Progetto di Reti Logiche A.A.2021/2022

Angelo Giovanni Gaillet
matricola ***, codice persona ***

27 marzo 2022

Sommario

Questo documento contiene la relazione sullo svolgimento del progetto di reti logiche dell'anno accademico 2021/2022 necessaria al soddisfacimento dei requisiti della Prova finale di Reti Logiche sez. Prof. Salice, per il C.d.L. in Ingegneria Informatica del Politecnico di Milano.

Indice

1 Introduzione

La specifica del progetto chiede di descrivere con linguaggio VHDL un componente hardware che compia (accedendo ad una memoria e in seguito alla ricezione di un segnale di 'start') i seguenti passi:

1. Lettura del numero n di parole da 8 bit da processare (memorizzato all'indirizzo 0);
2. Lettura delle n parole in ordine (memorizzate dall'indirizzo 1 a quello n);
3. Serializzazione delle parole per creare un flusso di 1 bit;
4. Passare il flusso di bit in un codice convoluzionale con rapporto 1/2 (la descrizione di questo codice è presente nella sezione 2.4 di questo documento);
5. Parallelizzazione del flusso ottenuto per formare $2n$ parole da 8 bit ciascuna;
6. Memorizzazione delle parole ottenute in memoria (dall'indirizzo 1000 all'indirizzo $1000 + (2n - 1)$).

Inoltre il componente deve essere in grado di ripartire dal punto 1 in seguito alla ricezione di un altro segnale di 'start'. È inoltre opportuno notare che la specifica informa che un segnale di 'reset' viene trasmesso solamente prima del primo segnale di start, e non prima dei successivi. È quindi necessario implementare un reset dello stato del componente al termine dell'elaborazione della prima sequenza di parole; in questa implementazione ciò verrà fatto tramite l'utilizzo del segnale di 'done'. È richiesto che il componente il cui comportamento è descritto sopra abbia 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_data    : in std_logic_vector(7 downto 0);
            o_address : out std_logic_vector(15 downto 0);
            o_done    : out std_logic;
            o_en      : out std_logic;
            o_we      : out std_logic;
            o_data    : out std_logic_vector (7 downto 0)
      );
end project_reti_logiche;
```

Figura 1: Interfaccia del componente

2 Architettura

In questa sezione viene discussa l'architettura del componente realizzato.

2.1 Descrizione generale

Il componente è stato progettato secondo una struttura a blocchi che riflette i passaggi da effettuare sequenzialmente descritti nella sezione 1. L'architettura è divisa in 4 blocchi principali disposti come in Figura 2.
Il modulo di gestione della memoria si occupa di compiere le operazioni di lettura e scrittura in memoria di cui ai punti 1, 2 e 6 della specifica descritta nell'introduzione di questo documento.
Il modulo serializzatore a 8 bit si occupa di produrre un flusso da 1 bit restituendo uno ad uno i bit della parola fornita in ingresso.
Il modulo convoluzionale si occupa di applicare il codice convoluzionale richiesto da specifica allo stream in ingresso.
Il successivo modulo parallelizzatore restituisce una parola da 8 bit composta a partire dal flusso in ingresso, i dettagli vengono discussi nella sezione 2.5.
L'architettura e la funzione dettagliata dei vari componenti sono trattate nelle apposite sezioni.

