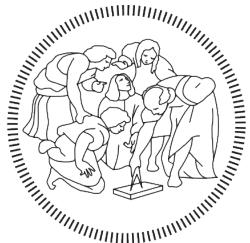


Prova Finale di Reti Logiche

Gianvito Caleca

Anno Accademico 2021/2022



POLITECNICO
MILANO 1863

Indice

1 Introduzione	3
1.1 Obiettivo del progetto	3
1.1.1 Specifica	3
1.1.2 Esempio	4
2 Architettura	5
2.1 Datapath	5
2.1.1 Sezione 1: convolutore	6
2.1.2 Sezione 2: stato	7
2.1.3 Sezione 3: Calcolo della selezione e controllo prossima parola	8
2.1.4 Sezione 4: Contatore parole lette e indirizzo di lettura/scrittura	9
2.2 Macchina a Stati	10
2.2.1 Valori di default	11
2.2.2 Descrizione degli stati	12
3 Risultati Sperimentali	13
3.1 Sintesi	13
3.2 Simulazioni	14
3.2.1 Test bench 1: Sequenza minima	14
3.2.2 Test bench 2: Multi start	14
3.2.3 Test bench 3: Reset asincrono	15
4 Conclusioni	15

1 Introduzione

1.1 Obiettivo del progetto

La specifica della "Prova Finale (Progetto di Reti Logiche)" 2021/2022 chiede di implementare un modulo Hardware descritto in VHDL che si interfacci con una memoria e che soddisfi la seguente specifica:

1.1.1 Specifica

Il modulo riceve in ingresso una sequenza continua di W parole di 8 bit, e restituisce una sequenza continua di $Z = 2W$ parole da 8 bit.

Ogni parola in ingresso viene serializzata per generare un flusso continuo da 1 bit, su questo flusso viene applicato il codice convoluzionale $\frac{1}{2}$, secondo lo schema riportato in figura 1.

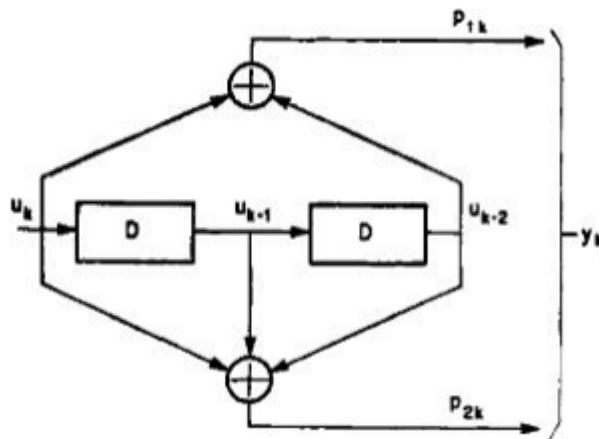


Figura 1: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$

Questa operazione genera in uscita un flusso continuo ottenuto come concatenamento alternato dei due bit di uscita p_{1k} e p_{2k} generati a partire dal bit in ingresso u_k .

La sequenza d'uscita Z è la parallelizzazione, su 8 bit, di questo flusso.

Il convolutore è una macchina sequenziale sincrona con un clock globale e un segnale di reset con il seguente diagramma degli stati che ha nel suo 00 lo stato iniziale, con uscite in ordine P1K, P2K (ogni transizione è annotata come $U_k/p_{1k}, p_{2k}$).

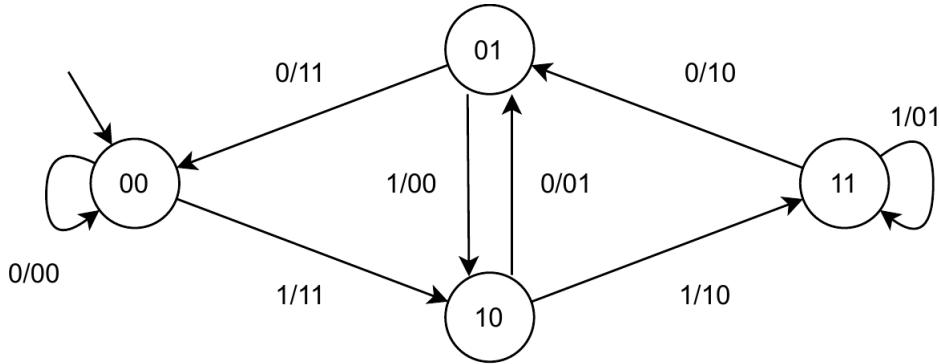


Figura 2: Macchina a stati

Il modulo da implementare leggerà la sequenza da codificare da una memoria con indirizzamento al byte. La sequenza di byte letti deve essere trasformata nella sequenza di bit U da elaborare.

