Documentazione prova finale Reti Logiche

Studentessa Alessandra Pasini

Introduzione

Obiettivo principale e ultimo di questo documento è quello di illustrare le varie fasi effettuate per progettare, sviluppare ed implementare il componente hardware descritto nella specifica assegnataci durante il corso "Prova Finale di Reti Logiche" durante il primo semestre dell'a.s. 2017/2018.

Per raggiungere tale scopo, nelle prossime pagine, saranno sviluppate le seguenti macro-aree:

- Progettazione: questa prima sezione prevede l'analisi delle fasi che hanno portato allo sviluppo di una macchina a stati finiti.
- Implementazione: analisi puntuale del codice scritto per ciascuno stato dell'FSM (finite state machine).
- Testing: quest'ultimo paragrafo ha lo scopo di dimostrare l'effettivo funzionamento del componente non solo nei 4 test-bench proposti ma anche in molteplici casi limite e non.

Progettazione

Analisi della Specifica

In questa prima sezione si desira ripercorre le varie fasi dell'analisi della specifica.
Da tale documento è facile evincere quale sia la finalità del componente che si vuole progettare: data, in memoria, una matrice di N righe ed M colonne ed un valore soglia compreso tra 0 e 255 si vuole calcolare la più piccola area contenente valori uguali o maggiori alla soglia stessa.

La specifica precisa inoltre che l'analisi e, in origine, la lettura dei dati in memoria può avvenire solo dopo aver ricevuto un segnale di reset e, di seguito a questo, un segnale di start. Entrambi i valori, per risultare validi, devono restare alti, cioè devono essere pari a 1, per un intero ciclo di clock ciascuno.
La figura che si desidera ottenere corrisponde sempre ad un poligono, in particolare ad un rettangolo o ad un quadrato. La formula matematica che consente di raggiungere lo scopo richiesto è dunque

$$A = b \times h \quad \text{(area = base x altezza)}$$

Una volta trovata l'operazione geometrica corrispondente risulta immediato ipotizzare che sia necessario scorrere i dati dell'intera matrice per andare alla ricerca dei lati della figura. Per fare ciò si è pensato di salvare 4 segnali corrispondenti alle coordinate dei lati più esterni della figura con valore uguale o superiore a quello soglia.

Una volta letta l'intera immagine e ottenuti i valori delle 4 variabili l'operazione successiva risulta essere quella computazionale: si effettuano le sottrazioni per ottenere i lati, base ed altezza, e si moltiplicano i due valori tra loro per l'area. Sempre analizzando attentamente la specifica si deduce che questo è lo step conclusivo di una prima macro-azione che deve essere compiuta dal componente; trovando il valore dell'area si porta infatti a termine la parte di lettura ed analisi dei dati presenti in memoria.

La seconda macro-azione è quella legata alla scrittura, precisamente agli indirizzi 0x00 e 0x01 preventivamente lasciati liberi per facilitare quest'ultimo atto. Sono disponibili per il caricamento del risultato dell'area due celle in quanto la specifica è basata su numeri decimali che, trasformati in binario,

sono lunghi un byte (8bit). Come è noto, la moltiplicazione in binario tra due numeri composti da 8bit l'uno da, come risultato, un numero binario formato da tanti bit quanti sono quelli dati dalla somma dei bit dei due fattori (16bit). Ogni cella di memoria ha dimensione pari a un byte (8bit).
Proprio per questo motivo risulta necessario scindere il risultato della moltiplicazione in due numeri; gli 8bit meno significativi sono dunque caricati all'indirizzo 0x00 mentre quelli più significativi all'indirizzo 0x01.

Per segnalare la conclusione del processo il segnale di done, o_done, deve restare alto per un intero ciclo di clock. Quando tale segnale torna a 0 si deve ripristinare la situazione iniziale per garantire la corretta possibilità di intraprendere una nuova computazione. Si ricorda che si può ricevere un secondo segnale di start solo dopo aver terminato l'intero processo in atto, ovvero solo dopo la ricezione di un segnale o_done = 1.