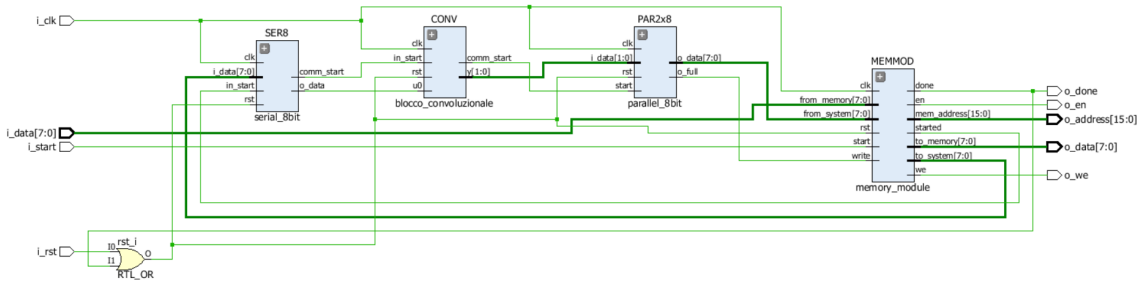

Figura 2: Diagramma schematico dell'architettura del componente

2.1.1 Segnale di reset

Siccome il componente riceve il segnale di reset solamente prima del primo segnale di start e non prima di altri segnali di start successivi ed eventuali, è necessario che tutti i moduli vengano resettati al termine delle operazioni. Per fare ciò si utilizza come segnale di reset un segnale interno ottenuto tramite un 'OR' logico del segnale esterno di reset e del segnale di done. Questo ultimo è un segnale prodotto dal modulo di gestione della memoria quando viene scritta in memoria l'ultima parola prodotta dall'elaborazione.

2.2 Modulo di gestione memoria

Il modulo di gestione della memoria esegue le operazioni di lettura e scrittura in memoria.
Questo modulo è realizzato tramite un process che implementa una macchina a stati e una parte di registri. La macchina a stati è schematizzata in figura 3. I registri memorizzano invece tutte le variabili utili a realizzare il funzionamento della macchina (contatori, salvataggio del valore in ingresso, memorizzazione degli indirizzi di memoria, salvataggio del flag di scrittura e della parola da scrivere).

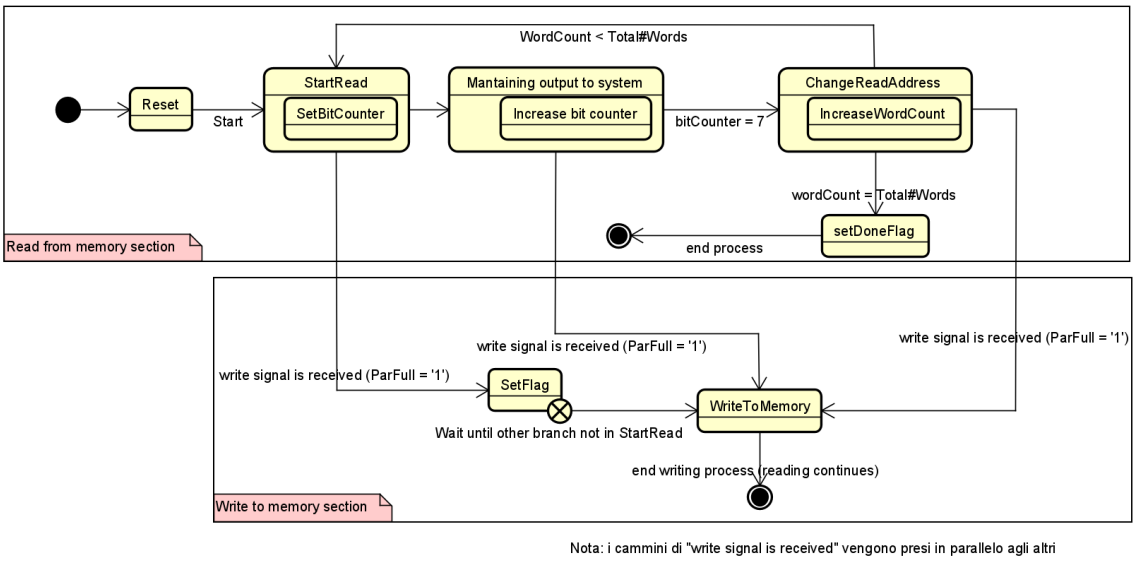

Figura 3: Diagramma semplificato della macchina a stati finiti del modulo di gestione della memoria

2.2.1 Funzionamento dettagliato

In questo paragrafo vengono dettagliati i compiti svolti dagli stati della machhina mostrata in figura 3.