All'indirizzo 0 è memorizzata la sequenza di parole W da codificare. Il primo byte della sequenza di byte da leggere è memorizzato all'indirizzo 1. Lo stream di uscita Z deve essere memorizzato a partire dall'indirizzo 1000. La dimensione massima della sequenza d'ingresso è 255 byte.

1.1.2 Esempio

La seguente contiene un esempio di funzionamento:

Sequenza in ingresso	1	0	1	0	0	0	1	0
Istante temporale	0	1	2	3	4	5	6	7
U_k	1	0	1	0	0	0	1	0
P_{1k}	1	0	0	0	1	0	1	0
P_{2k}	1	1	0	1	1	0	1	1

Corrispondente ai byte in uscita $Z_1: 11010001$ e $Z_2: 11001101$

2 Architettura

L'architettura del progetto consiste in due moduli differenti: Datapath e Macchina a stati

2.1 Datapath

Il datapath è costituito da una serie di processi che implementano i registri contenenti i valori letti dalla memoria e/o calcolati dai diversi sommatori, sottrattori presenti nel modulo.

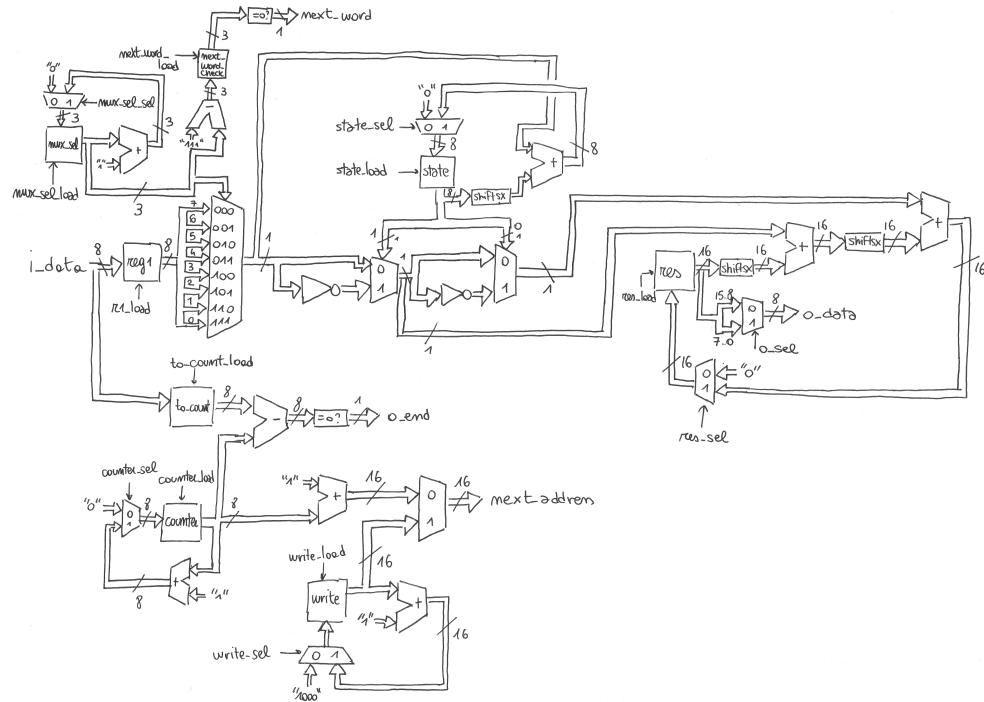


Figura 3: Datapath completo

È possibile considerare il modulo come composto da quattro differenti sezioni:

2.1.1 Sezione 1: convolutore

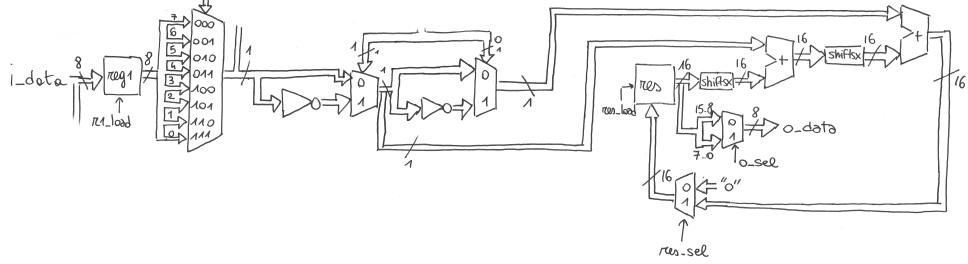


Figura 4: Convolutore

Questa sezione si occupa di generare i segnali P1k e P2k e di concatenarli nella creazione del risultato (a 16 bit che verranno poi "divisi" dal mux *o_sel*), ottenuto con due conseguenti **shift** a sinistra e somma prima con p1k, poi con p2k. Il multiplexer *res_sel* viene utilizzato per reimpostare il valore del risultato ad ogni computazione.

Il dato letto in ingresso viene salvato nel registro **reg1**, da cui vengono poi estratti i singoli bit tramite il multiplexer ed il segnale di selezione **mux_sel**, calcolato in un'altra sezione del datapath.

Da ogni singolo bit vengono poi generati i segnali P1k e P2k, in base allo stato corrente che viene "salvato" in una diversa sezione, seguendo la tabella qui riportata, realizzata a partire dai valori presenti nella macchina a stati fornita con la specifica:

Stato	bit in ingresso	p1k	p2k
00	0	0	0
00	1	1	1
01	0	1	1
01	1	0	0
10	0	0	1
10	1	1	0
11	0	1	0
11	1	0	1

Da cui si evince che:

- **p1k = bit in ingresso** se il secondo bit dello stato è uguale a 0
- **p1k = bit in ingresso negato** se il secondo bit dello stato è uguale a 1
- **p2k = p1k** se il primo bit dello stato è uguale a 0
- **p2k = p1k negato** se il primo bit dello stato è uguale a 1

2.1.2 Sezione 2: stato

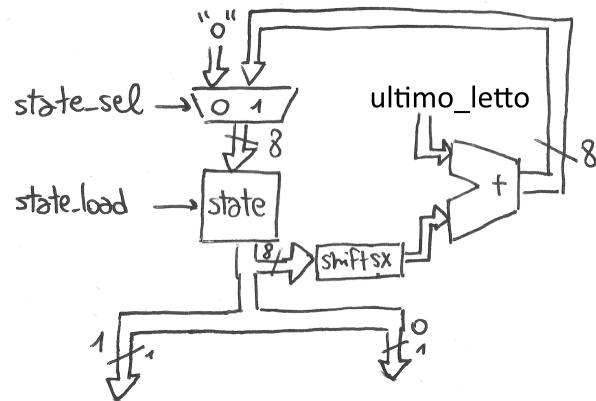


Figura 5: Stato

Questa sezione si occupa di aggiornare lo stato corrente.
Il mux *state_sel* viene utilizzato per reimpostare lo stato ad ogni differente computazione (*state_sel* = 0), o per aggiornarlo ad ogni lettura (*state_sel* = 1).

L'aggiornamento dello stato avviene tramite uno **shift** a sinistra ed alla somma del valore ottenuto con l'ultimo bit letto, in uscita dal multiplexer della sezione convolutore.

I bit in posizione '1' e '0' vengono utilizzati come bit di selezione per i multiplexer generatori dei segnali P1k e P2k (vedi sezione convolutore).

