PROVA FINALE:

PROGETTO DI RETI LOGICHE 2022/2023

Professore di riferimento: Fabio Salice

Studente: Alessandro Lucci Cod. Pers: 10685820

Matricola: 911121

1. Introduzione

Specifica generale:

La specifica del progetto di quest'anno accademico richiede di implementare un modulo HW in VHDL che si interfacci e comunichi con una memoria simulata.

Il modulo riceve in ingresso due segnali **START e W**, il segnale w è una sequenza di massimo 18 bit di cui i primi due indicano l'ID di una delle 4 uscite del modulo, i bit successivi, che possono variare da 0 ad un massimo di 16 bit, indicano un indirizzo di memoria che verrà inviato alla memoria, la quale risponderà con i dati corrispondenti all'indirizzo di memoria a seguito di una richiesta di lettura da parte del modulo. Questi dati ricevuti dalla memoria verranno inseriti nell'uscita indicata dai primi due bit. Al momento di generare i segnali di output l'uscita presa in considerazione presenterà i dati ricevuti mentre le altre i dati precedentemente da loro esposti.

Il segnale **START** scandisce il periodo di tempo in cui il segnale **W** dovrà essere letto.

I bit vengono letti in corrispondenza del fronte di salita di un segnale di clock **CLK** ed il modulo può essere riportato allo stato iniziale in qualunque momento da un segnale di reset **RST**.

Interfaccia del componente:

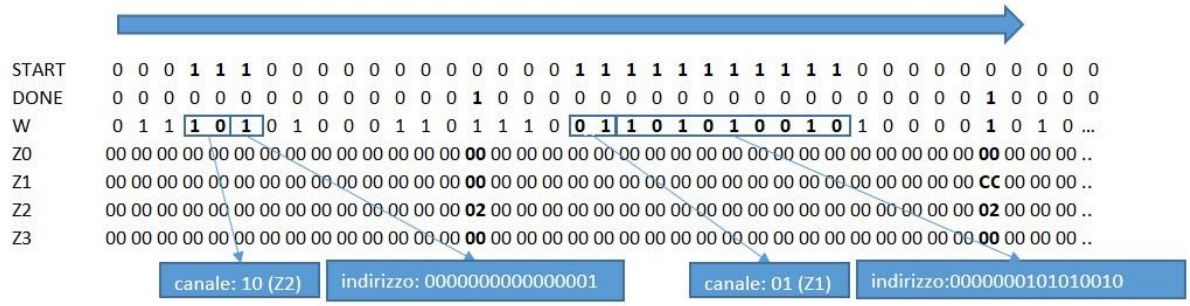
È riportata in seguito l'interfaccia del componente in linguaggio VHDL:

```
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 segnali i_clk, i_rst, i_start e i_w sono i segnali in input corrispondenti a **CLK, RST, START** e **W** introdotti precedentemente;
- I segnali o_z0, o_z1, o_z2, o_z3 sono i segnali in output corrispondenti alle 4 uscite del componente;
- Il segnale o_done è un segnale che indica la fine di un ciclo di elaborazione e viene portato ad 1 quando vengono aggiornate le uscite del modulo;
- Il segnale o_mem_addr è il vettore in output che contiene l'indirizzo da inviare alla memoria;
- Il segnale i_mem_data è il vettore in input proveniente dalla memoria che contiene i dati corrispondenti all'indirizzo inviato;
- Il segnale o_mem_en è il segnale che il modulo invia alla memoria per poter comunicare;
- Il segnale o_mem_we è il segnale che il modulo invia alla memoria per poterci scrivere, in questo progetto non è necessario;

Esempio di funzionamento riportato sulla specifica:



		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**
02	045B	A1	00	CC	A1	00	**1101101000101**
00	0751	7E	7E	CC	A1	00	**1000101011100**
00	07B2	69	69	CC	A1	00	**0100110111100**
02	0A58	7B	69	CC	7B	00	**00011010010101**
00	0B1C	AC	AC	CC	7B	00	**00111000110100**
03	0DFC	EA	AC	CC	7B	EA	**00111111101111**
00	1025	03	03	CC	7B	EA	**101001000000100**
01	1335	D8	03	D8	7B	EA	**101011001100110**
02	1605	4F	03	D8	4F	EA	**101000000110101**
…	…	…	…	…	…	…	…

2.Architettura

Per implementare il modulo richiesto ho implementato il datapath rinominato MemAnalizer ed i seguenti segnali:

```
--Segnali Macchina a stati
signal set0 : std_logic; --Segnale inizio nuovo ciclo
signal z_load : std_logic; --Segnale aggiornamento uscite
signal zp_load : std_logic; --Segnale aggiornamento registri uscite
signal w_load :  std_logic; --Segnale lettura bit0 ID uscita
signal w_send : std_logic; --Segnale invio indirizzo a mem


