PROVA FINALE

PROGETTO DI RETI LOGICHE

POLITECNICO DI MILANO

INGEGNERIA INFORMATICA

Studente: GIOVANNI NI

Docente: Fabio Salice

Anno accademico: 2023-2024

INDICE

1. INTRODUZIONE

1.1 Scopo del progetto

Lo scopo del progetto è quello di implementare un modulo hardware descritto in VHDL che si interfacci con una memoria.
Tale sistema deve riuscire a leggere una sequenza di parole memorizzate a partire da un indirizzo specificato (ADD) ogni due byte. Per il byte mancante, il modulo da progettare avrà il compito di completarlo, sostituendo gli zero inserendo un valore di "credibilità" C per ogni valore della sequenza.

1.2 Specifiche generali

Entrando nel dettaglio del funzionamento del modulo.

Il modulo da implementare presenta tre ingressi primari: uno a 1 bit (i_START), uno a 16 bit (ADD) e uno a 10 bit (i_K), oltre a una uscita primaria a 1 bit (DONE). In aggiunta, il modulo è dotato di un segnale di clock (CLK), unico per l'intero sistema, e di un segnale di reset, anch'esso unico per l'intero sistema. Tutti i segnali sono sincroni e devono essere interpretati sul fronte di salita del clock, con l'eccezione di RESET, che è asincrono.

All'inizio, al momento del reset del sistema, l'uscita DONE deve essere a 0. Quando RESET torna a zero, il modulo inizierà l'elaborazione quando il segnale START in ingresso verrà portato a 1. Il segnale START rimarrà alto fino a quando il segnale DONE non verrà portato alto. Al termine della computazione (e dopo aver scritto il risultato in memoria), il modulo da progettare deve innalzare (portare a 1) il segnale DONE per segnalare la fine dell'elaborazione. Il segnale DONE deve rimanere alto fino a quando il segnale START non viene riportato a 0. Un nuovo segnale START non può essere inviato fino a quando DONE non viene riportato a zero.

Il modulo deve essere progettato tenendo conto che prima del primo START=1 (e prima di verificare il corretto funzionamento del modulo) verrà sempre inviato il RESET (RESET=1). Prima del primo reset del sistema, il funzionamento del modulo da implementare non è specificato. Una seconda (o successiva) elaborazione con START=1 non dovrà invece attendere il reset del modulo. Ogni volta che viene inviato il segnale di RESET (RESET=1), il modulo viene re-inizializzato.

Quando il segnale di START viene portato a 1 (e per l'intero periodo in cui rimane alto), sugli ingressi ADD e K vengono impostati il primo indirizzo e la dimensione della sequenza da elaborare. Prima di alzare il segnale di DONE, il modulo deve aggiornare la sequenza e i relativi valori di credibilità al valore appropriato, seguendo la descrizione generale del modulo.

Nota: Se il primo dato della sequenza è pari a zero, il suo valore rimane tale e il valore di credibilità deve essere posto a 0(zero).Lo stesso succede fino al raggiungimento del primo dato della sequenza con valore diverso da zero.

Riassumendo:

- Ogni parola contenuta in ogni indirizzo ha un valore tra 0 e 255.
- Il valore 0 non viene interpretato come valore ma come informazione "il valore non è specificato"

- La sequenza di K parole da elaborare è memorizzata a partire da un indirizzo specificato (ADD), ogni 2 byte (per esempio ADD, ADD+2, ADD+4 …) Il byte mancante dovrà essere completato dal modulo con il valore di credibilità.
- La sostituzione degli zero avviene copiando l'ultimo valore valido (non zero) letto precedente e appartenente alla sequenza.
- Il valore di credibilità C è pari a 31 ogni volta che il valore della sequenza è non zero, mentre viene decrementato (minimo C=0) rispetto al valore precedente ogni volta che si incontra uno zero.
- Un segnale di START determina la richiesta di codifica, un segnale DONE la sua fine.