- **reset**: il contatore del numero di parole scritte, il contatore di bit, le variabili che memorizzano gli indirizzi di lettura e scrittura correnti vengono resettati. Il numero di parole da 8 bit da leggere viene letto da memoria e conservato in un'apposita variabile;

- **start read**: la parola da elaborare viene letta da memoria e memorizzata nell'apposita variabile e trasmessa in uscita al sistema, l'indirizzo di lettura viene incrementato di 1 e l'inizio della lettura viene comunicato al modulo successivo tramite l'apposito pin. Il contatore dei bit viene posto a 0;
- **mantaining output to system**: la parola letta e memorizzata viene mantenuta fissa in uscita verso il sistema, questo rende possibile accedere alla memoria per altre operazioni. Il contatore dei bit viene aumentato di 1 ad ogni ciclo di clock;
- **change read address**: viene incrementato il contatore del numero di parole lette e viene calcolato il nuovo indirizzo di memoria da cui leggere;
- **set done flag**: il valore dell'uscita 'done' viene portato a '1';
- **set flag**: il flag di 'parola in attesa di essere scritta' viene posto a '1', in attesa che la memoria sia libera, e la parola da scrivere viene memorizzata nell'apposita variabile;
- **write to memory**: qualora il flag di scrittura (di cui al punto precedente) sia '0' la parola da scrivere in ingresso viene scritta in memoria, altrimenti (nel caso in cui il flag sia '1') viene scritta in memoria la parola memorizzata nello stato 'SetFlag' e il flag viene riportato a '0'.

2.3 Modulo di serializzazione

Il serializzatore ad 8 bit è realizzato tramite un process che implementa una macchina a stati e una parte di registri. I quattro registri memorizzano in modo sincrono la parola in ingresso, un valore di indirizzamento compreso tra 0 e 7, il bit in uscita e il valore del segnale di start da trasmettere in output. La transizione tra gli stati della macchina a stati avviene qualora vi sia il segnale di start posto a '1' ogni volta che si ha il rising edge del segnale di clock. La macchina a stati presenta uno stato di reset in cui viene portato a zero il valore di ogni bit di ogni registro. Gli altri stati della MSF corrispondono ognuno ad un valore i del registro numerico di indirizzamento (7-0) e viene dato in ingresso al registro di output l'i-esimo bit del valore memorizzato nel registro (a 8 bit) di ingresso. Inoltre il registro di start viene posto a '1' nel secondo stato della MSF (corrispondente al valore di indirizzamento 7) e rimane stabile fino a quando non si torna nello stato di reset. In ogni stato il valore di uscita del registro di start e del registro di output vengono portati in uscita alle porte 'o_start' e 'o_data' rispettivamente.
Lo schematico della descrizione a Register Transfer Level di questo process è presentato in figura 4.

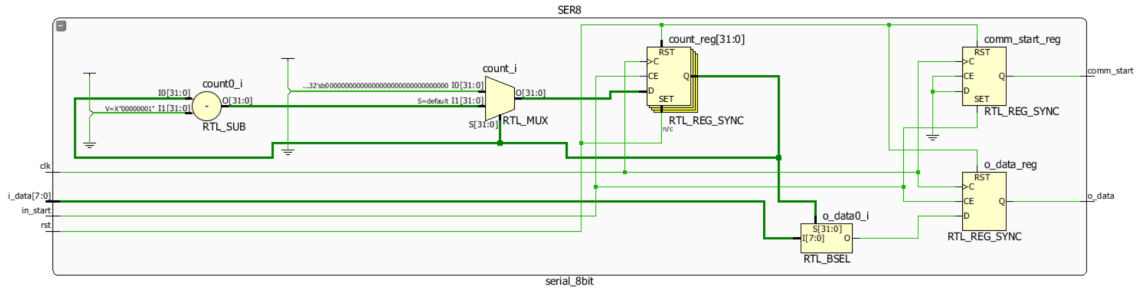

Figura 4: Diagramma schematico dell'architettura del modulo di serializzazione

2.4 Modulo convoluzionale

Il modulo convoluzionale implementa, tramite due flip-flop di tipo 'D' e tre porte logiche 'XOR' a due ingressi, la funzione convoluzionale descritta dalla macchina a stati finiti in figura 5. I segnali di clock e di reset sono direttamente collegati a clock e reset dei flip-flop e il segnale di start viene semplicemente passato attraverso e comunicato al modulo successivo in quanto questo componente non introduce ritardi significativi. Il segnale in input è passato ai due flip-flop in cascata. Il valore del primo bit di output è lo XOR del segnale di ingresso e dell'uscita del secondo flip-flop. Il valore del secondo bit di output è lo XOR del primo output e dell'uscita del primo flip-flop. La struttura interna del componente è descritta in figura 6.

