Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo – AA 2023/2024

***

Sommario

Introduzione

Scopo del progetto

Implementare un componente hardware in Vivado Hardware Defined Language capace di ricevere in input una sequenza di K parole, da completare sostituendo gli zero presenti con l'ultimo valore letto diverso da zero e inserendo un valore di credibilità C

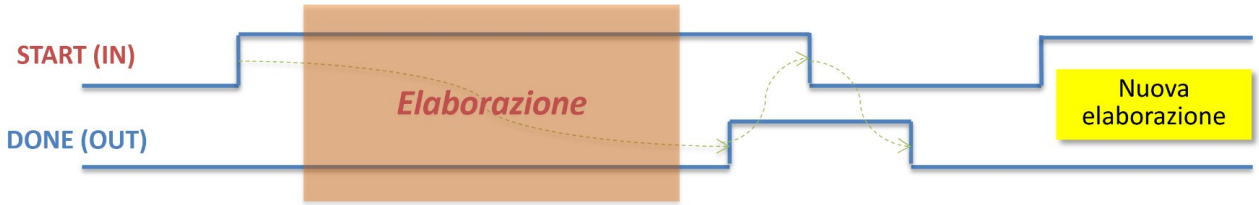
Esempio:

128	0	64	0	0	0	0	0	0	0	0	0	0	0	100	0	1	0	0	0
128	31	64	31	64	30	64	29	64	28	64	27	64	26	100	31	1	31	1	30

Specifiche generali

La specifica chiede di implementare un modulo HW (descri6o in VHDL) che si interfacci con una memoria e che rispetti le indicazioni riportate nella seguente specifica:

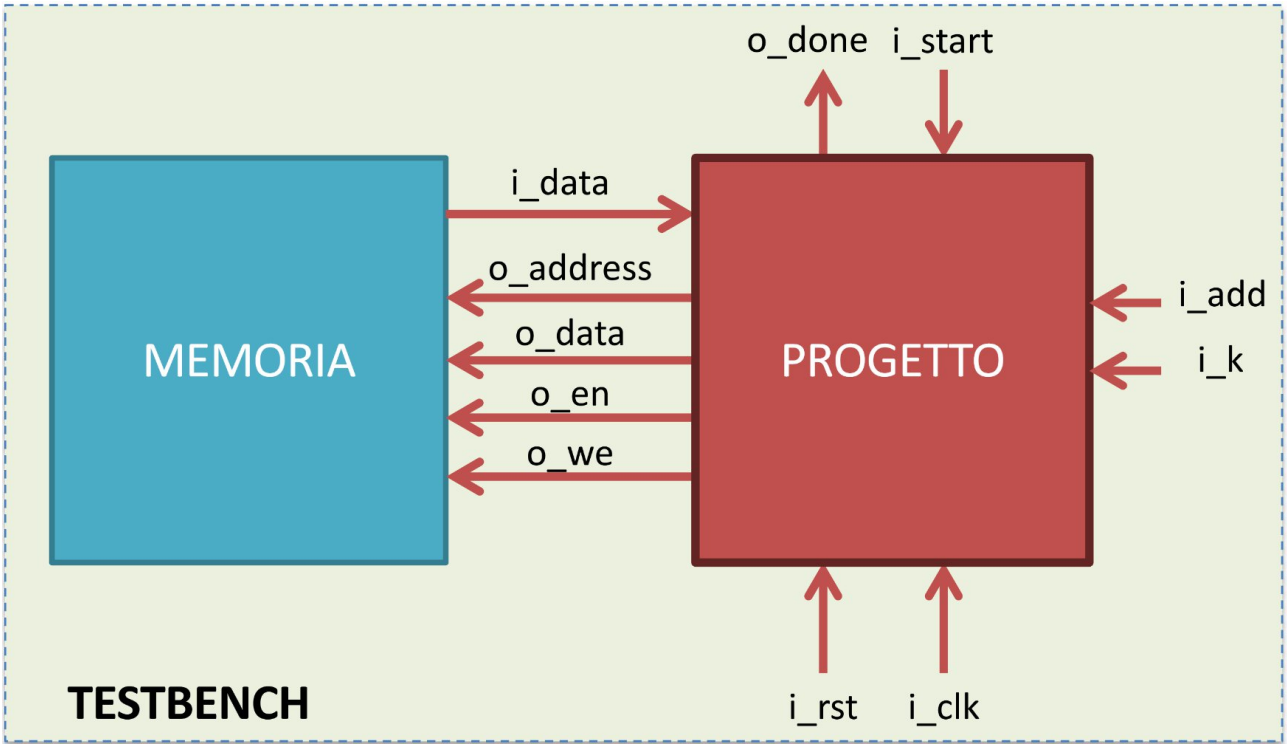
- Il sistema legge un messaggio costituito da una sequenza di K parole il cui valore è tra 0 e 255
- Il valore 0 all'interno della sequenza deve essere considerato non come valore ma come informazione "il valore non è specificato"
- La sequenza di K parole da elaborare è memorizzata a partire da un indirizzo specificato (ADD), ogni 2 byte (e.g. ADD, ADD+2, ADD+4, …, ADD+2*(K-1)). Il byte mancante dovrà essere completato come descritto in seguito
- Il modulo da progettare ha il compito di completare la sequenza, sostituendo gli zero laddove presenti con l'ultimo valore letto diverso da zero, ed inserendo un valore di "credibilità" C, nel byte mancante, per ogni valore della sequenza
- 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 (con associato ADD e K) determina la richiesta di codifica, un segnale DONE la sua fine
- Il modulo deve essere progettato considerando che prima della prima codifica verrà sempre dato il reset al modulo
- Il modulo deve essere progettato per poter codificare più sequenze
- Una seconda elaborazione non dovrà attendere il reset del modulo, ma si deve rispettare il seguente protocollo