Di seguito due esempi:

- Sequenza di partenza (in grassetto i valori W delle parole, K=14):

128 0 **64** 0 **0** 0 **0** 0 **0** 0 **0** 0 **100** 0 **1** 0 **0** 0 **5** 0 **23** 0 **200** 0 **0** 0

128 31 **64** 31 **64** 30 **64** 29 **64** 28 **64** 27 **64** 26 **100** 31 **1** 31 **1** 30 **5** 31 **23** 31 **200** 31 **200** 30

- Sequenza Finale

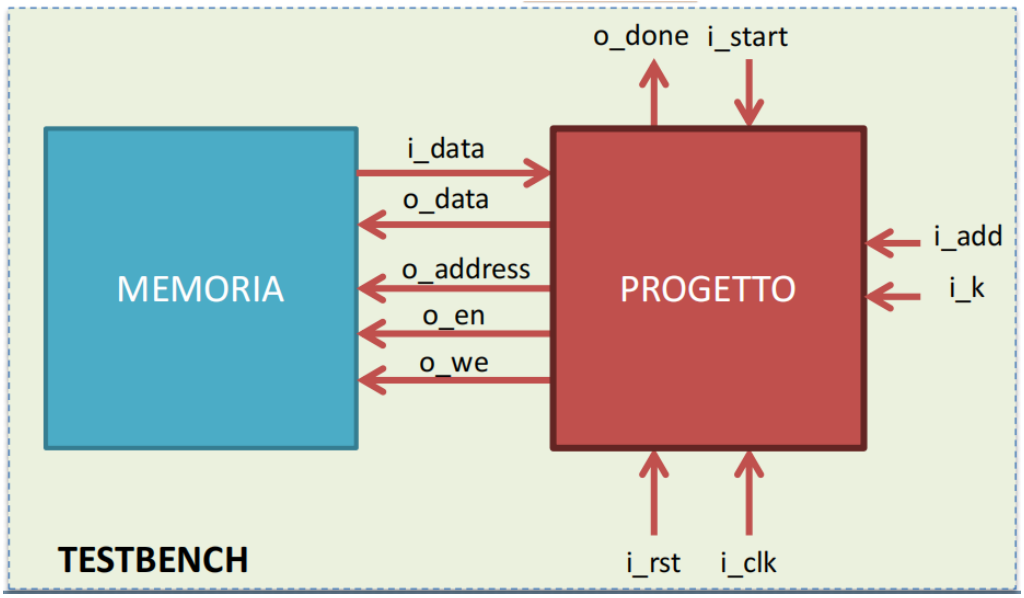
Esempio 1

K 00A

ADD 0064

Prima		Dopo	
ADD	W	ADD	W
0064	33	0064	33
0065	00	0065	1F
0066	00	0066	33
0067	00	0067	1E
0068	39	0068	39
0069	00	0069	1F
006A	18	006A	18
006B	00	006B	1F
006C	00	006C	18
006D	00	006D	1E
006E	00	006E	18
006F	00	006F	1D
0070	7E	0070	7E
0071	00	0071	1F
0072	00	0072	7E
0073	00	0073	1E
0074	C0	0074	C0
0075	00	0075	1F
0076	00	0076	C0
0077	00	0077	1E

1.3 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_add   : in std_logic_vector(15 downto 0);
        i_k     : in std_logic_vector(9 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_data : out 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_k*** è il segnale (vettore) K generato dal Test Bench rappresentante la lunghezza della sequenza;
- ***i_add*** è il segnale (vettore) ADD generato dal Test Bench che rappresenta l'indirizzo dal quale parte la sequenza da elaborare;
- ***o_done*** è il segnale DONE 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 e contiene il dato inseguito ad una richiesta di lettura
- ***o_mem_data*** è il segnale (vettore) che va verso la memoria e contiene il dato che verrà successivamente scritto;

- ***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)perpoter scriverci.Per leggere da memoria esso deve essere 0.

1.4 Dati e descrizione della memoria

I dati sono di dimensione 8 bit, memorizzati in una memoria data.
La memoria viene istanziata all'interno dei test bench a cui è sottoposto il componente. La struttura della memoria e il suo protocollo seguono le linee guida proposte nella documentazione di Xilinx.