2.1.3 Sezione 3: Calcolo della selezione e controllo prossima parola

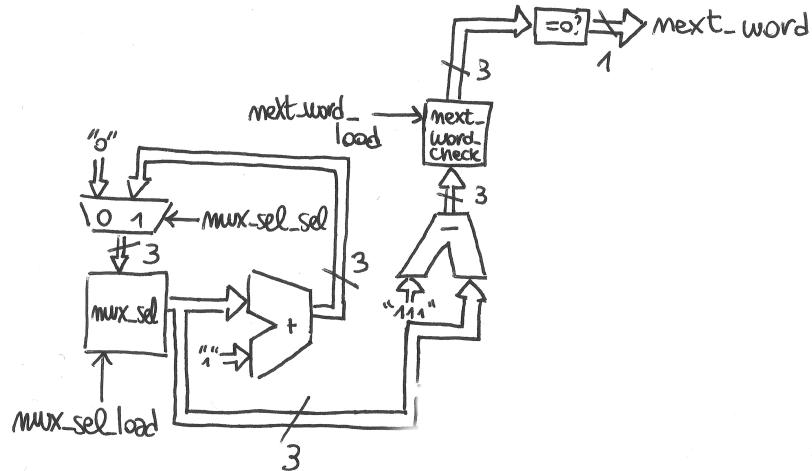


Figura 6: Calcolo della selezione e controllo prossima parola

In questa sezione viene mantenuto il dato riguardante l'ultimo bit computato della parola corrente, il valore generato viene utilizzato come selettore per il multiplexer della prima sezione. Questo viene fatto incrementando continuamente il valore utilizzando un sommatore.

Conseguentemente, il valore ottenuto viene confrontato, tramite una sottrazione ed un controllo di uguaglianza con zero con il valore massimo possibile (111). Quando questo è raggiunto, il segnale *next_word* vale '1', indicando come terminata la lettura della parola corrente.

Il mux *mux_sel_sel* viene utilizzato per reimpostare lo stato ad ogni differente computazione (*mux_sel_sel* = 0), o per aggiornarlo (*mux_sel_sel* = 1).

2.1.4 Sezione 4: Contatore parole lette e indirizzo di lettura/scrittura

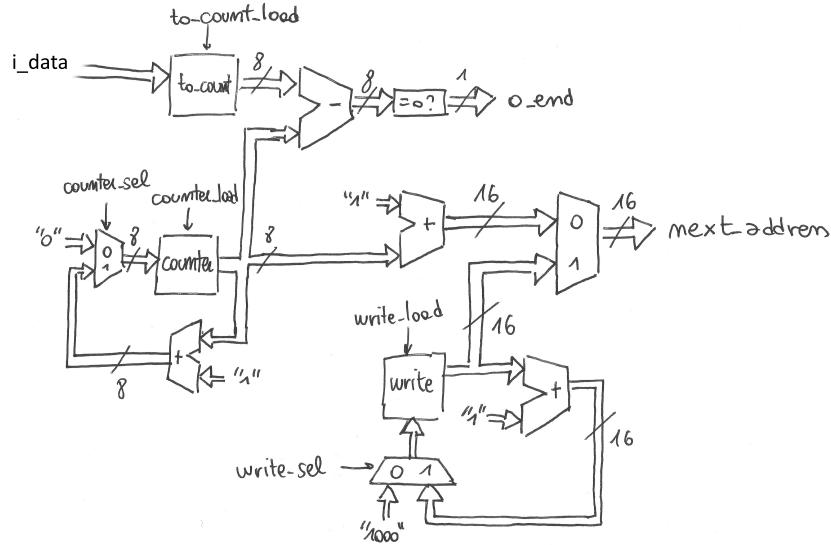


Figura 7: Contatore parole lette e indirizzo di lettura/scrittura

In questa sezione vengono contate il numero di parole lette correntemente. Il dato, mantenuto nel registro **counter** viene utilizzato per calcolare l'indirizzo di lettura, tramite una somma con il valore 1 (corrispondente al primo indirizzo di memoria da cui leggere), e per effettuare un confronto con il valore contenuto nel registro **to count**: se il numero di parole contate corrisponde a quelle che erano inizialmente da contare, allora il segnale *o_end* viene portato a '1'.