- Quando START è alto i valori di K e ADD rimangono costanti
- Il TestBench rispetterà sempre questo protocollo

Interfaccia del componente

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_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 rappresentante l'indirizzo dal quale parte la sequenza da elaborare

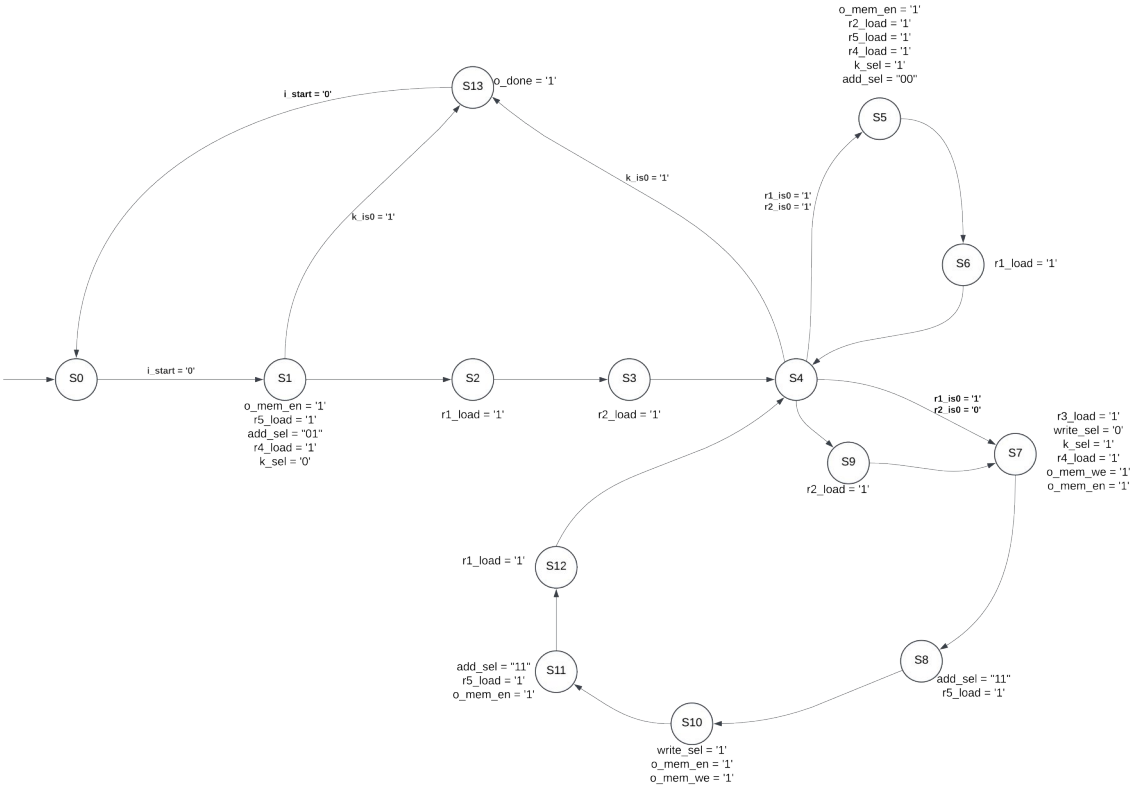
- o_done è il segnale DONE di uscita che comunica la fine dell'elaborazione
- o_mem_addr è il segnale (vettore) che arriva dalla memoria e contiene il dato in seguito ad una richiesta di lettura
- o_mem_data è il segnale (vettore) che va verso la memoria e contiene il dato che verrà successivamente scritto
- 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, varrà
    - 1 per scrivere
    - 0 per leggere