2. DESIGN

2.1 Stati della macchina

- ***Stato RESET***

  Stato in cui viene resettato tutto e si mette in attesa del segnale i_start per passare allo stato di START e prepararsi per i passi successivi.
  In questo stato infatti o_done, o_mem_we, o_mem_en sono tutti settati a 0;

- ***Stato START***

  Stato in cui si prepara e passa allo stato successivo di calcolo indirizzo.
  o_done, o_mem_we, o_mem_en rimangono a 0;

- ***Stato CALC***

  Stato in cui calcola qual è il prossimo indirizzo di memoria da visitare. Dopo aver terminato ciò si passa allo stato di lettura.
  o_done, o_mem_we, o_mem_en rimangono a 0;

- ***Stato READ***

  In questo stato si effettua la lettura.
  o_mem_en viene messo a 1. Mentre o_done e o_mem_we rimangono a 0.

- ***Stato PROCESSING***

  Stato in cui vengono fatti i controlli e aumentati gli eventuali counter. Poi si passa allo stato di WRITE.

- ***Stato WRITE***

Stato di scrittura, individua se l'indirizzo visitato è un indirizzo dove il contenuto è una parola della sequenza o un indirizzo nella quale deve essere riportare il valore di credibilità. Nel secondo caso, riporta il valore di credibilità finora conservato. Nel primo caso invece, possono accadere due situazioni, se viene visitato il valore 0 (valore non specificato), questo viene sostituito con l'ultimo valore letto non nullo e il valore di affidabilità viene decrementato di 1. Se invece il valore letto non è 0, semplicemente il valore viene lasciato invariato e questo valore diventa l'ultimo valore non nullo e il valore di affidabilità viene reimpostata al valore massimo che è 31.
Successivamente viene controllato se ci sono ancora parole da leggere, in caso positivo ritorna allo stato di CALC per trovare il prossimo indirizzo, in caso contrario passa allo stato finale che è quello di DONE.
In questo stato o_mem_we ed o_mem_en sono settati a 1. Mentre o_done rimane a 0.

- ***Stato DONE***

  Stato finale, sono state gestite tutte le parole. O_done = 1.
  Aspetta finché i_start diventi 0 per ritornare allo stato di RESET.

2.2 Scelte progettuali

Di seguito una tabella dei segnali e variabili interne utilizzate

NOME	TIPO	VALORE INIZIALE	NOTE
curr_state	state	RESET	Memorizza lo stato corrente della FSM
counter	Integer range to 1024	0	Utilizzato per calcolare l'indirizzo di memoria da visitare. Incrementa di uno ad ogni ciclo.
write_confidence	std_logic	0	Utilizzata nello stato WRITE per stabilire se in un certo indirizzo è contenuto una parola della sequenza o meno.
confidence	std_logic_vector（4 down to 0)	11111	Memorizza il numero di credibilità attuale
last_value	std_logic_vector(7 down to 0)	00000000	Memorizza l'ultimo valore diverso non nullo.
cycle	std_logic_vector（10 down to 0)	0000000000	Memorizza il numero di indirizzi totali che la FSM deve visitare. Sarà il doppio dell'i_k, (numero di parole da visitare).
addr	std_logic_vector（15 down to 0)	0000000000000000	Utilizzato per il calcolo dell'indirizzo da visitare
first	std_logic	0	Utilizzato per gestire il caso speciale, ovvero quando una sequenza di parole inizia con dei valori non specificati (0)

Nota: per ogni variabile/segnale interno sopra elencato, ci sarà anche un x_next (per esempio state e state_next). Questo per stabilire qual è il prossimo valore che questa variabile/segnale deve assumere.

Il componente è dotato di due processi:

- Il primo processo rappresenta la parte sequenziale e la gestione dei registri

Precisamente:

Condizione con cui il reset si alza => lo stato ritorna allo stato di RESET indipendentemente da quale stato si trova in quel momento.