--Segnali MemAnalizer
signal    zx0 :  std_logic; --Bit0 ID uscita
signal    zx1 :  std_logic; --Bit1 ID uscita
signal    raddr :  std_logic_vector(15 downto 0); --Registro indirizzo di mem

signal    z0prec :  std_logic_vector(7 downto 0); --Registro uscita z0
signal    z1prec :  std_logic_vector(7 downto 0); --Registro uscita z1
signal    z2prec :  std_logic_vector(7 downto 0); --Registro uscita z2
signal    z3prec :  std_logic_vector(7 downto 0); --Registro uscita z3
```

In particolare:

- Il segnale set0 indica l'inizio di un nuovo ciclo di elaborazione e viene utilizzato per riportare le uscite ed i segnali/registri temporanei a valori di default;
- Il segnale z_load è il segnale di aggiornamento delle uscite a fine elaborazione;
- Il segnale zp_load serve ad aggiornare i registri z0prec – z3prec in cui sono salvati i valori precedenti delle uscite;
- Il segnale w_load indica il momento di inizio lettura dell'indirizzo contenuto nell'input i_w ed il suo caricamento nel registro raddr;
- I segnali zx0 e zx1 rappresentano i primi due bit ricevuti in i_w ed indentificano l'ID dell'uscita che verrà aggiornata;

I segnali hanno i seguenti valori di default:

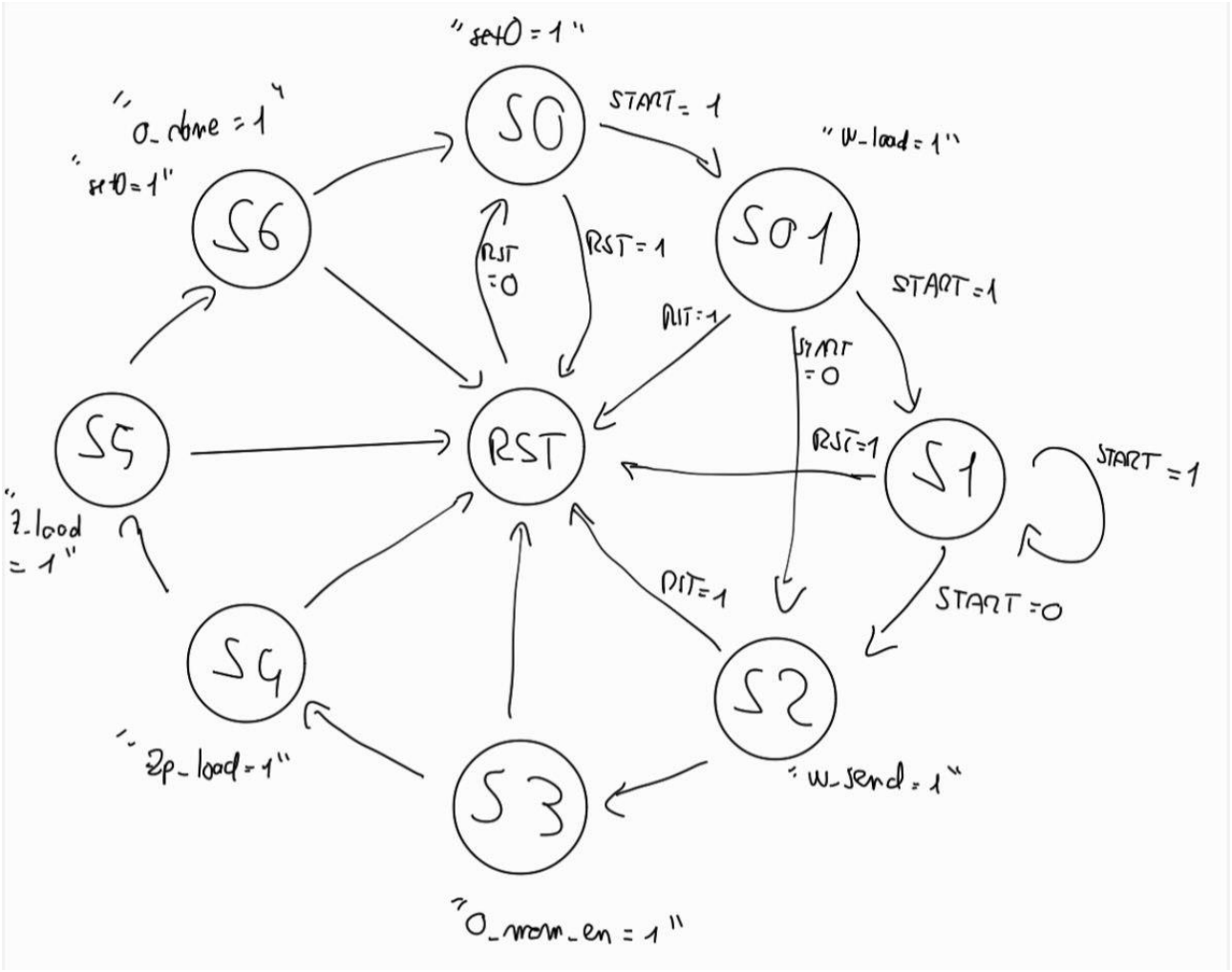
```
w_send <='0';
w_load <='0';
z_load <='0';
zp_load<='0';