Design

Stati dell'FSM

La macchina a stati è così composta:

- S0: stato di IDLE, attendiamo che i_start = 1. Torniamo in questo stato ogni volta che i_rst = 1
- S1: in questo stato carichiamo nel registro 5 l'indirizzo iniziale (i_add) e nel registro 4 la lunghezza della sequenza (i_k). Settiamo inoltre o_mem_en a 1.
  Se la lunghezza k è pari a 0 passiamo subito allo stato 13
- S2: carichiamo nel registro 1 il valore del primo dato
- S3: carichiamo nel registro 2 il valore del registro 1
- S4: qui valutiamo il prossimo stato in cui andare:
  - se reg1 e reg2 sono uguali a zero (r1_is0 = 1 AND r2_is0 = 1) passiamo a S5
  - se reg1 è zero mentre reg2 no passiamo a S7
  - se k è zero (k_is0 = 1) passiamo a s13
  - altrimenti passiamo a S9
- S5: carichiamo nel registro 2 il valore del registro 1, aumentiamo di 2 il valore dell'indirizzo corrente e diminuiamo di 1 il valore di k
- S6: carichiamo il prossimo dato nel registro 1
- S9: carichiamo nel registro 2 il valore del registro 1
- S7: carichiamo nel registro 3 il valore della credibilità (gestito dalla logica del progetto), scriviamo il dato (contenuto in reg2), diminuiamo il valore di k e lo carichiamo nel registro 4
- S8: aumentiamo di uno l'indirizzo e lo carichiamo nel registro 5
- S10: scriviamo la credibilità (contenuta in reg3)
- S11: aumentiamo di uno l'indirizzo e lo carichiamo nel registro 5
- S12: carichiamo il prossimo dato nel registro 1
- S13: settiamo o_done a 1 e attendiamo che i_start torni a 0



Segnali

Di seguito i segnali utilizzati:

- Il registro 1 contiene il corrente valore della sequenza, sarà sempre un valore in posizione dispari (partendo da indice 1)
- Il registro 2 contiene l'eventuale valore precedente della sequenza
- Il registro 3 contiene il valore della credibilità
- Il registro 4 contiene il valore della lunghezza k rimanente
- Il registro 5 contiene l'attuale indirizzo
- I segnali r1_load, r2_load, r3_load, r4_load, r5_load gestiscono il salvataggio dei dati nei rispettivi registri
- I segnali r1_is0, r2_is0, k_is0 saranno pari a 1 quando reg1, reg2 o reg4 sono pari a zero
- r3_is0 ha una logica differente (capiremo il motivo più tardi):
    - è pari a "00" quando *(r2_0 = '0' and r1_0 = '0')*
    - è pari a "11" quando *(r2_0 = '1' and r1_0 = '1') or reg3 = "00000")*
    - è pari a "01" quando *(r2_0 = '0' and r1_0 = '1')*
    - "XX" altrimenti
- Il segnale *write_sel* controlla un *mux* che scrive in *o_mem_data* (quando necessario)
    - Il contenuto del registro 2 quando write_sel è 0
    - Il contenuto del registro 3 quando write_sel è 1
- Il segnale *k_sel* controlla un *mux* che inizialmente (0) è pari a *i_k* e che diminuisce *k* iterativamente quando *k_sel = 1*
- Il segnale *add_sel* comanda un *mux* che inizialmente (01) è pari all'indirizzo fornito che può aumentare di 2 quando *add_sel = "00"* o di 1 quando *add_sel = "11"*
- Il segnale *mux3_sel* che gestisce la credibilità concatena *r2_is0* e *r1_is0* in or con *r3_is0* *((r2_is0 & r1_is0) or r3_is0)*