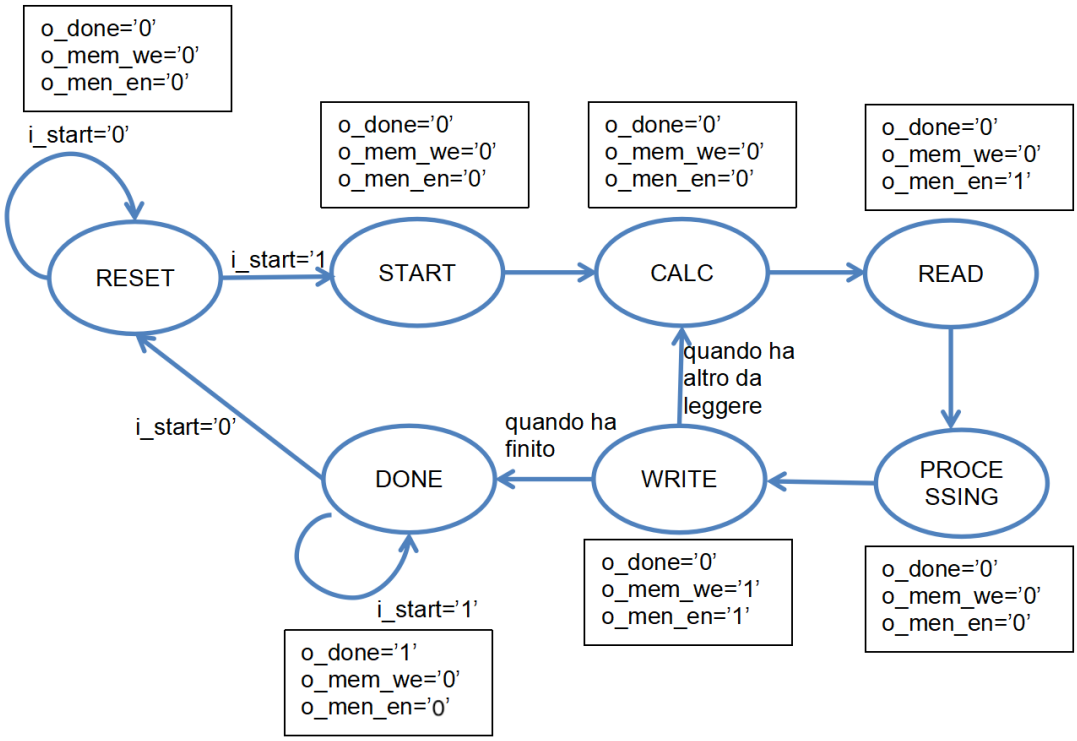
Condizione con cui il clock si alza => tutti i segnali vengono aggiornati dai segnali precedentemente elaborati. (x <= x_next)

- Il secondo processo rappresenta la FSM, che dopo aver le operazioni necessarie in base allo stato corrente determina il prossimo stato in cui si troverà il componente

Pre-eseguimento (prima che esegue la FSM): tutti i segnali di output assegna i valori di default, e i segnali di "x_next" ai segnali attuali. (dove x sono i vari segnali attuali)

La FSM viene eseguita sequenzialmente e gli stati di CALC e PROCESSING rappresentano gli stati intermedi tra READ e WRITE. Per la funzione dettagliata della macchina a stati si rimanda alla sezione precedente *STATI DELLA MACCHINA*.

