Prova Finale (Progetto di Reti Logiche)

Prof. William Fornaciari - A.A. 2021/2022

Agatino Bonanno - Codice Persona: 10618216
Teodoro Arioli - Codice Persona: 10685275

Indice

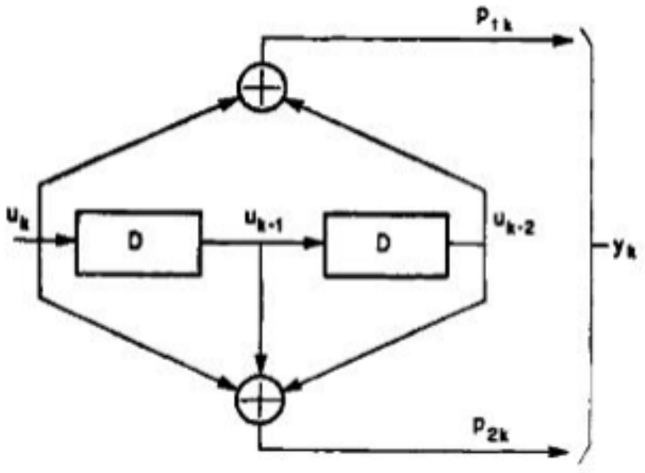
1 Introduzione

1.1 Finalità del progetto

L'obbiettivo del progetto è la specifica in VHDL di un modulo hardware che, presa in ingresso una sequenza di W parole (di lunghezza 1 Byte ciascuna), serializzi ognuna di esse generando un flusso U di singoli bit: questo verrà elaborato da un *codificatore convoluzionale* con tasso di trasmissione $\frac{1}{2}$ che genererà due nuovi bit da ciascuno dei bit in ingresso; la concatenazione di ognuna di queste coppie di bit in uscita sarà il flusso Y.
La sequenza Z di uscita altro non sarà che la parallelizzazione su 8 bit del flusso Y: questo stream Z verrà poi salvato su una memoria come sequenza di parole sempre da 8 bit (di conseguenza queste saranno sempre quantitativamente il doppio di quelle in ingresso, 2*W).

1.2 Descrizione generale

Il *convolutore* è una macchina sequenziale sincrona che prendendo in ingresso un singolo bit u_k produce due bit p_1k, p_2k, grazie all'elaborazione permessa dai due flip-flop D e le due porte XOR.



Un esempio di funzionamento è il seguente:

- Il Byte in ingresso è 10100010: il bit più significativo è il primo ad essere serializzato, seguito dal secondo, dal terzo, ecc. fino al meno significativo.

- Ogni bit viene serializzato ad ogni ciclo di `CLOCK`: u_k sarà quindi = 1 al primo instante T = 0; all'istante successivo T = 1, $u_k = 0$; a T = 2, $u_k = 1$ ecc. Schematizzando in una tabella in cui ogni colonna rappresenta un istante di `CLOCK` ben preciso, otteniamo come risultato il seguente flusso Z di uscita:

T	0	1	2	3	4	5	6	7
Uk	1	0	1	0	0	0	1	0
P1k	1	0	0	0	1	0	1	0
P2k	1	1	0	1	1	0	1	1

Z sarà ottenuta dal concatenamento dei valori di p_1k e p_2k, quindi:

$$\mathrm{Z} = p_1k(\mathrm{T}=0) + p_2k(\mathrm{T}=0) + p_1k(\mathrm{T}=1) + p_2k(\mathrm{T}=1) + ...$$

(dove per '+' si intende l'operazione di concatenamento).
Otterremo così Z = 1101000111001101.

- Infine il flusso Z verrà parallelizzato su 8 bit ottenendo due barole da 1 Byte ciascuna, 11010001 e 11001101, pronte per essere scritte in memoria.

1.3 Comportamento

Il modulo comincia la sua elaborazione al sollevamento del segnale `START` al valore 1. Questo segnale rimarrà alto fino al sollevamento al valore 1 del segnale `DONE`, quando l'intera computazione (più la scrittura in memoria) sarà completata. Il segnale `DONE` rimarrà alto finchè il segnale `START` non verrà portato a 0, ma un nuovo segnale di `START` non potrà essere dato finchè il `DONE` relativo alla precedente computazione risulterà ancora a 1.

Il convulatore verrà resettato automaticamente dal modulo al termine di ogni computazione: solamente la prima codifica richiederà il sollevamento da parte del TestBench del segnale `RESET` prima di procedere con l'avvio tramite `START` = 1.