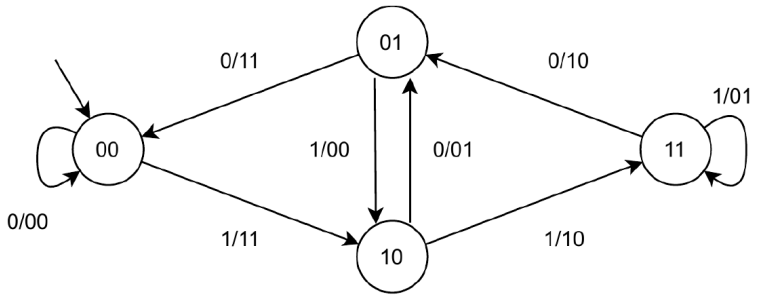

Figura 5: Diagramma della macchina a stati

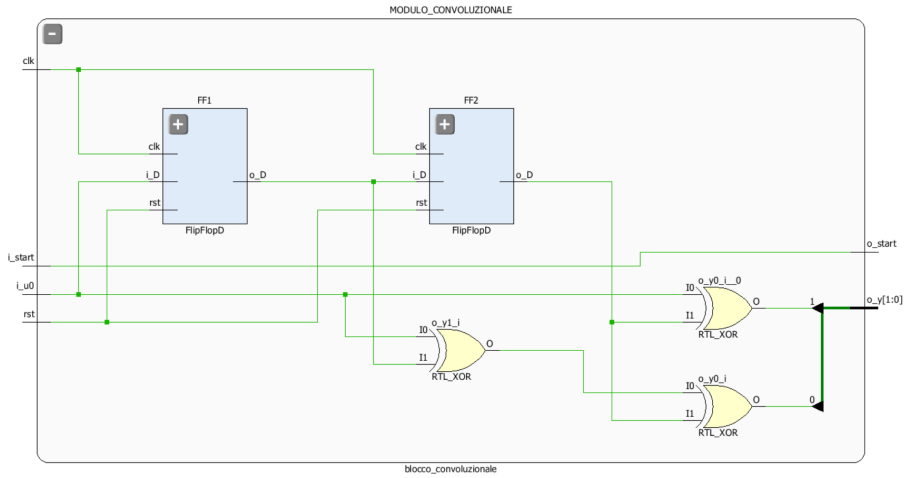

Figura 6: Diagramma schematico del modulo convoluzionale

2.4.1 Flip-Flop D

Il componente Flip-FlopD è un process che implementa la logica di funzionamento di un flip-flop di tipo 'D' con reset asincrono, restituendo ad ogni tempo di clock il valore ricevuto in input al tempo precedente. Il segnale di reset porta a '0' il valore memorizzato dal flip-flop in modo asincrono.