o_mem_en <= '0';
o_mem_we <= '0';
o_done<='0';

set0<='0';
```

```
o_z0 <= "00000000";
o_z1 <= "00000000";
o_z2 <= "00000000";
o_z3 <= "00000000";
zx1 <= 'U';
zx0 <= 'U';
raddr <= "0000000000000000";
o_mem_addr<="0000000000000000";
```

```
z0prec <= "00000000";
z1prec <= "00000000";
z2prec <= "00000000";
z3prec <= "00000000";
```

Macchina a stati:

Grafo:



Descrizione stati:

State	New Encoding
rst	0000
s0	0001
s01	0010
s1	0011
s2	0100
s3	0101
s4	0110
s5	0111
s6	1000

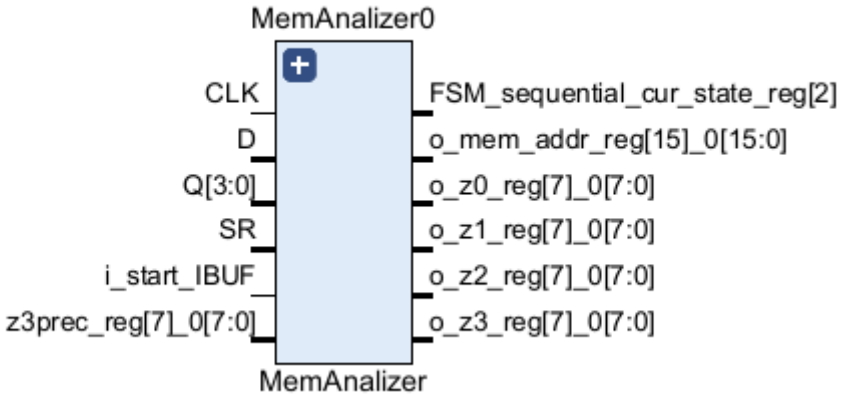
- **RST**: stato di reset, accessibile da qualunque stato quando **i_rst** = 1, porta tutti i valori a valori di default, quando **i_rst**=0 si passa a S0;
- **S0**: stato di attesa del segnale di start, porta tutti i valori temporanei a valori di default tranne i registri **z0prec-z3prec**, quando **i_start** = 1 legge il primo bit di **i_w** e lo inserisce in **zx1**;
- **S01**: avvio di un ciclo di elaborazione, legge il secondo bit di **i_w** e lo inserisce in **zx0**;
- **S1**: stato di lettura bit dell'indirizzo di memoria da **i_w** e registrazione in **raddr**;