1.4 Interfaccia

L'interfaccia fornita per il componente è la seguente:

```
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;
```

In particolare:

- il nome del modulo deve essere `project_reti_logiche`
- `i_clk` è il segnale di `CLOCK` in ingresso generato dal TestBench;
- `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_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- `o_en` è il segnale di `ENABLE` da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we` è il segnale di `WRITE ENABLE` da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

1.5 Memoria

Il modulo lavora tramite la lettura e la scrittura dei valori da/su una memoria indicizzata al Byte con indirizzi da 16 bit: ogni singola parola di memoria è di 8 bit di lunghezza.

- La cella di indirizzo 0000000000000000 conterrà il numero W di parole da codificare
- Le singole parole che andranno serializzate nel flusso U saranno contenute a partire dall'indirizzo 0000000000000001.
- Il flusso Z di uscita dovrà essere salvato come sequenza di parole di lunghezza 1 Byte a partire dalla cella di indirizzo 1000 (0000001111101000 in binario).

L'accesso in memoria sarà dato dal sollevamento del segnale `ENABLE`, sia che si tratti di lettura o scrittura: la distinzione tra i due casi sarà data dal valore del segnale `WRITE ENABLE`: 1 per abilitare la scrittura, 0 per permettere la sola lettura.

2 Architettura

2.1 Datapath

Il modulo è stato progettato considerando le diverse funzionalità da implementare come "sottomoduli", ognuno dedito ad una specifica operazione, interlacciati dai vari bus e pilotati dall'automa a cui si rimanda la descrizione nel paragrafo **2.2 Descrizione degli stati**.

Contatore di indirizzi

Questa sezione è stata progettata come un contatore allo scopo di tenere salvato nel corso della computazione l'indirizzo di memoria al momento in lettura. La componenti sono:

- un registro a 16 bit chiamato `addr_reg` contenente l'indirizzo della cella di memoria da leggere, pilotato dai segnali `CLOCK`, `RESET` (quest'ultimo forza lo stato interno al valore 0000000000000000, ovvero l'indirizzo della prima cella da leggere) e dal segnale `addr_reg_load` che permette il salvataggio dei valori all'ingresso dello stesso.
  L'uscita del registro è resa disponibile sul bus a 16 bit `o_addr_reg`;
- un sommatore a 16 bit con ingressi il valore fisso 1 (esteso su 16 bit) e `o_addr_reg`: l'uscita del sommatore è direttamente collegata all'ingresso del registro, fornendo così a questo il valore della prossima cella di memoria da leggere (incrementando di 1);
- un comparatore che verifica se il valore di uscita del registro sia 0 o altro: il fine di questo componente è spiegato nella sezione **Contatore di parole**.

Calcolatore di indirizzi

Questa sezione è speculare alla precedente, relativamente però al calcolo degli indirizzi di memoria su cui scrivere i valori computati. È composta da un identico registro a 16 bit (il quale però verrà forzato dal segnale di `RESET` al valore 0000001111101000) e, sempre come la precedente, da un sommatore che prende in ingresso il valore dal bus `o_mem_reg`, uscita del registro, e dopo aver sommato ad esso 1, fornisce il risultato in ingresso allo stesso registro come nuovo valore, che verrà caricato solo al sollevamento a 1 del segnale `o_mem_reg_load`.
I due bus di uscita dai registri sono poi connessi al bus `o_address` tramite un MUX pilotato dal segnale `o_addr_select` cosicché venga caricato esclusivamente uno dei due indirizzi relativi alla lettura o alla scrittura.

Contatore di parole

Questa sezione riprende lo schema delle due precedenti ma con differenze più marcate dettate dal fatto che il valore da salvare all'inizio della computazione è quello presente nella prima cella di memoria: è infatti questa sezione che conta la sequenza di parole e ferma l'esecuzione al loro termine. Le componenti sono:

- un registro a 8 bit `n_word_reg` pilotato dal segnale `n_word_reg_load` la cui uscita, `o_n_word_reg`, è collegata ad un sottrattore;
- un sottrattore, che sottrae sempre un'unità al valore presente su `o_n_word_reg`;
- un MUX avente due ingressi (l'uscita del sottrattore e `i_data`) pilotato dal segnale generato dal comparatore della sezione **Calcolatore di indirizzi**, in modo che solo quando l'indirizzo attualmente in lettura sia 0000000000000000 allora venga salvato il suo contenuto (presente su `i_data` nel registro `n_word_reg`, altrimenti viene sempre aggiornato il suo contenuto con il numero corrente di parole non ancora lette;
- un comparatore all'uscita del registro: quando il suo contenuto è 00000000 viene sollevato il segnale `end_words` che avvia la fine della computazione.