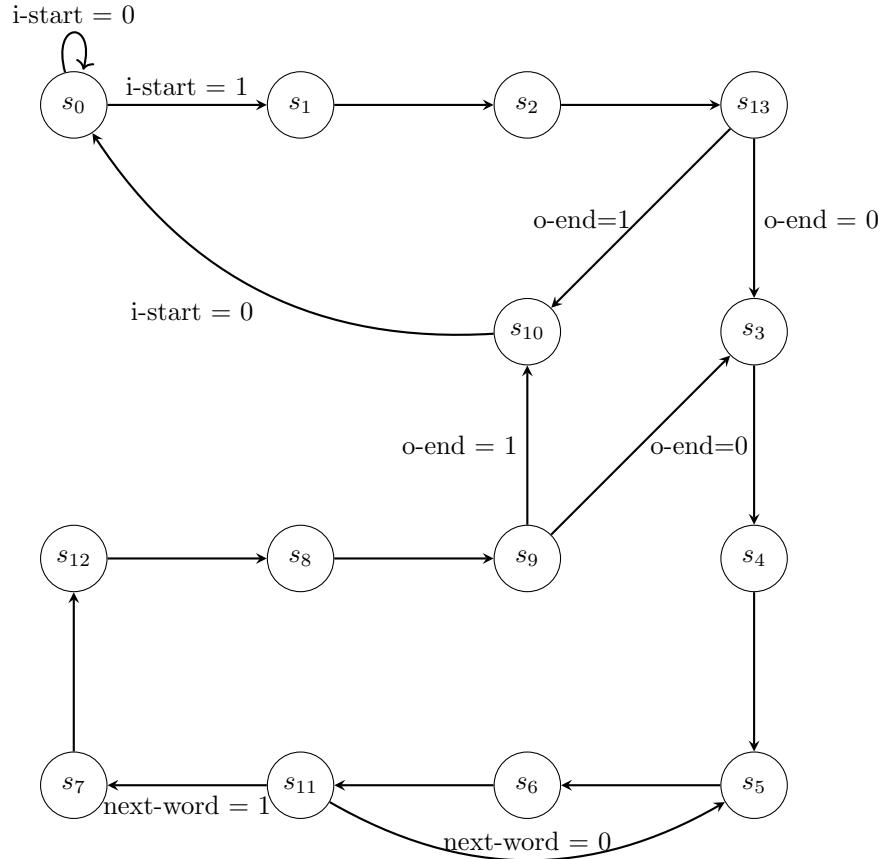
L'indirizzo di scrittura viene invece calcolato incrementando il primo indirizzo (1000), ottenuto dal multiplexer *write_sel* per utilizzarlo ad ogni computazione.

L'indirizzo di memoria da utilizzare in ogni momento viene fornito dal multiplexer *next_address* in base al valore di *o_addr_sel*, ottenuto in input dalla macchina a stati

2.2 Macchina a Stati

Il macchina a stati è implementata da tre process, uno per reimpostare lo stato corrente a quello iniziale in caso di segnale di reset attivo, uno per determinare ed assegnare lo stato prossimo allo stato corrente ad ogni ciclo di clock, ed uno per assegnare i valori ai segnali del datapath in base allo stato corrente

Il diagramma che rappresenta la macchina a stati è riportato qui sotto:



2.2.1 Valori di default

Segnale	Valore
<i>r1_load</i>	0
<i>res_load</i>	0
<i>to_count_load</i>	0
<i>state_load</i>	0
<i>counter_load</i>	0
<i>write_load</i>	0
<i>mux_sel_load</i>	0
<i>next_word_load</i>	0
<i>res_sel</i>	0
<i>state_sel</i>	0
<i>write_sel</i>	1
<i>counter_sel</i>	0
<i>mux_sel_sel</i>	0
<i>o_addr_sel</i>	0
<i>o_sel</i>	0
<i>o_done</i>	0
<i>o_address</i>	0000000000000000
<i>o_en</i>	0
<i>o_we</i>	0