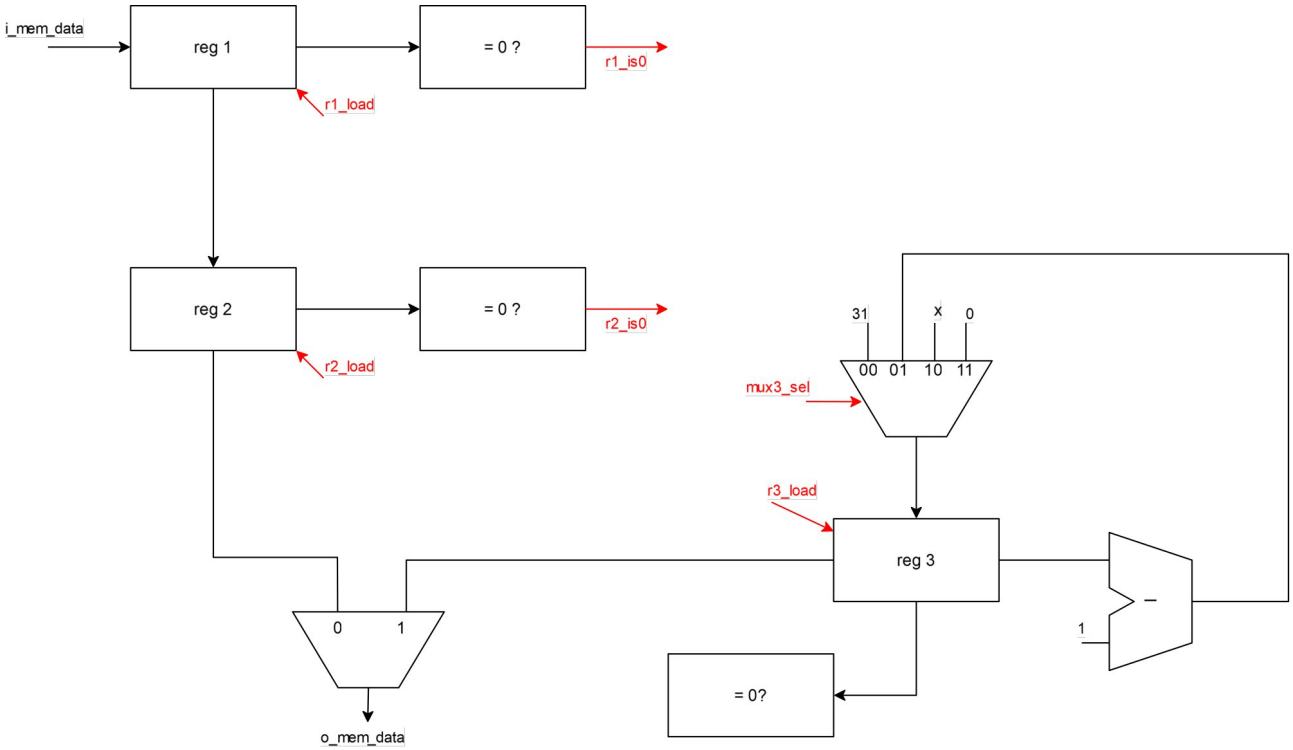


Figura 1 – Datapath

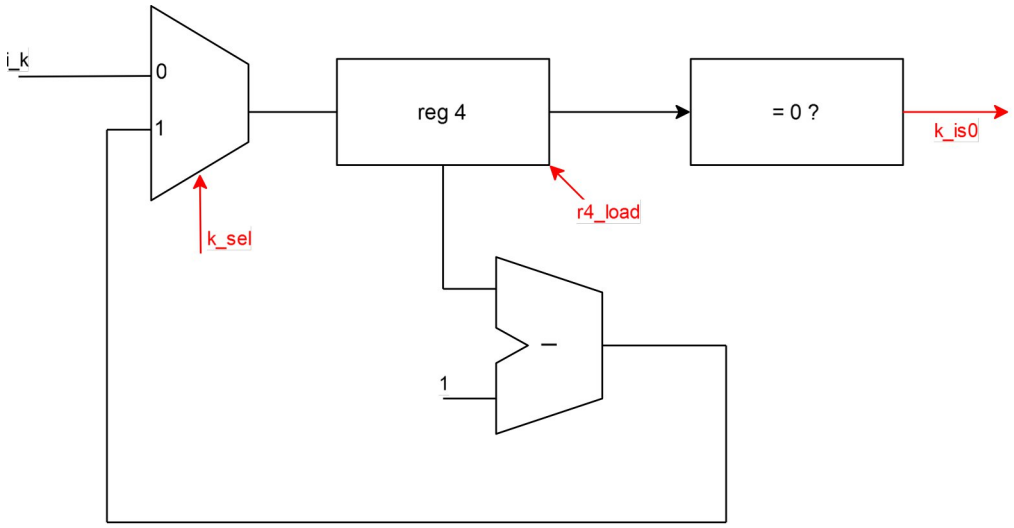


Figura 2 -Calcolatore del k

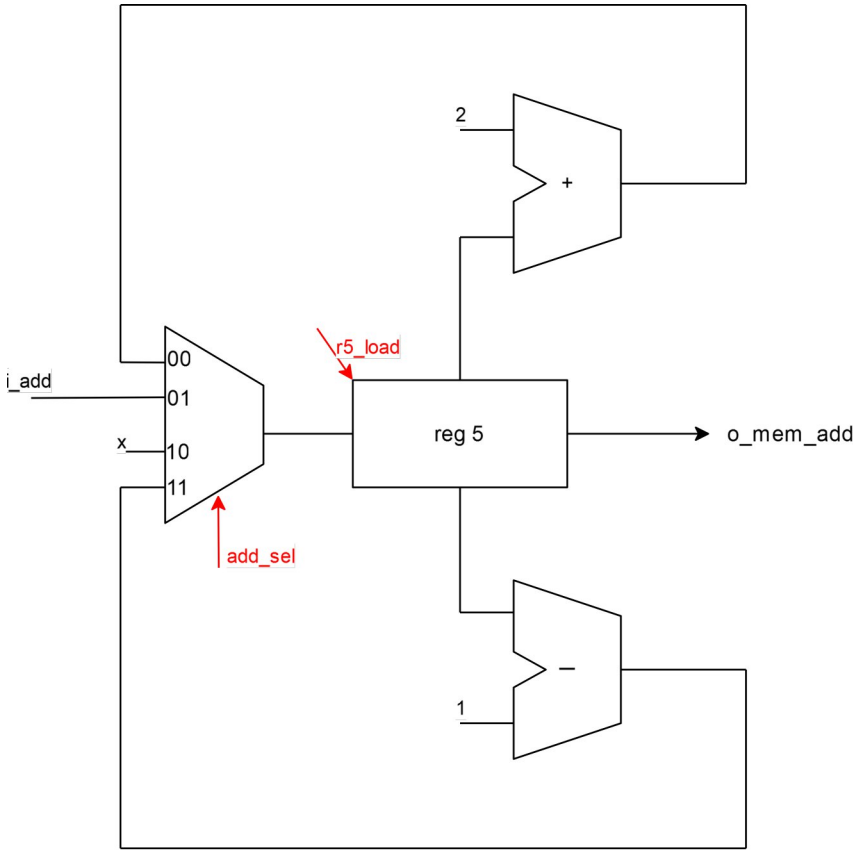


Figura 3 - Calcolatore dell'indirizzo

Risultati sperimentali

Report di sintesi

Dalla tabella è possibile osservare che il modulo non fa uso di alcun Latch, mentre i Flip Flop utilizzati sono 61 e le Look Up Tables 118.

```
report_utilisation:

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  118 |     0 |     10400 |  1.13 |
|   LUT as Logic          |  118 |     0 |     10400 |  1.13 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |   61 |     0 |     20800 |  0.29 |
|   Register as Flip Flop |   61 |     0 |     20800 |  0.29 |
|   Register as Latch     |    0 |     0 |     20800 |  0.00 |
| F7 Muxes                |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

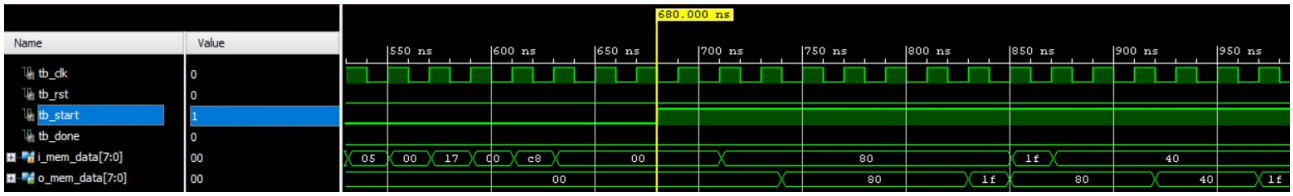
Per quanto riguarda invece il report_timing riusciamo tranquillamente a rientrare nel periodo di clock, infatti:

```
Slack (MET) :             6.133ns  (required time - arrival time)
```

Simulazioni

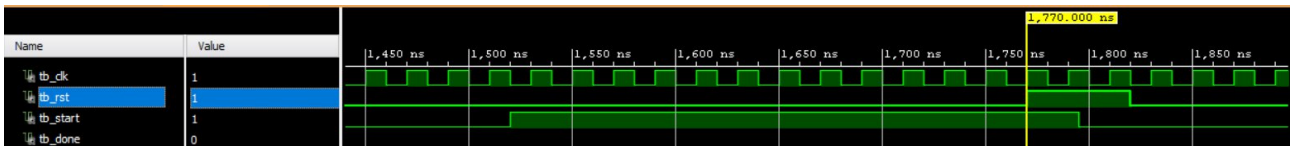
project_tb.vhd

Il primo testbench effettuato è stato quello fornito in partenza, contenente una generica sequenza. Questo testbench è utile per verificare il corretto funzionamento del modulo senza casi limite o particolari.



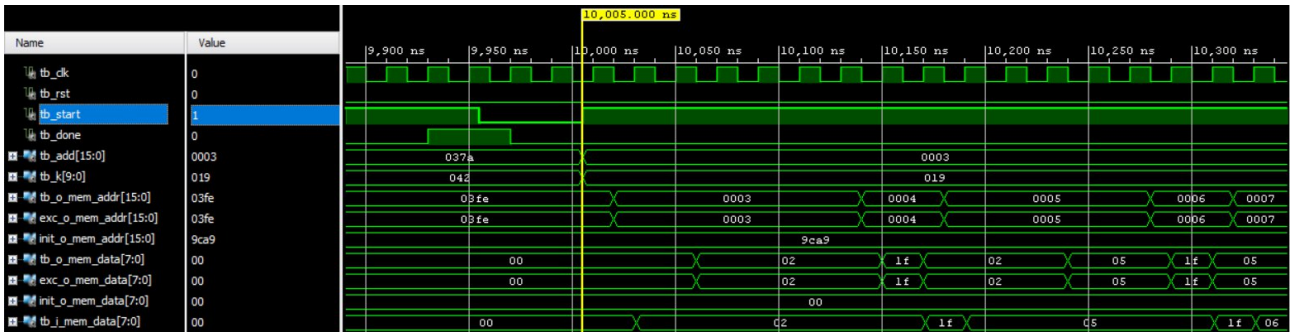
tb_reset_during_read.vhd

Questo è un testbench utile per controllare alcuni edge-case, infatti qui i_rst viene portato a 1 mentre start è ancora a 1. Inoltre, contiene anche diverse sequenze il che permette di verificare il corretto funzionamento anche con diversi input.



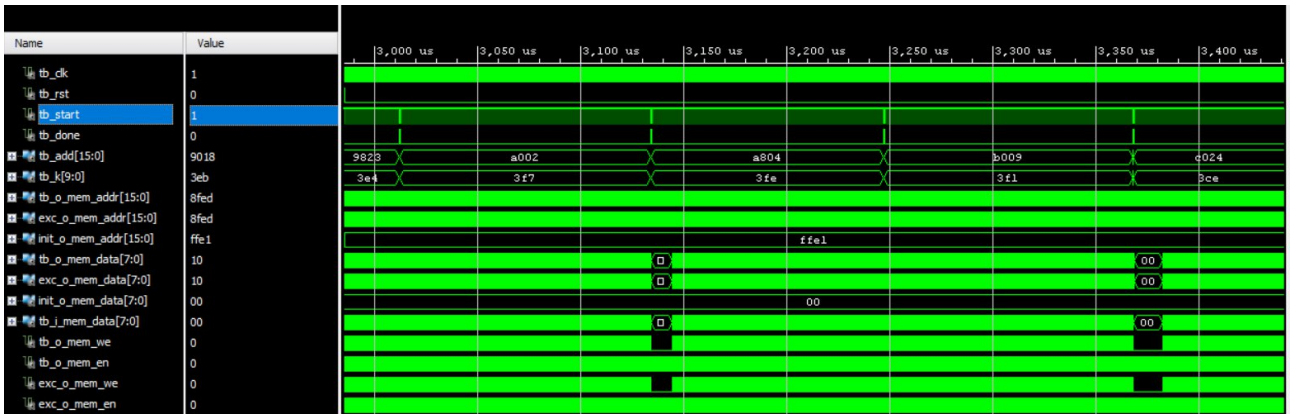
project_tb_07.vhd

Questo testbench, tra le altre, presenta una sequenza di soli 0. Ci consente dunque di valutare il corretto funzionamento anche in questo edge case.



Test_8214.vhd

Quest'ultimo testbench contiene 31 diverse (e lunghe) sequenze. Permette di valutare il corretto funzionamento del componente nel tempo



Conclusioni

I risultati ottenuti dai test bench effettuati (quindici in totale) confermano il corretto funzionamento del modulo, anche in post-synthesis (functional e timing), come da specifica.

L'FSM contiene probabilmente più stati di quelli strettamente necessari, ma questo consente di avere chiaro il ragionamento sottostante, senza ricercare semplificazioni estreme.