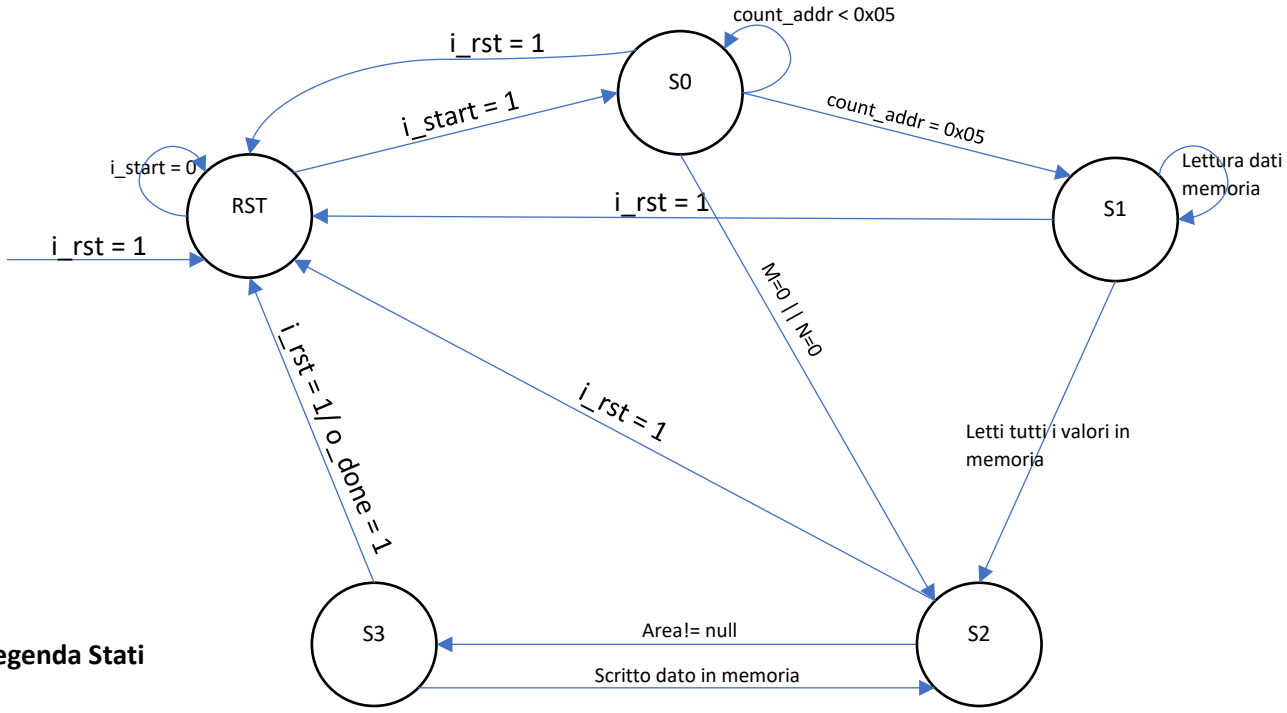
Scelte Implementative

Analizzando la traccia si è potuto osservare e si sono potute evidenziare delle fasi distinte e definite.
Da ciò si è dedotto che, nell'implementazione, si sarebbero potute seguire due strade differenti:

- Un approccio fondato sullo sviluppo di più sottocomponenti, ognuno con una precisa mansione
- Un approccio basato sulla logica sequenziale nella quale l'output non dipende solo dai valori in ingresso ma anche dallo stato in cui ci si trova.

Seppur entrambi siano dei validi metodi la scelta è ricaduta sulla logica sequenziale in quanto un FSM è dotato di memoria che consente il mantenimento non solo dello stato attuale ma anche di quello passato. Lo stato di uscita di un circuito sviluppato con logica sequenziale è funzione dell'input presente, di quello passato e/o dell'output passato. Un FSM salva questi dati e rimane nello stato corrente fintantoché un segnale di clock successivo non modifica uno di questi stati.
Il circuito sequenziale sviluppato è dunque un FSM sincrono, ovvero guidato dal segnale di clock, e ciclico in quanto ritorna nello stato di reset ogni qualvolta ci sia un segnale di reset alto o sia conclusa la computazione.

Di seguito è possibile vedere lo schema dell'FSM.



Legenda Stati

Studentessa: Alessandra Pasini
A.S.: 2017/2018

- RST: stato iniziale nel quale la macchina permane finché non si ha la ricezione di un segnale di start alto; una volta rilevato tale segnale si inizializzano tutti i valori necessari al corretto funzionamento del componente e si passa allo stato S0.
- S0: stato all'interno del quale vengono letti e salvati per mezzo di segnali i valori di colonne, righe e soglia; si passa allo stato successivo nel momento in cui il dato in ingresso è il primo elemento della matrice.
- S1: in S1 avviene la lettura dell'intera matrice; si passa a S2 quando sia il segnale contatore per le colonne che quello per le righe sono uguali alla dimensione M (colonne), N (righe) della matrice.
- S2: stato intermedio dove si calcola l'area e al termine del quale si passa all'ultimo stato.
- S3: al suo interno si ha la scrittura in memoria e conclusione del processo con il ritorno allo stato di RST.