2.5 Modulo parallelizzatore

Il modulo parallelizzatore prende in ingresso i due bit prodotti dal modulo convoluzionale e svolge una serializzazione implicita (completando il funzionamento richiesto dalla descrizione dell'algoritmo convoluzionale data nella specifica), per poi parallelizzare questo flusso di bit in una parola da 8 bit. Per serializzazione implicita in questa descrizione si intende che i 2 bit in ingresso vengono trattati astrattamente come se fossero un flusso da 1 bit ottenuto alternando i due bit di ingresso, ma questa operazione non è effettuata esplicitamente. Un contatore, decrementato di 2 ogni ciclo di clock, indica il bit x della parola di uscita su cui riportare il primo dei bit in ingresso, il secondo viene riportato al bit x-1.
Si è deciso di sviluppare il componente in questo modo e di evitare la serializzazione implicita in quanto questa avrebbe introdotto la necessità di far funzionare i moduli a valle con una frequenza dimezzata rispetto a quelli a valle del serializzatore a 2 bit, raddoppiando il tempo necessario all'elaborazione e incrementando notevolmente il numero di transistor necessari a parità di funzionalità realizzata.
Quando il contatore (originariamente x=7) raggiunge l'1 (vengono scritti i due bit di ingresso sui bit 1 e 0 della parola di uscita) il segnale di 'full' in uscita da questo modulo viene portato a '1' (per tutti gli altri valori del contatore il segnale è portato a '0').
Il segnale di reset in ingresso porta al valore di default (x=7) il contatore interno di questo modulo e resetta i registri.

3 Risultati sperimentali

In questa sezione sono trattati i risultati di sintesi e dei test effettuati sul componente descritto nelle precedenti sezioni.

3.1 Sintesi

Il componente viene sintetizzato correttamente. Le tabelle 1 e 2 sono estratte dall'utilization report generato in fase di sintesi da Xilinx Vivado®.

Site Type	Used	Fixed	Available	Util%
LUT as Logic	609	0	41000	1.49
LUT as Memory	0	0	13400	0.00
Register as Flip Flop	290	0	8200	0.35
Register as Latch	0	0	82000	0.00
F7 or F8 Muxes	0	0	0	0.00

Tabella 1: Slice logic

Si nota la positiva assenza di latch, la cui presenza (se non voluta esplicitamente) potrebbe causare problemi di sincronizzazione all'interno del componente.

Total	Clock Enable	Synchronous	Asynchronous
43	Yes	-	Reset
15	Yes	Set	-
232	Yes	Reset	-

Tabella 2: Registers by type

La sintesi produce un messaggio di 'warning' che avvisa che gli elementi dal 16 al 31 del read_address_register e del write_address_register del modulo di gestione di memoria non sono utilizzati e sono stati rimossi dal componente. Questo è in linea con le aspettative in quanto gli indirizzi memorizzati in questi registri assumono un valore costante in corrispondenza di questi bit. Questo messaggio è perciò da considerare del tutto benigno.

3.2 Simulazioni

Tutte le simulazioni di cui di seguito sono state effettuate sia in pre-sintesi che in post-sintesi, i dati presentati qui di seguito (in particolare tempo di esecuzione e memory usage) fanno riferimento alla 'post-synthesis functional simulation'. I risultati delle simulazioni presentati in questa sezione sono stati ottenuti con un tempo di clock pari a 100 ns. Ciononostante, il componente è stato testato anche con tempi di clock inferiori e ha eseguito correttamente tutti i test a cui è stato sottoposto con un tempo di clock fino a 14 ns.

Nota: In tutte le descrizioni dei test effettuati, nella tabella dei valori attesi la dicitura 'altri' non si riferisce a tutti gli altri indirizzi in memoria ma a tutti gli altri indirizzi di scrittura (i valori agli indirizzi di lettura sono lasciati invariati).

3.2.1 Testbench 1: Sequenza minima

Questo test è stato effettuato per verificare il corretto funzionamento del componente in caso di sequenza di input di lunghezza minima (lunghezza nulla).