Di seguito una figura della FSM:



3. RISULTATI DEI TEST

3.1 Report

Timing Report

```
Slack (MET) :             17.533ns  (required time - arrival time)
  Source:                 FSM_sequential_current_state_reg[2]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Destination:            confidence_reg[0]/S
                            (rising edge-triggered cell FDSE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns - clock rise@0.000ns)
```

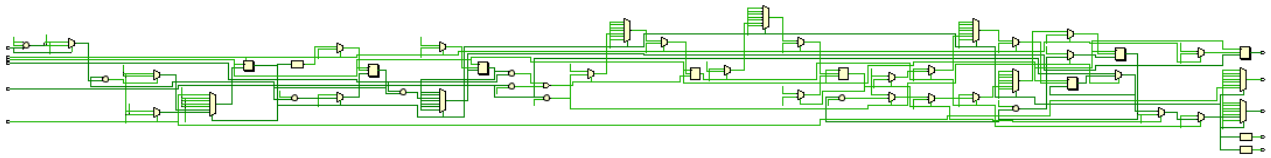
La figura è stata ripresa dalla console di Vivado.
Possiamo vedere che il timing soddisfa perfettamente gli obiettivi richiesti.

Utillization Report

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	80	0	0	78600	0.10
LUT as Logic	80	0	0	78600	0.10
LUT as Memory	0	0	0	26600	0.00
Slice Registers	45	0	0	157200	0.03
Register as Flip Flop	45	0	0	157200	0.03
Register as Latch	0	0	0	157200	0.00
F7 Muxes	0	0	0	39300	0.00
F8 Muxes	0	0	0	19650	0.00

Sempre ripresa dalla console di Vivado.
Vediamo che il dispositivo progettato è perfettamente sintetizzabile con 80 Look Up Table e 45 registri tra cui 45 Flip Flop e nessun inferred latch come richiesto. (FPGA usato: xc7z030sbg485-1)

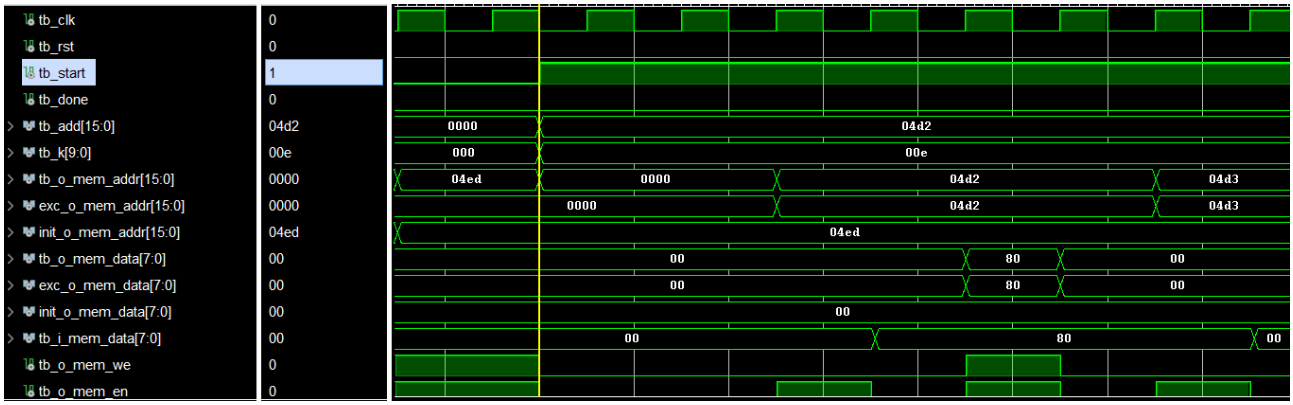
Schema del componente



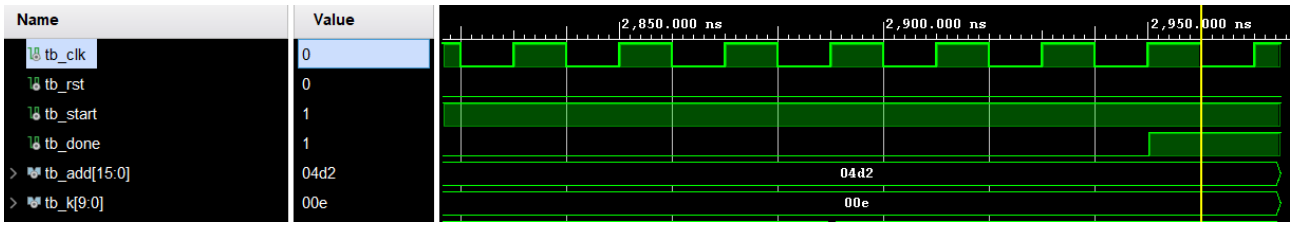
3.2 Simulazioni

Tutti i Test Bench forniti sono terminati con successo superando sia la Behavioral Simulation che la Post-Synthesis Functional Simulation