2.2.2 Descrizione degli stati

Stato	Descrizione	Segnali assegnati
S0	Stato iniziale	$o_done = 0$
S1	Inizializzazione: i valori dei registri vengono portati a quelli iniziali, viene attivata la memoria per ottenere il numero di parole da leggere	$o_en = 1$ $counter_load = 1$ $state_load = 1$ $write_sel = 0$ $write_load = 1$ $res_load = 1$
S2	Viene caricato il numero di parole da contare nel registro <i>to_count</i>	$to_count_load = 1$
S3	Viene abilitata la lettura da memoria e caricato l'indirizzo di lettura	$o_en = 1$ $o_address = next_address$
S4	Viene caricata la prima parola letta nel registro <i>reg_1</i>	$r1_load = 1$
S5	Viene caricato nel registro del risultato il valore calcolato dal datapath	$res_load = 1$ $res_sel = 1$
S6	Viene aggiornato lo stato corrente, ed incrementato il selettore del multiplexer per il prossimo bit	$state_load = 1$ $state_sel = 1$ $mux_sel_load = 1$ $mux_sel_sel = 1$
S7	Viene scritta in memoria la prima porzione di output	$o_addr_sel = 1$ $o_en = 1$ $o_we = 1$ $o_address = next_address$
S8	Viene scritta in memoria la seconda porzione di output, il contatore viene incrementato	$o_addr_sel = 1$ $o_en = 1$ $o_we = 1$ $o_sel = 1$ $o_address = next_address$ $counter_load = 1$ $counter_sel = 1$
S9	Viene aggiornato l'indirizzo di scrittura in memoria	$write_load = 1$
S10	Alza il segnale <i>o_done</i> per tornare in attesa di input	$o_done = 1$
S11	Carica il valore di <i>next_word_checker</i> nel registro per determinare se la lettura della parola è terminata	$next_word_load = 1$
S12	Viene aggiornato l'indirizzo di scrittura in memoria	$write_load = 1$
S13	Viene ripristinato il valore '000' per il <i>mux_sel</i>	$mux_sel_load = 1$

3 Risultati Sperimentali

3.1 Sintesi

Il componente è correttamente sintetizzabile ed implementabile con l'utilizzo di 70 LUT, 78 Flip Flops e 0 Latch.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	70	0	0	134600	0.05
LUT as Logic	70	0	0	134600	0.05
LUT as Memory	0	0	0	46200	0.00
Slice Registers	78	0	0	269200	0.03
Register as Flip Flop	78	0	0	269200	0.03
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

Figura 8: Slice Logic

Il componente inoltre ha uno Slack di 96.069 ns

```
Timing Report

Slack (MET) : 96.069ns (required time - arrival time)
```

Figura 9: Slack

3.2 Simulazioni

3.2.1 Test bench 1: Sequenza minima

Questo test bench verifica che il componente si comporti in maniera corretta in presenza del valore '0' nella cella di indirizzo '0' in memoria, ovvero in presenza di zero parole da codificare.

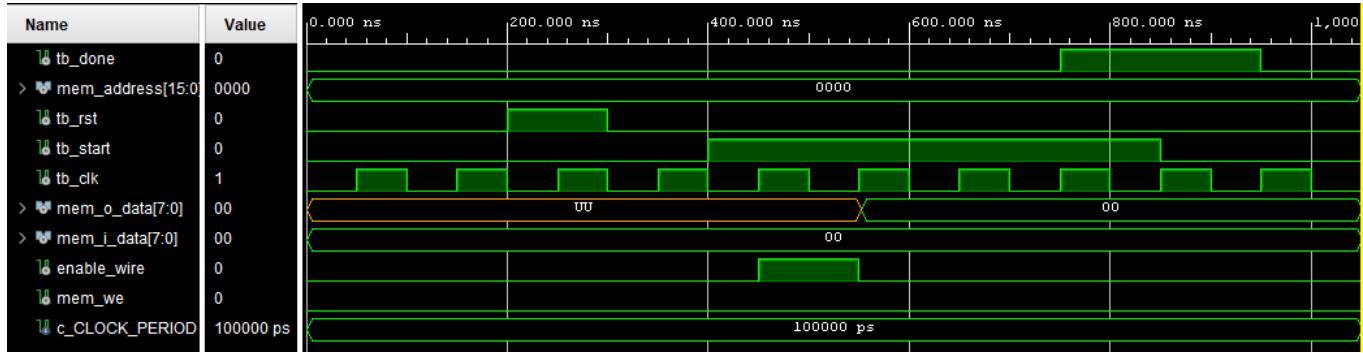


Figura 10: Sequenza minima

3.2.2 Test bench 2: Multi start

Il test bench verifica che il componente si comporti in maniera corretta quando viene richiesto di eseguire computazioni multiple l'una di seguito all'altra.