- **S2**: invio indirizzo contenuto in **raddr** alla memoria tramite **o_mem_addr**;
- **S3**: invio segnale **o_mem_en** per comunicare alla memoria che il componente è pronto a ricevere i dati corrispondenti all'indirizzo inviato;
- **S4**: ricezione dati dalla memoria che vengono registrati nel registro **zXprec** corrispondente all'uscita letta dai primi due bit;
- **S5**: aggiornamento uscite;
- **S6**: attivazione segnale **o_done** in contemporanea all'aggiornamento delle uscite, porta tutti i valori temporanei a valori di default tranne i registri **z0prec-z3prec**;

MemAnalizer:

Il modulo MemAnalizer è una collezione di processi che implementa la macchina a stati sopra descritta: a seconda dei segnali ricevuti, svolge le operazioni dello stato corrispondente.

I segnali di input e output che gestisce sono qui segnati:



3.Risultati sperimentali

Report di Sintesi:

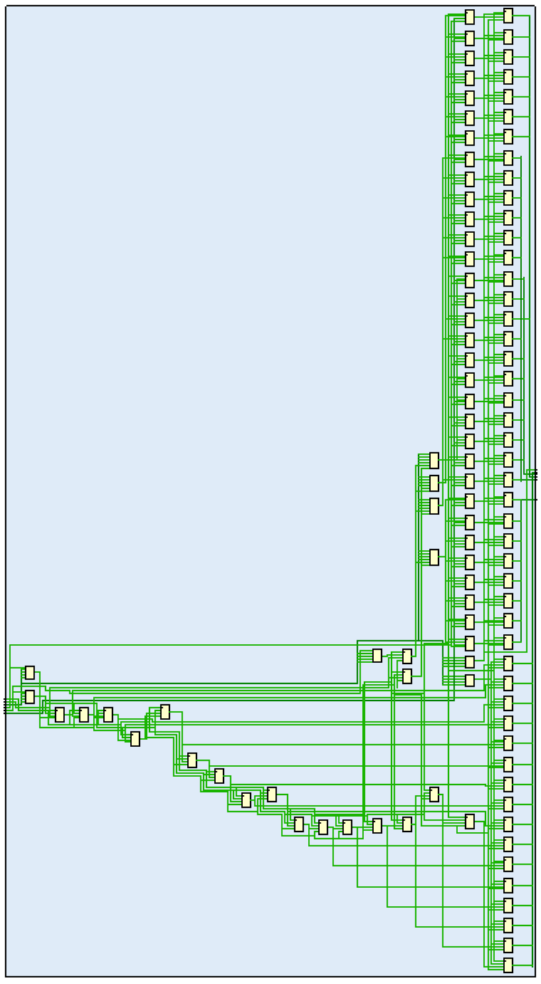
Il componente risulta correttamente sintetizzabile ed implementabile secondo la seguente tabella di Slice logic ricavata dal software Vivado:

1. Slice Logic

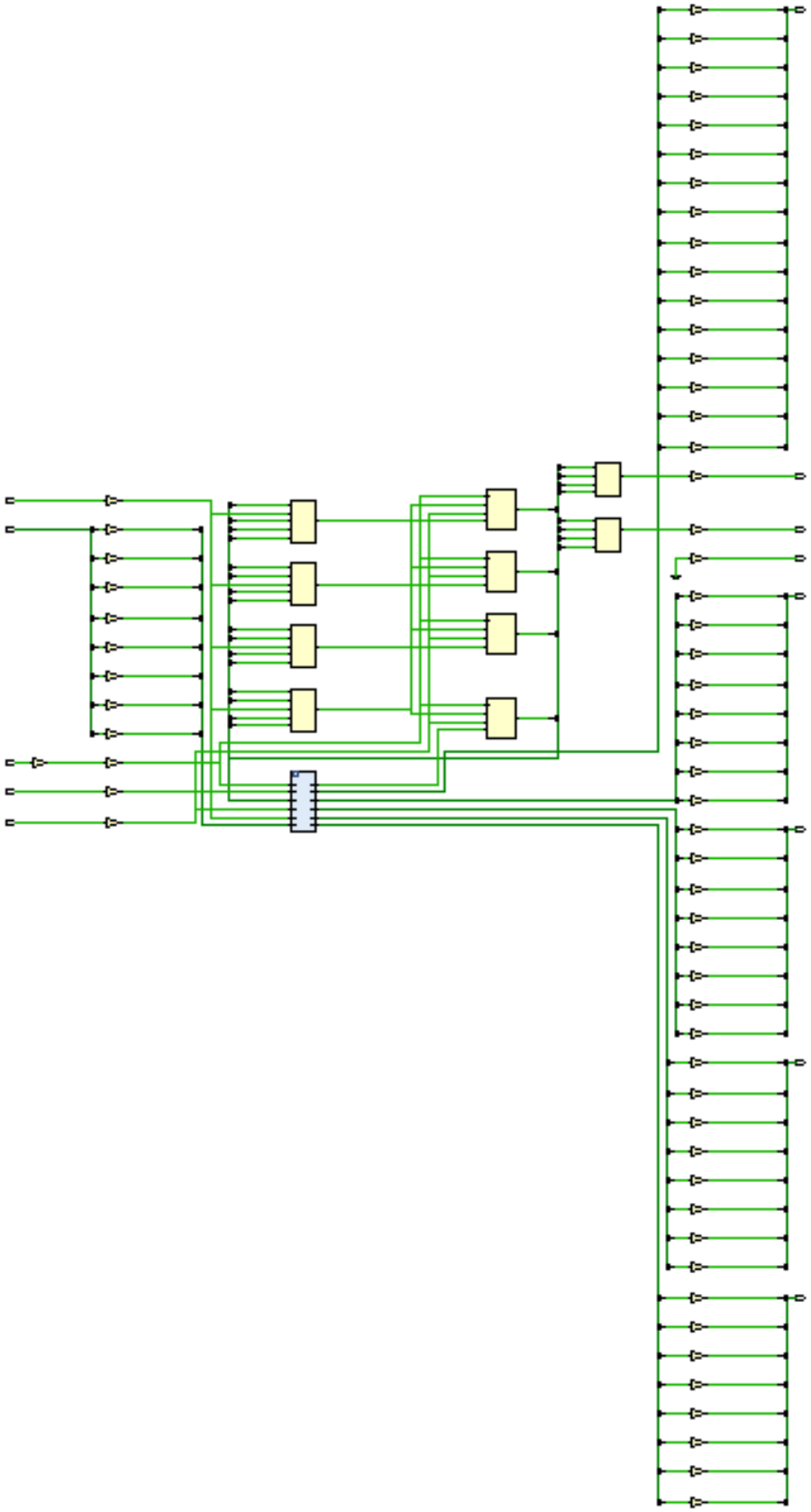
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	13	0	0	134600	<0.01
LUT as Logic	13	0	0	134600	<0.01
LUT as Memory	0	0	0	46200	0.00
Slice Registers	102	0	0	269200	0.04
Register as Flip Flop	102	0	0	269200	0.04
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

I design di sintesi ricavati sono i seguenti:

Design MemAnalizer:



Design componente completo con MemAnalizer compatto(componente celeste):



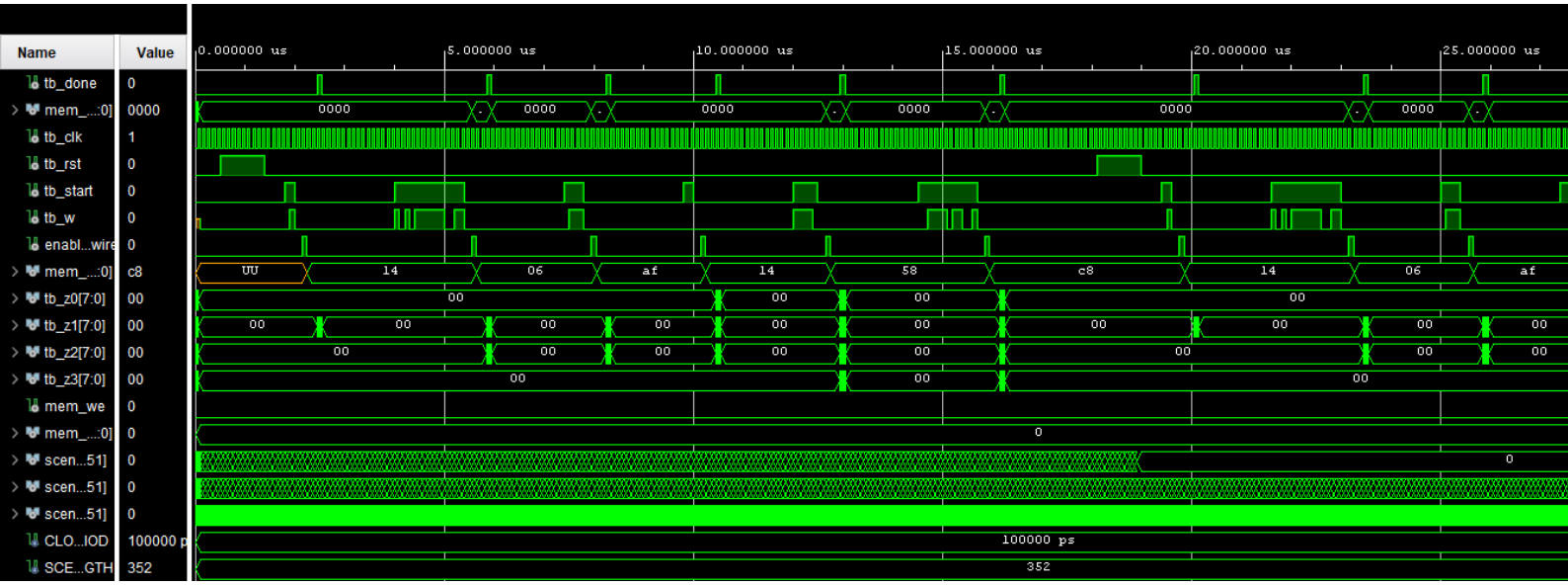
Risultati Simulazioni:

Tutte le Testbench a cui è stato sottoposto il componente danno in input diversi segnali **W, START, RESET** e diversi dati corrispondenti a diversi indirizzi di memoria. Ogni Testbench inizia con un segnale di **RESET** e controlla che ci siano gli output esatti alle uscite esatte al momento in cui viene dato il segnale **DONE** che deve essere dato entro 20 cicli di clock dal primo ciclo in cui **START** = 1.

In ogni simulazione il segnale **o_mem_we** sarà costante a 0 poiché non è necessario scrivere in memoria.