Serializzatore e convolutore

Questa sezione si occuppa effettivamente della serializzazione del flusso U e del successivo calcolo del flusso Y. Le componenti sono:

- un MUX pilotato da `data_load` che permette il l'ingresso del contenuto di `i_data` quando è a 1, mentre di un generico valore 00000000 quando è a 0, in modo da resettare il valore dei due flip-flop D del convolutore;
- un MUX a 8 ingressi che serializza `i_data`, pilotato dal segnale a 3 bit `o_j_reg_select` in uscita dal registro `j_reg`;
- un registro a 3 bit detto `j_reg` pilotato dal segnale `j_reg_load` il quale è sia segnale di `LOAD` sia (posto in NOT) segnale di `RESET`: il registro quando attivato fornisce in uscita un numero tra 0 e 7 incrememntandolo per ogni ciclo di clock; si ottiene così la funzione di serializzazione dell'ingresso del MUX;
- un sommatore a 3 bit che prende in ingresso l'uscita di `j_reg` e, sommato 1, fornisce il nuovo valore all'ingresso dello stesso registro;
- il convolutore, il cui scopo e funzionamento è stata già ampiamente discussa nel paragrafo **1.2 Descrizione generale**, i cui flip-flop sono controllati anche dal segnale `automa_clk`, in AND con il segnale `i_clk`: questo permette (analogamente, come si vedrà, al caso di `reg16`) di "congelare" il loro contenuto nelle fasi finali della computazione dedite alla scrittura in memoria.

Registro di uscita

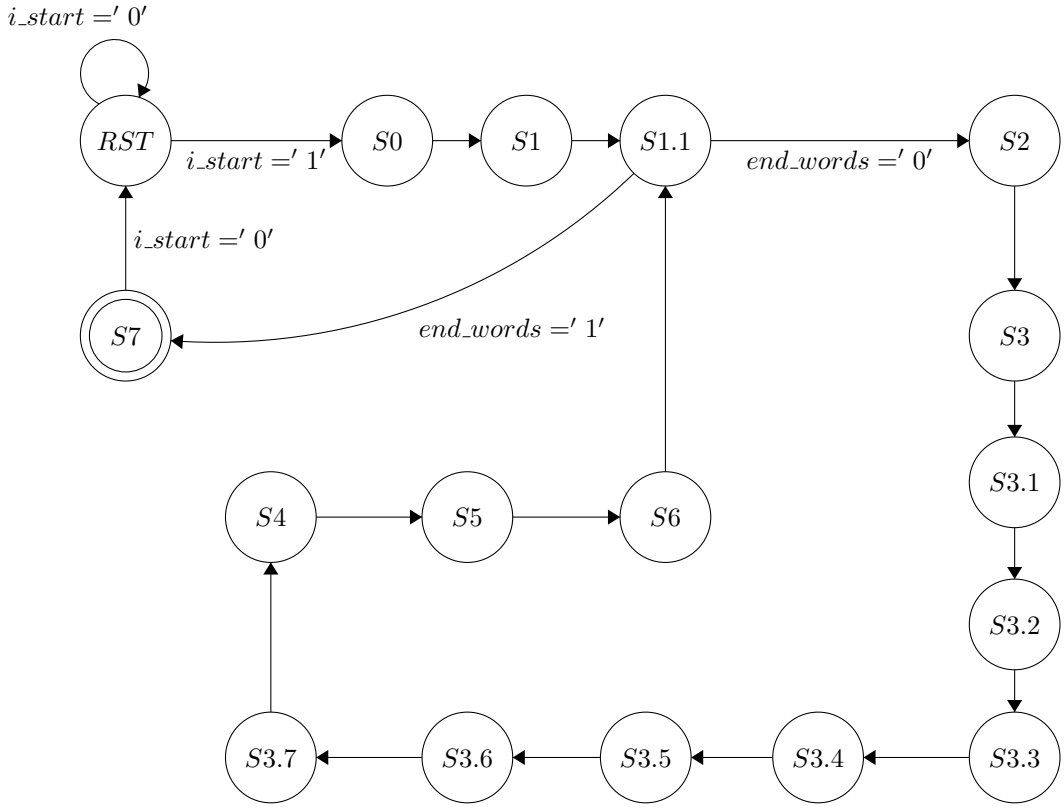
Il registro a 16 bit `reg16` prende in ingresso i due valori p_1k e p_2k e li aggiunge al valore precedentemente salvato come bit meno significativi, facendo uno shift di due posizioni di tutti gli altri bit salvati. Il processo è descritto dalla formula ricorsiva:

```
process(i_clk, out_shift)
begin
   if(i_clk'event and i_clk = '1' and out_shift = '1') then
      o_reg16 <= o_reg16(13 downto 0) & p1k & p2k;
   end if;
end process;
```

