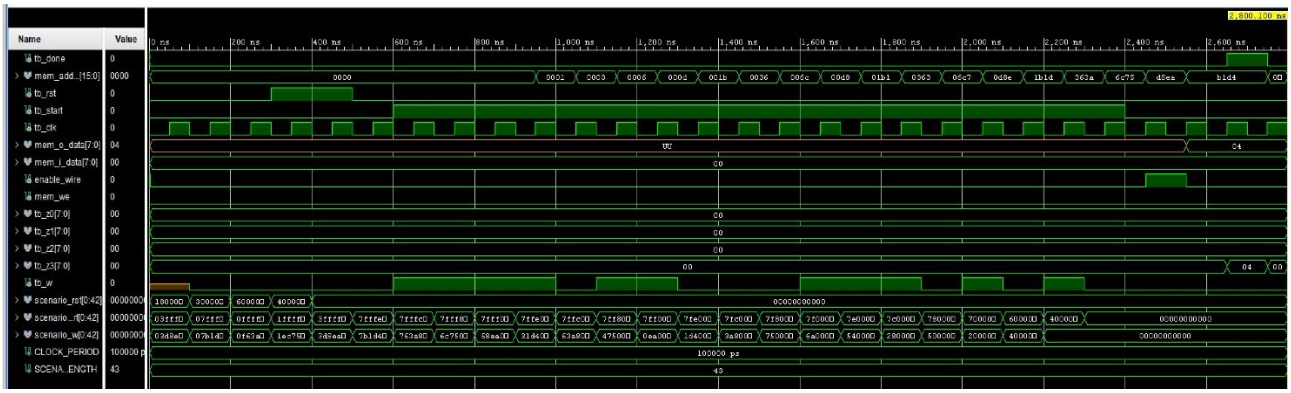
In questo test affrontiamo il caso limite dovuto alla lunghezza nulla dell'indirizzo di memoria ricevuto in input. Il segnale *start* ha valore 1 per 2 cicli di clock, quindi leggiamo solo i 2 bit che indicano il canale di uscita. L'indirizzo di memoria, avendo lunghezza nulla, lo si pone uguale all'indirizzo 0x0000.

b. Reset = 1, Start = 1.



In questo test controlliamo il corretto funzionamento del segnale *i_rst*. Quando abbiamo $i\_rst = 1$, tutte le uscite e i registri vengono re-inizializzati. In questo caso *i_rst* sale durante la 2° lettura (*start = 1*); si può notare come le uscite siano correttamente inizializzate e assumano i corretti valori finali.

c. 16 bit di indirizzo

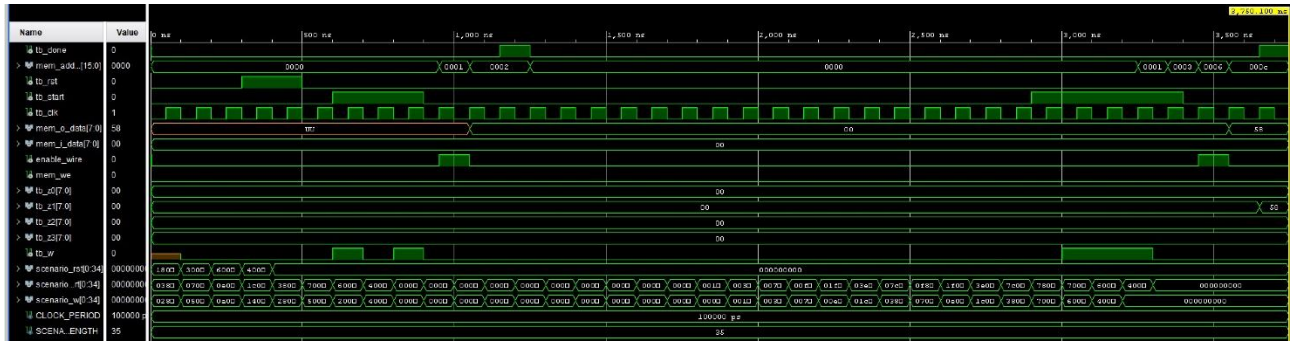


In questo test affrontiamo il caso limite dovuto alla lunghezza dell'indirizzo di memoria ricevuto in input pari a 16 bit (il cui valore è xD8EA nell'immagine riportata).

d. Stress Test. Input di 17 indirizzi di memoria con 17 lunghezze diverse (0-16 bit)

Questo test è stato realizzato per provare ogni possibile lunghezza dell'indirizzo di memoria in input. Il circuito riceve in ingresso 17 indirizzi, il primo da 0 bit, il secondo da 1 bit e così via fino a 16.

e. Dato letto dall'indirizzo di memoria pari a 0x00



Questo test affronta un caso limite un po' diverso dai precedenti, in quanto non dipende dall'input seriale. Può succedere che nell'indirizzo di memoria ricevuto il dato abbia valore nullo (0x00). Questo valore avrebbe potuto crea problemi se il segnale *o_done* fosse dipeso direttamente dal valore delle uscite o_z0, o_z1, o_z2 e o_z3.

4. CONCLUSIONI

Riteniamo che siano stati raggiunti gli obiettivi richiesti per la esatta e precisa elaborazione del progetto assegnato, il tutto dopo aver seguito le varie fasi che ci hanno permesso di pervenire al report finale.

Nello specifico, dopo la prima fase del problem finding, con la relativa individuazione di potenziali problemi e lettura dati, siamo passati al setting ed abbiamo sintetizzato, definito e descritto i problemi. Fatto ciò li abbiamo analizzati e scomposti in una serie di problemi secondari più piccoli (analisi scompositiva). Ciò ci ha permesso di affrontare i problemi principali in modo logico e di evidenziare i fattori critici.

Abbiamo proseguito con il riepilogo (decision making) e sintesi delle possibili azioni da intraprendere per poi studiare la soluzione possibile con la individuazione e rimozione delle cause al fine di impedire che si potesse ripresentare in futuro.

Ritieniamo che l'architettura abbia rispecchiato quanto specificamente richiesto nel progetto; inoltre lavorare su questo progetto ci ha permesso di ampliare conoscenze, abilità e competenze sulla tematica assegnata.