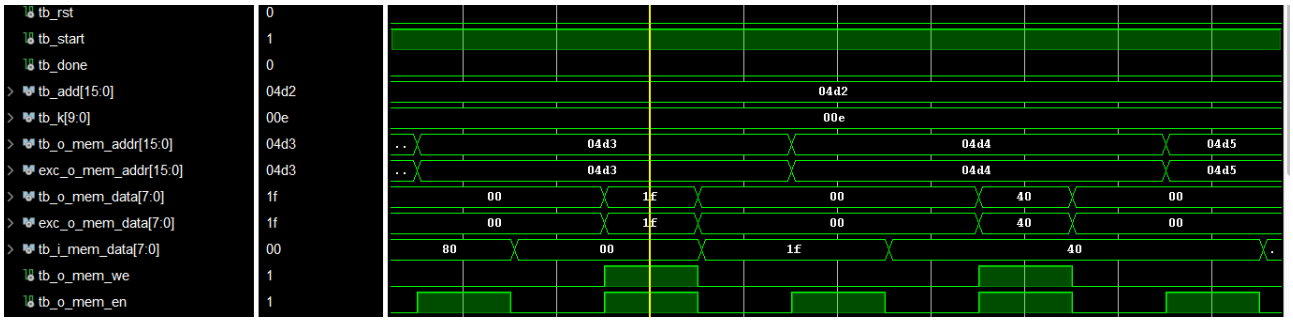
- i_start che passa da 0 a 1, fa iniziare il processo e si passa dallo stato di reset allo stato di start.



- Quando il modulo ha finito di leggere e scrivere tutta la sequenza, imposta la o_done a '1' e attende quando l'i_start diventa 0 per ritornare allo stato di reset.

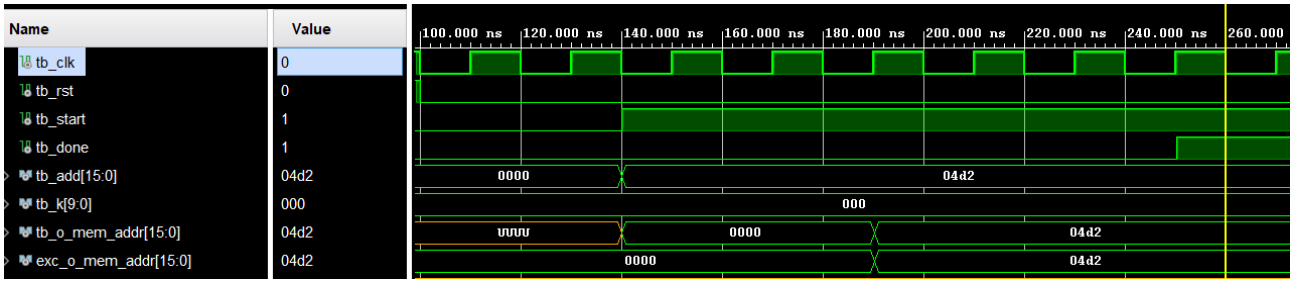


- Lettura e scrittura



In questo caso, per esempio, legge il valore 00 e scrive il valore 1f (31 in decimale) all'indirizzo 04d3.