Ogni simulazione presenta casi particolari:

- **tb_1:** durata totale = 33.9µs

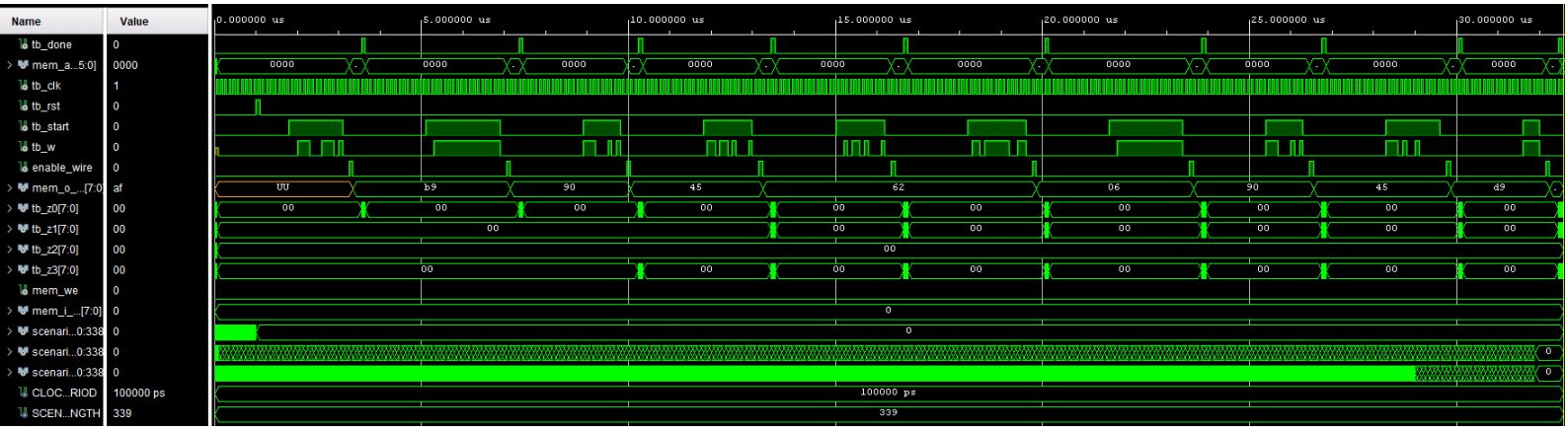


In questa simulazione sono presenti 2 segnali di **RESET**: uno all'inizio come in tutte le altre ed uno a 18µs dall'inizio;

Alcuni segnali **START** durano solamente 2 bit permettendo solo di identificare l'uscita **zX** che, non essendo comunicato alcun indirizzo, si aggiornerà al valore restituito dall'indirizzo di default "0000000000000000";

Vengono testate tutte le uscite con diversi dati di memoria;

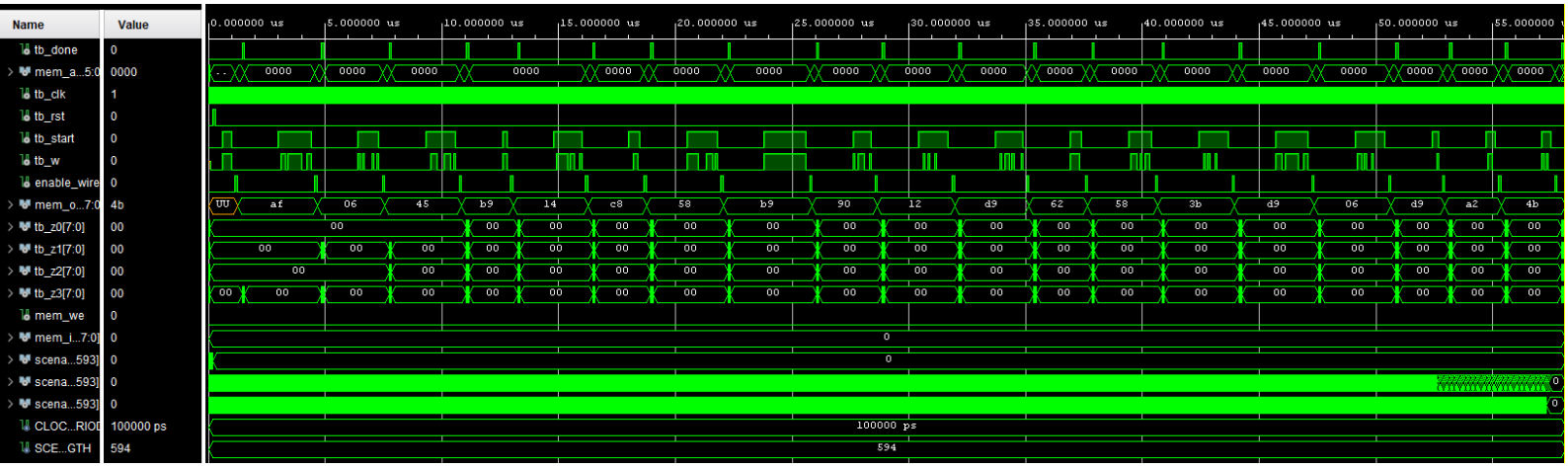
- **tb_2**: durata totale = 32.6µs



I flussi di bit in ingresso sono numerosi e lunghi avvicinandosi quasi sempre ai 18 bit;

L'uscita **z2** viene tenuta costante a valore di default 0 mentre tutte le altre vengono aggiornate;

- **tb_3**: durata totale = 58.1µs

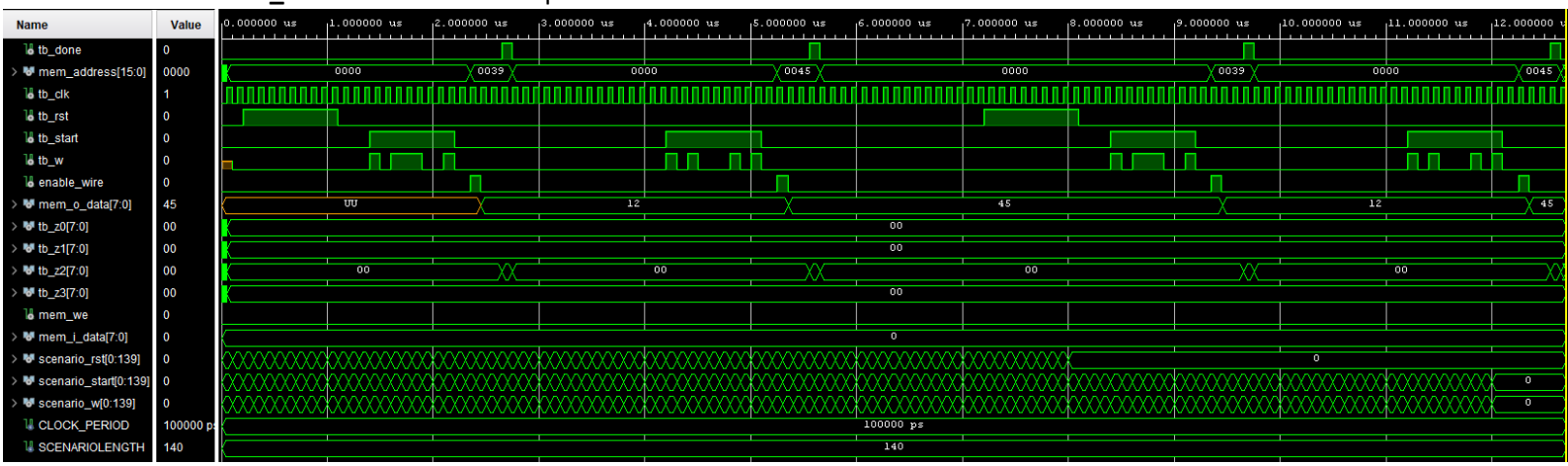


Una delle simulazioni più durature con il solo **RESET** iniziale, adatta a testare il componente su lunga durata e gli azzeramenti dei segnali e registri utilizzati ad ogni ciclo di elaborazione;