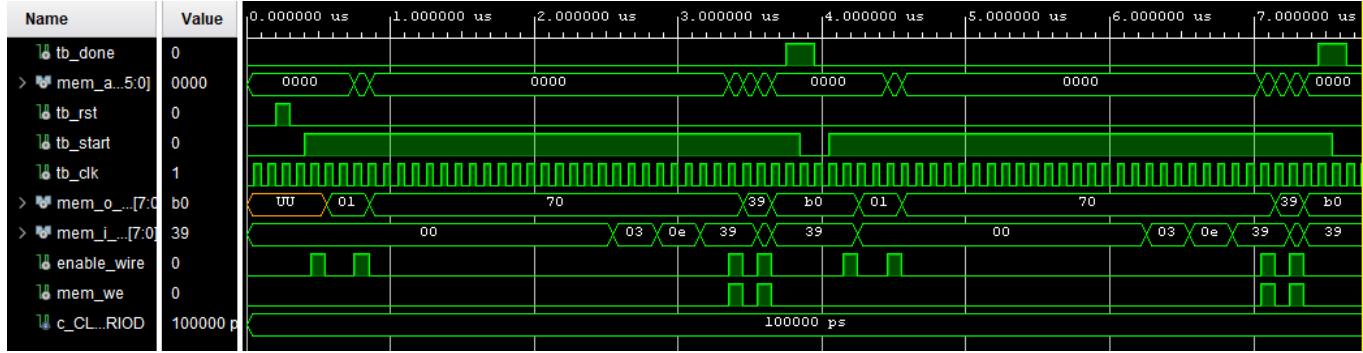


Figura 11: Multi start

3.2.3 Test bench 3: Reset asincrono

Il test bench verifica che il componente si comporti in maniera corretta quando il segnale di reset viene portato ad '1' durante un'elaborazione in corso

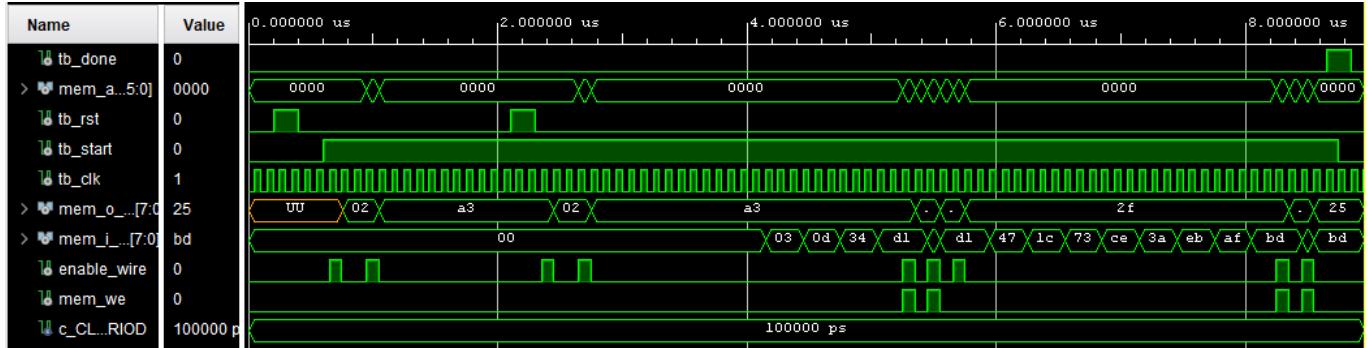


Figura 12: Reset Asincrono

4 Conclusioni

La realizzazione di un componente di questo tipo è una sfida che a mio parere ogni studente d'ingegneria informatica dovrebbe affrontare.

L'utilizzo di strumenti di modellazione e sintesi di componenti come Vivado e di linguaggi come VHDL, di molto differenti da quelli utilizzati di norma nella classica programmazione, aprono ad un nuovo modo di pensare e di interpretare concetti che solitamente venivano trascurati.

Personalmente, posso ritenermi soddisfatto, essendo riuscito a sviluppare un componente simile in autonomia, della mia preparazione complessiva per gli argomenti affrontati nel corso di Reti Logiche.