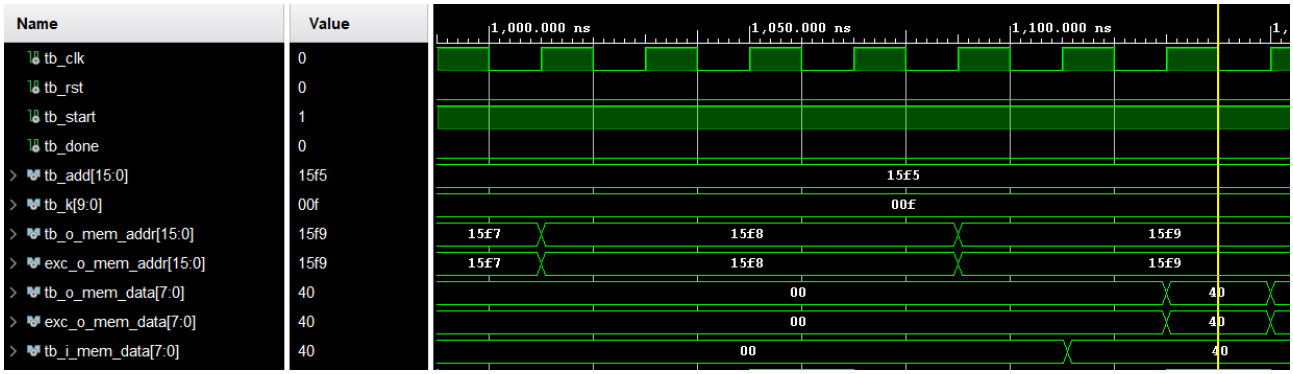
- Caso particolare in cui la sequenza è vuota



in poco tempo viene determinato che la sequenza è vuota, e subito viene segnato la fine (o_done = '1')

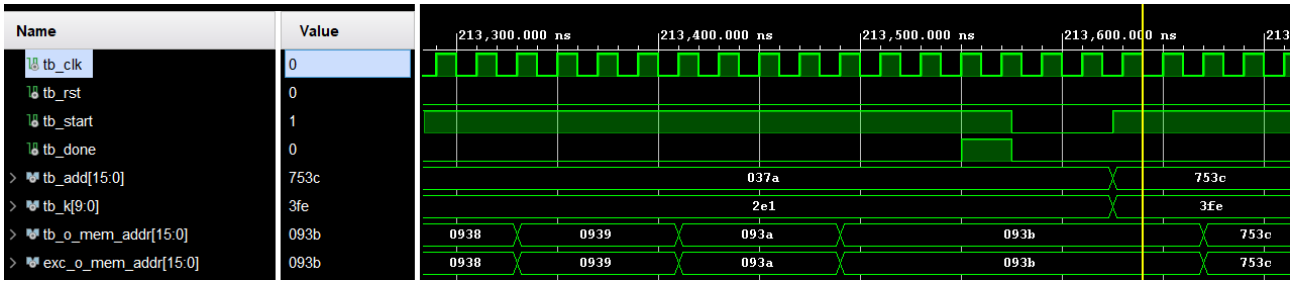
- Caso particolare quando una sequenza inizia con alcuni valore non specificati (0)

```
(0, 0, 0, 0,  64,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 100,  0, 1,  0, 0,  0, 5,  0, 23,  0, 200,  0,  0,  0 );
```



partendo dall'indirizzo 15f5, le prime due coppie devono essere invariate (valore non specificato 0), solo dall'indirizzo 15f9 inizia a gestire.

- Lettura di più sequenza (i_start viene abbassato e rialzato)



Si noti che una volta finito la prima sequenza, o_done viene settato a 1. Una volta abbassato la i_start, anche o_done ritorna a 0. E si prepara per la seconda sequenza.

4. CONCLUSIONI

Il sistema progettato soddisfa tutte gli obiettivi richiesti, e svolge le attività in modo corretto sia in pre-sintesi che in post-sintesi. Il componente realizzato ha dimostrato di essere in grado di superare tutti i test eseguiti (sia test bench forniti dal professore che test bench autogenerati).

Come visto dalla sezione Report, le tempistiche rispettano il vincolo riguardante il tempo minimo di clock pari a 20ns.

Si ritiene quindi di aver descritto un componente hardware conforme alle specifiche.

Grazie alla progettazione di questo progetto con l'uso del software Vivado, sono state consolidate le conoscenze acquisite nel corso di Reti Logiche.