Indirizzo	Valore
0	0
altri	0

Tabella 3: Valori iniziali

Indirizzo	Valore
1000	0
1001	0
altri	0

Tabella 4: Valori attesi TB1

Test passato	SI
Tempo di esecuzione	1750100 ps
Memory peak	1557.820 MB

Tabella 5: Risultati TB1

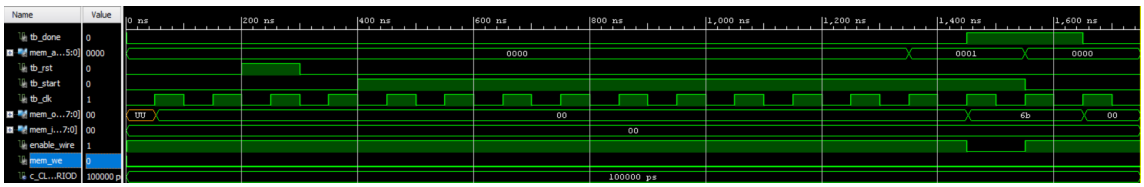

Figura 7: Forme d'onda dei segnali ai morsetti TB1

3.2.2 Testbench 2: Sequenza massima

Questo test rappresenta il duale di quello descritto nella sezione 3.2.1 e verifica il corretto funzionamento del componente in caso gli venga richiesto di elaborare il numero massimo possibile di parole in ingresso (255, corrispondente a RAM(0)="11111111"). La verifica è stata effettuata utilizzando la testbench tb_seq_max fornita con la specifica del progetto. Il componente ha superato il test, le tabelle dei valori non sono però qui riportate dato che l'alto numero di elementi renderebbe la sezione di difficile lettura.

Test passato	SI
Tempo di esecuzione	206150100 ps
Memory peak	1397.227 MB

Tabella 6: Risultati TB2

3.2.3 Testbench 3: Doppio reset

Questo test è stato effettutato per verificare il corretto funzionamento del componente qualora un secondo segnale di reset venga trasmesso prima del termine dell'elaborazione.

Indirizzo	**Valore**
0	2
1	67
2	13
altri	0

Tabella 7: Valori iniziali

Indirizzo	**Valore**
1000	55
1001	14
1002	176
1003	232
altri	0

Tabella 8: Valori attesi TB3

Test passato	SI
Tempo di esecuzione	3750100 ps
Memory peak	1782.688 MB

Tabella 9: Risultati TB3

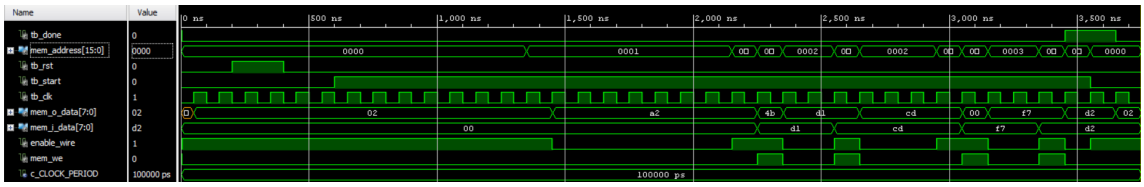

Figura 8: Forme d'onda dei segnali ai morsetti TB3

3.2.4 Testbench 4: Seconda sequenza

Questo test è stato effettutato per verificare il corretto funzionamento del componente qualora venga richiesto di effettuare una nuova codifica in seguito al completamento delle operazioni su un primo set di parole. Per verificare i risultati sono state usate due memorie per l'ingresso e due memorie per l'uscita.