Il segnale `out_shift` pilota il `CLOCK` del registro: solo quando entrambi sono alzati il registro prosegue con lo shifting, "congelando" di fatto il suo stato non appena viene abbassato a 0.

2.2 Descrizione degli stati

L'automa è stato pensato per avanzare ad ogni ciclo di `CLOCK`: è quindi stato omesso questo segnale da ogni arco del grafo esclusivamente per motivi di leggibilità.



Stato RST

Stato di entrata usato per resettare tutti i segnali e registri al valore iniziale così da rendere il modulo pronto all'elaborazione.

Stato S0

Stato in cui l'automa si sposta dopo aver ricevuto il segnale di inizio elaborazione. Qui viene alzato il segnale `o_en` abilitando la lettura della memoria e ottenendo così il numero delle parole W che verranno lette nell'elaborazione.
Segnali modificati:
`o_en` = '1'

Stato S1

Stato in cui viene abilitata la modifica dei registri contenenti il numero delle parole rimanenti e i registri contenenti gli indirizzi di memoria di input e output.
Segnali modificati:
`addr_reg_load` = '1',
`n_word_reg_load` = '1'

Stato S1.1

Stato in cui viene controllato se il numero delle parole rimanenti è uguale a 0. In caso affermativo l'automa salta direttamente allo stato finale S7

Stato S2

Stato in cui inizia il ciclo di esecuzione e dunque l'elaborazione. Qui viene alzato il segnale di lettura `o_en` così da rendere disponibile nel successivo stato la parola da computare
Segnali modificati:
`o_en` = '1'

Stato S3 - S3.7

Stati in cui avviene l'effettiva trasformazione della parola letta in memoria andando a scrivere l'uscita in un registro. Per ogni bit in entrata avremo 2 bit in uscita.
Segnali modificati:
`automa_clk` = '1',
`data_load` = '1',
`j_reg_load` = '1',
`out_shift` = '1'

Stato S4

Stato in cui il risultato della precedente elaborazione viene assestato e reso disponibile al salvataggio in memoria.

Stato S5

Stato in cui viene presa la prima parte di output generata (gli 8 bit più significativi) per essere scritta in memoria.
Segnali modificati:
`o_mem_reg_load` = '1',
`o_en` = '1',
`o_we` = '1',
`o_addr_select` = '1',
`load_second_part` = '0'

Stato S6

Stato in cui viene caricata in memoria la seconda parte di output generata. Qui viene anche controllato se sono state computate tutte le parole o se ne mancano altre. In caso di terminazione l'automa si sposta nello stato finale altrimenti riparte il ciclo (S2) andando a prendere la parola successiva a quella letta in precedenza.
Segnali modificati:
`o_mem_reg_load` = '1',
`o_en` = '1',
`o_we` = '1',
`o_addr_select` = '1',
`load_second_part` = '1'

Stato S7

Stato di accettazione dell'automa che indica la terminazione dell'elaborazione. Se il segnale `i_start` viene abbassato in seguito a un abbassamento del segnale di fine `o_done` allora l'automa si riposiziona nello stato di reset pronto per un'altra elaborazione.
Segnali modificati:
`o_done` = '1'

2.3 Ulteriori note sul `RESET`

La funzionalità del `RESET` è implementata da due segnali: `i_rst`, fornito dalla specifica e pilotato dal testBench, che oltre a effettuare la prima inizializzazione permette anche un `RESET` esterno e asincrono dalla computazione; l'altro segnale è `reset_state`, interno e pilotato dall'automa, che permette la re-inizializzazione del modulo al termine di una computazione completata (`o_done = '1'`).