Numerosi segnali di input di lunghezza variabile;

Tutte le uscite vengono testate;

- **tb_4**: durata totale = 12.7µs



Simulazione con 2 segnali di **RESET**;

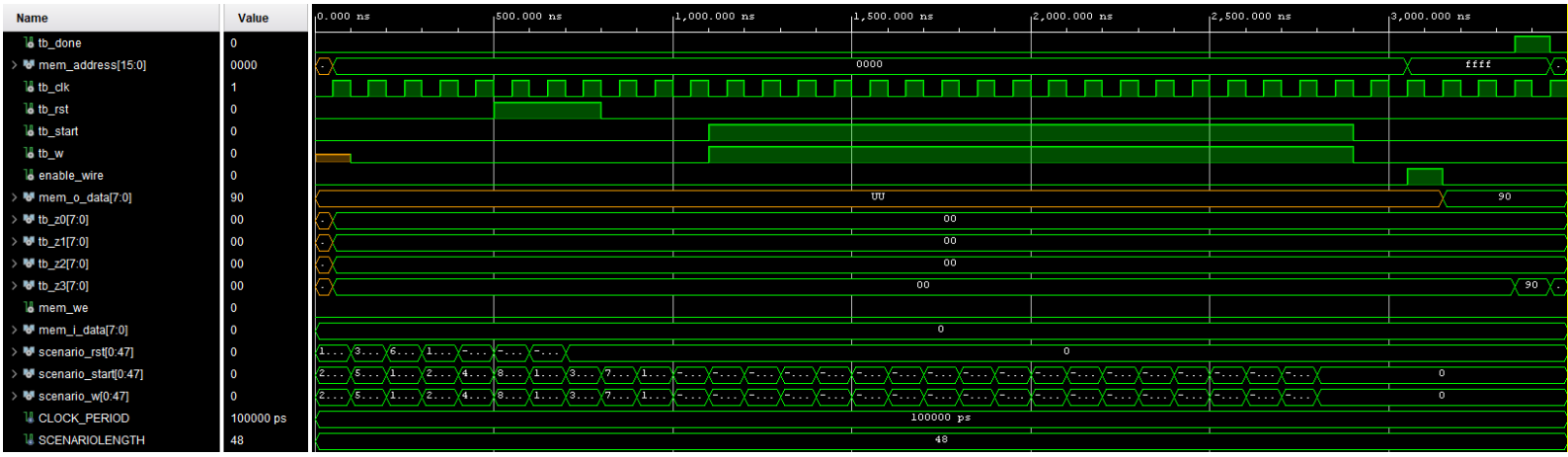
Questa simulazione si concentra sull'uscita **z2** tenendo tutte le altre costanti a 0;

È una simulazione che ripete due volte gli stessi input, adatta a testare il segnale **RESET**;

- **tb_5**: durata totale = 62.5µs

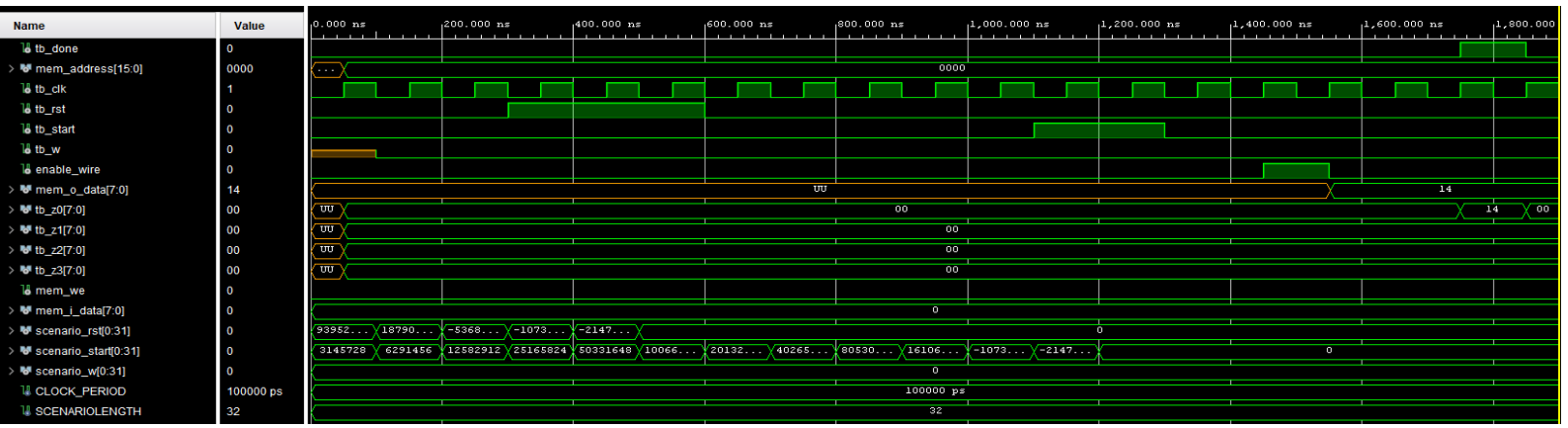
La simulazione più duratura, ma al contrario della **tb_3** ha due segnali di **RESET** rendendo la 3 più adatta al test di durabilità;
Questa simulazione ha molti input che aggiornano tutte le uscite tranne la **z0**;
Viene anche qui ripetuto due volte lo stesso input;
È una simulazione adatta a testare tutte le funzionalità che il componente deve avere;

- **tb_6**: durata totale = 3.5ns



Simulazione molto veloce che testa il caso in cui **START** e **W** sono segnali identici e ricevuti in contemporanea formati da 18 bit tutti con valore 1;

- **tb_7**: durata totale = 1.9ns



Simulazione molto veloce che testa il caso in cui **W** è fisso a 0 durante il periodo in cui **START** è attivo (2 cicli di clock).

4.Conclusioni

Questo progetto, oltre ad avermi fatto imparare le basi del linguaggio VHDL mi ha fatto rendere conto di quanto sia “potente” la logica dei circuiti, grazie ad essa si possono creare componenti capaci di replicare funzionalità che prima credevo possibili solo tramite algoritmo programmato e compilato.

Anche l’utilizzo della piattaforma Vivado mi ha interessato molto poiché mi ha permesso di vedere il componente in un suo potenziale design e dopo l’implementazione mi ha dato dei valori riguardo il suo consumo energetico ed altre caratteristiche.

Il più grande problema che ho riscontrato è stata la comprensione del funzionamento e dell’interfacciamento del programma da me scritto con le Testbench ma una volta capiti i miei errori, e dopo molti messaggi di debug e di arresto delle simulazioni, tutto è andato secondo i piani.