Indirizzo	Valore
M0-0	2
M0-1	162
M0-2	75
M0-altri	0
M1-0	2
M1-1	112
M1-2	164
M1-altri	0

Tabella 10: Valori iniziali TB4

Indirizzo	Valore
M0-1000	209
M0-1001	205
M0-1002	247
M0-1003	210
M0-altri	0
M1-1000	57
M1-1001	176
M1-1002	209
M1-1003	247
M1-altri	0

Tabella 11: Valori attesi TB4

Test passato	SI
Tempo di esecuzione	1352600 ps
Memory peak	1579.887 MB

Tabella 12: Risultati TB4

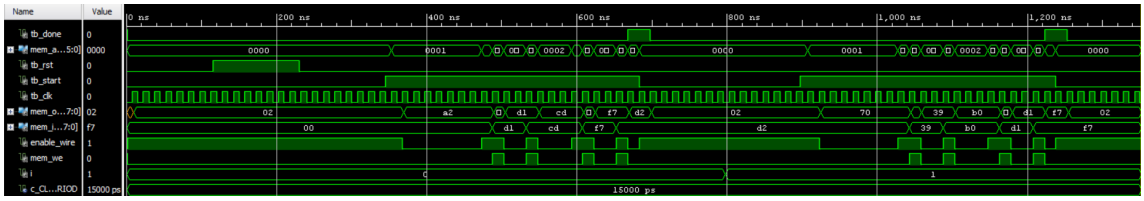

Figura 9: Forme d'onda dei segnali ai morsetti TB4

3.2.5 Testbench 5: Caso base

Al contrario dei test menzionati in precedenza questo non verifica il corretto comportamento del componente in un caso limite, ma si limita a verificare il corretto funzionamento di base del componente su una singola sequenza. Si approfitta della descrizione di questo caso di test molto semplice per mostrare al lettore un grafico di forme d'onda che include una selezione di segnali interni al componente che potrebbero essere utili al lettore qualora volesse osservare il funzionamento dei alcuni moduli sviluppati.

Indirizzo	Valore
0	2
1	162
2	75
altri	0

Tabella 13: Valori iniziali

Indirizzo	Valore
1000	209
1001	205
1002	247
1003	210
altri	0

Tabella 14: Valori attesi TB5

Test passato	SI
Tempo di esecuzione	3750100 ps
Memory peak	1782.688 MB

Tabella 15: Risultati TB5

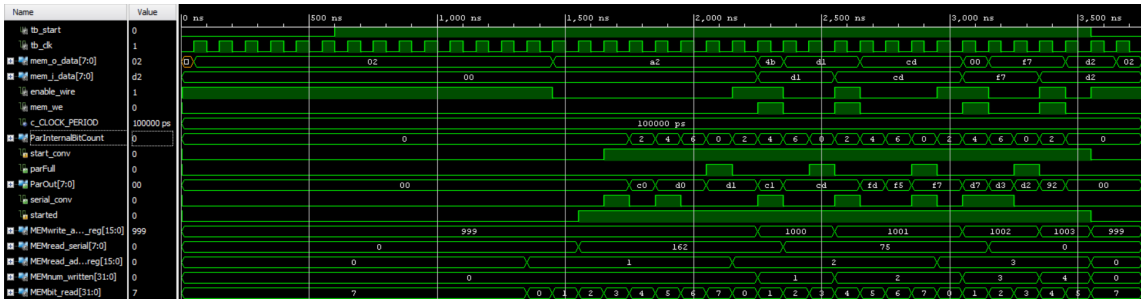

Figura 10: Forme d'onda dei segnali TB5

4 Conclusioni

Il componente è sintetizzabile e si comporta come da specifica sia in simulazione pre-sintesi che in simulazione-post sintesi (functional post-synthesis simulation), superando tutti i test a cui è stato sottoposto in tempi ragionevoli. Xilinx Vivado® non segnala errori o avvisi che possano compromettere il componente. L'approccio modulare utilizzato per lo sviluppo si è rivelato efficace e permetterà il riutilizzo dei moduli dovesse questa necessità manifestarsi in futuro. Durante lo sviluppo sono state riscontrate delle problematiche riguardanti la sincronizzazione dell'istante di inizio elaborazione dei moduli che costituiscono il sistema, questo problema è stato risolto introducendo una 'catena di start' attraverso cui ogni componente comunica a quello successivo quando iniziare ad elaborare i dati in ingresso.
In conclusione mi sento di considerare il risultato di questa prova finale positivo sia dal punto di vista del componente risultante che da quello formativo.