I due segnali di fatto svolgono la stessa funzione sui registri ma in due contesti completamente diversi (asincrono all'elaborazione il primo e successiva al completamento delle operazioni il secondo)

3 Risultati sperimentali

3.1 Sintesi

Di seguito i risultati della sintesi. Si può notare come nel circuito non vengano inseriti Latch che potrebbero causare comportamenti inaspettati e che le varie commutazioni rientrano largamente nel periodo del clock:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	52	0	0	134600	0.04
LUT as Logic	52	0	0	134600	0.04
LUT as Memory	0	0	0	46200	0.00
Slice Registers	77	0	0	269200	0.03
Register as Flip Flop	77	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

Fig 1 - Risultato sintesi: logica usata

```
Timing Report

Slack (MET) :             96.407ns  (required time - arrival time)
  Source:                 FSM_onehot_curr_state_reg[10]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Destination:            FSM_onehot_curr_state_reg[0]/CE
                            (rising edge-triggered cell FDPE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        3.211ns  (logic 0.999ns (31.112%)  route 2.212ns (68.888%))
  Logic Levels:           3  (LUT4=1 LUT5=1 LUT6=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

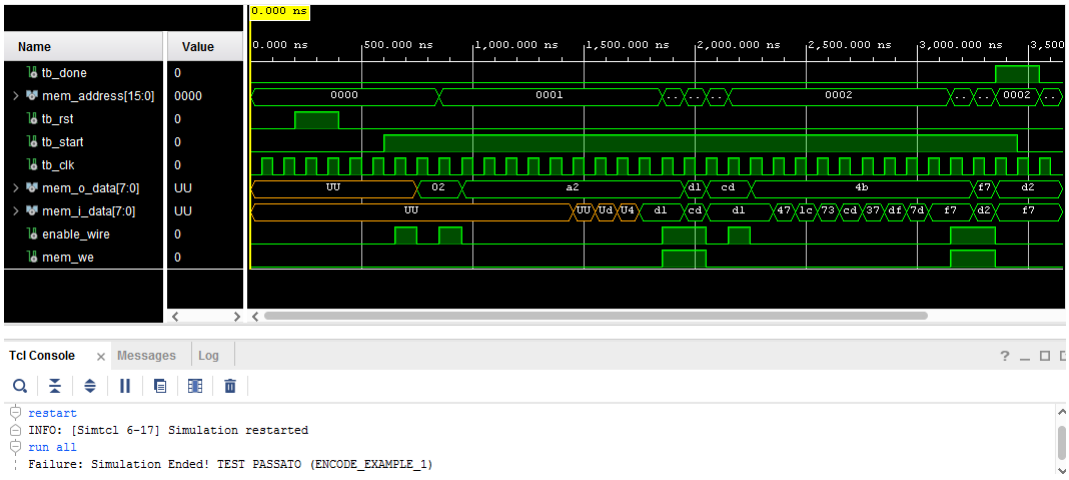
Fig 2 - Risultato sintesi: analisi timing

3.2 Simulazioni

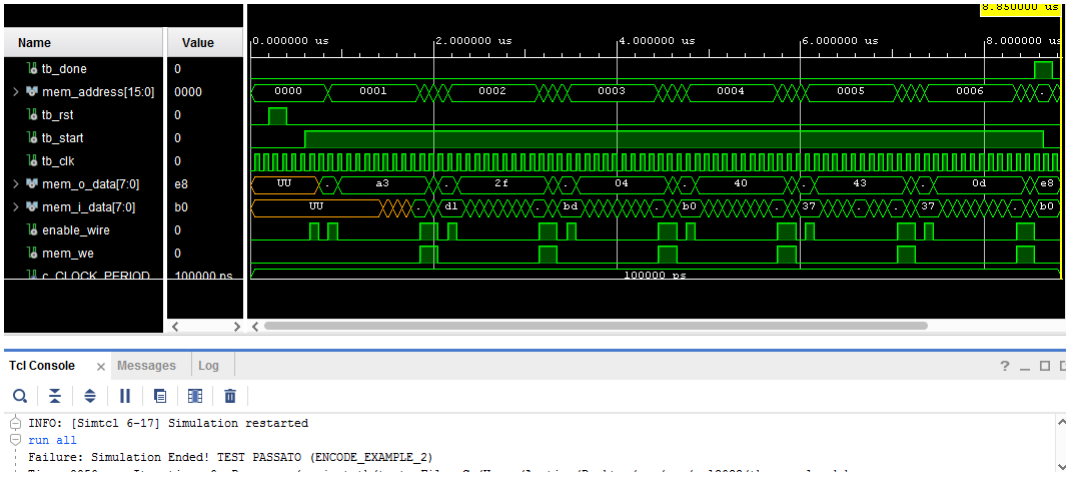
Tutti i seguenti test vengono correttamente superati sia in Behavioral sia in post sintesi.

Test forniti dal docente

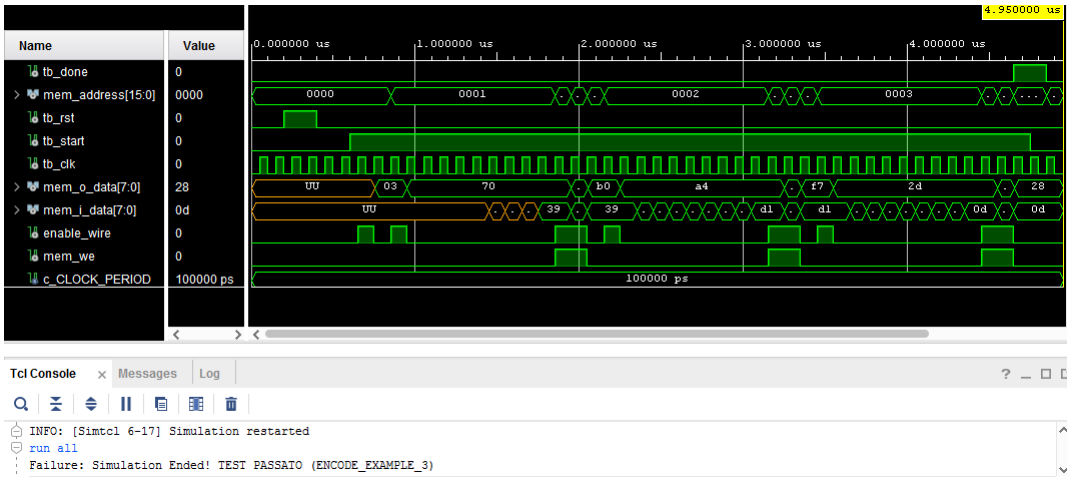
I qui presenti test sono quelli inseriti come esempi nella specifica del componente



Test 1



Test 2

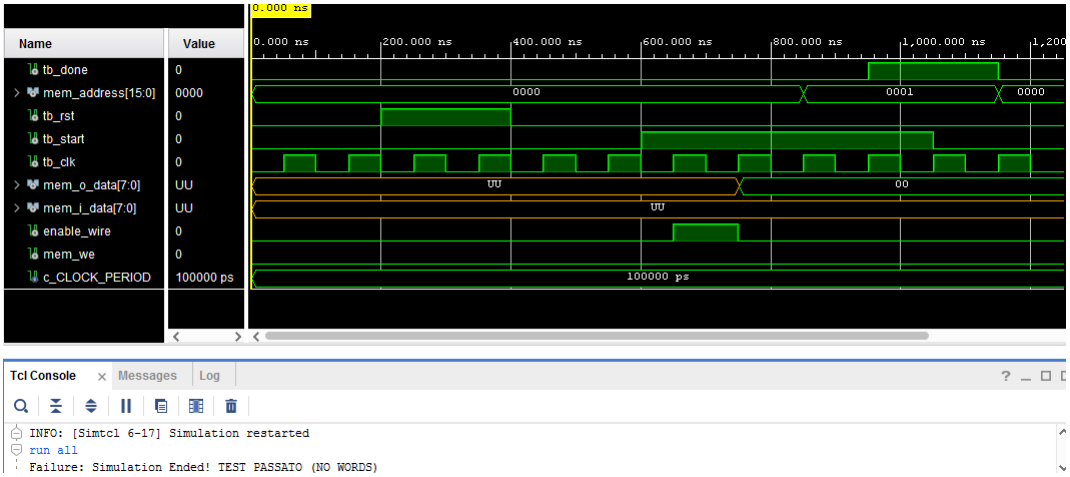


Test 3

Casi limite

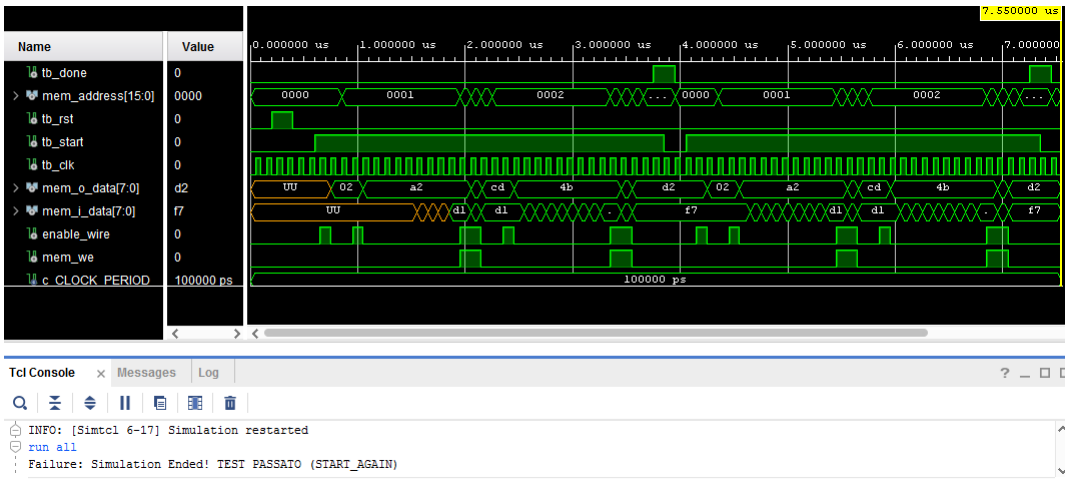
A seguire sono presenti dei casi limite, anche questi vengono correttamente eseguiti sia in Behavioral sia in post sintesi.

- Caso "no words":
  In posizione 0 della memoria è presente il valore 0 indicando che non ci sono parole da computare. L'output atteso è che non venga scritto nulla in memoria:

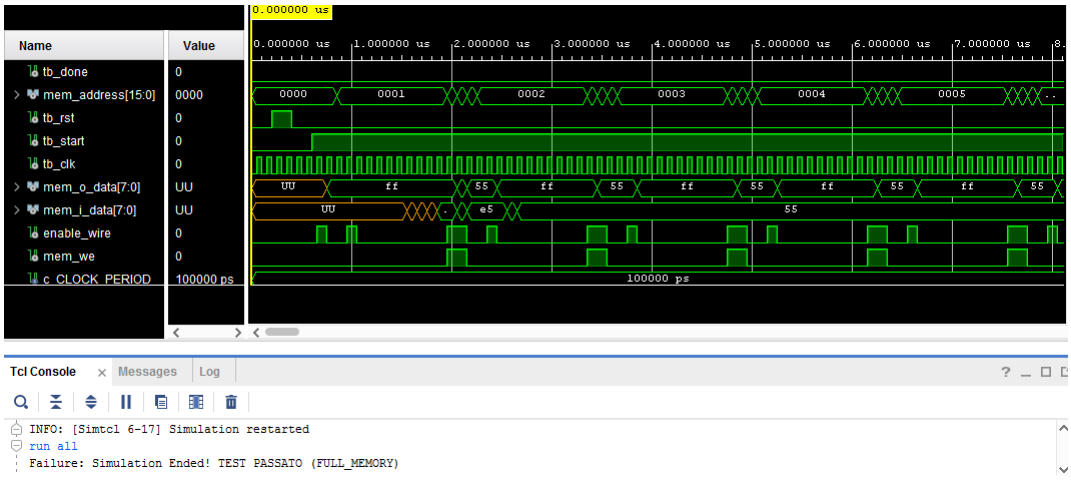


- Caso "start again":
  A fine computazione viene rialzato il segnale di start e si osserva che il

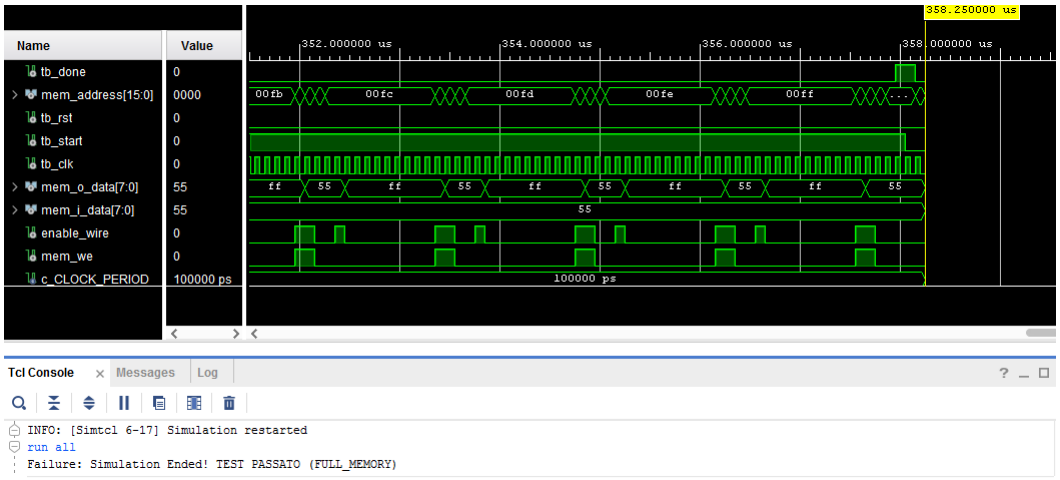
modulo riparte con la computazione senza bisogno del segnale `i_rst`:



- Caso "full memory":
  Si inserisce il valore massimo di parole memorizzabili (255) e si osserva che l'automa si comporta come previsto andando a scrivere in memoria le 510 parole correttamente:

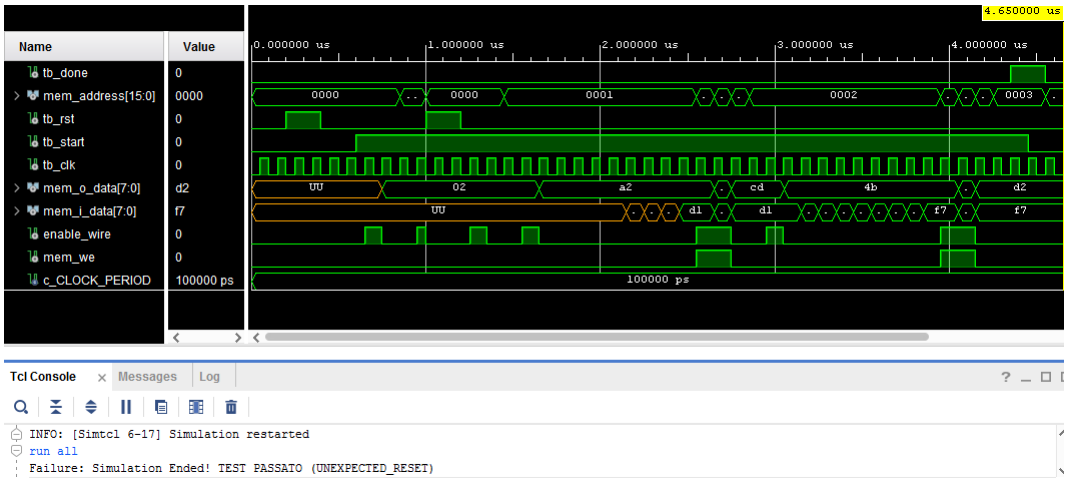


Parte iniziale



Parte finale

- Caso "Re-Reset":
  Si rialza il segnale di `i_rst` a metà computazione e si osserva che il corretto comportamento di ripartenza della computazione (essendo il segnale `i_rst` applicabile asicronicamente)



4 Conclusioni

Come evidente dal risultato delle prestazioni visibile nel paragrafo **3.1 Sintesi**, in particolare dalla voce `Slack`, il periodo del ciclo di `CLOCK` può essere ampiamente ridotto per migliorare la durata dell'esecuzione senza intaccare la corretta esecuzione dei processi.

Un'altra ottimizzazione possibile riguarda l'implementazione del convolutore: la scelta di design di inizializzare i due registri flip-flop D tramite un MUX pilotato dal segnale `data_load` è stata dettata solo dalla necessità di voler mantenere un diretto controllo sull'intero flusso di bit che attraversa quella sezione: lo stesso risultato sarebbe comunque ottenibile collegando il segnale `i_rst` OR `reset_state` ai due flip flop D per inizializzarli e abbassando a 0 il segnale `automa_clk` nel momento in cui u_k è uguale al bit meno significativo della parola presente su `i_data`.
Sebbene questa modifica non comporterebbe un miglioramento in termini di prestazioni temporali (in quanto già ora l'inizializzazione dei due registri viene fatta parallelamente ad altri processi senza rallentare), rappresenterebbe un piccolo miglioramento in termini di area, risparmiando l'impiego di un MUX nella sintesi del modulo.