Algoritmo per la lettura della matrice

Per la lettura dei dati in memoria si è scelto di utilizzare un algoritmo lineare ovvero un algoritmo che scorre per intero la matrice seguendo il posizionamento dei dati in memoria.
Si è ritenuto adatto sviluppare un algoritmo semplice che assecondasse il flusso dei dati impostato dalla memoria stessa.

Implementazione

Analisi del codice
Dopo aver illustrato i vari step antecedenti lo sviluppo, questa sezione si prefigge l'obiettivo di commentare alcuni punti salienti del codice scritto.

- L'entity corrisponde a quella definita all'interno delle specifiche, viene qui riportata per questioni di completezza.

```
37  entity project_reti_logiche is
38      port (
39              i_clk         : in  std_logic;
40              i_start       : in  std_logic;
41              i_rst         : in  std_logic;
42              i_data       : in  std_logic_vector(7 downto 0); --1 byte
43              o_address     : out std_logic_vector(15 downto 0); --16 bit addr: max size is 255*255 + 3 more for max x and y and thresh.
44              o_done            : out std_logic;
45              o_en         : out std_logic;
46              o_we       : out std_logic;
47              o_data            : out std_logic_vector (7 downto 0)
48          );
49  end project_reti_logiche;
50
```

- All'interno dell'architettura non solo sono stati inseriti i 5 stati dell'FSM ma anche tutti quei contatori e quelle variabili che si sono dimostrati necessari per il corretto funzionamento del componente.
  Sono tutti già commentati nel codice ma vorrei illustrare l'utilizzo di alcuni di questi.
  Alla riga 57 sono presenti i 4 valori individuati durante l'analisi della specifica, la loro inizializzazione avviene solo nello stato S0. Sx_col e high_row sono inizializzati a zero poiché rappresentano i valori bassi dell'area mentre dx_col e low_row sono posti pari alle dimensioni M, il primo, ed N, il secondo, della matrice.
  Per poter incrementare il valore dell'indirizzo, ovvero per poter effettuare delle operazioni sul segnale d'uscita o_address si è vista la necessità di inserire due segnali, rispettivamente count_addr alla riga 58 e i (incremento) alla riga 59. Ciò è dovuto al fatto che non è possibile modificare direttamente un valore d'uscita ma è obbligatorio il passaggio per variabili intermedie.

```
51  architecture FSM of project_reti_logiche is
52  type state_type is (RST, S0, S1, S2, S3);  --stati dell'FSM
53  signal next_state, current_state: state_type;
54  signal col,row,threshold: std_logic_vector(7 downto 0); --segnali atti a salvare le dimensioni della matrice e il valore soglia
55  signal count_col, count_row: std_logic_vector(7 downto 0);  --contatori utilizzati durante la lettura della matrice
56  signal lenght, high: std_logic_vector(7 downto 0);  --terminata la lettura della matrice si salvano qui le lunghezze di base e altezza
57  signal sx_col, dx_col, high_row, low_row: std_logic_vector(7 downto 0); --4 segnali atti a salvare gli estremi dell'immagine
58  signal count_addr: std_logic_vector(15 downto 0):= "0000000000000010";  --contatore per gli indirizzi in memoria
59  signal i:std_logic_vector(15 downto 0); --incremento dell'indirizzo
```

- L'FSM scritto è composto da due processi.
  Il primo, riportato qui accanto, è di tipo sequenziale e funge da elemento di memoria per quello che nello stato seguente sarà considerato lo stato presente.
  Nel caso specifico è stato scelto di inserire nella lista di sensitività solo il segnale di clock.
  La transizione tra lo stato corrente e lo stato prossimo è basata su un costrutto di tipo if-then-else.
  Vorrei sottolineare che alla riga 67 si pone l'uscita o_address pari al contatore, questa operazione viene effettuata ad ogni ciclo di clock. Fintantoché non si incrementa count_addr il valore dell'uscita corrisponde al primo indirizzo occupato in memoria ovvero 0x02.

```
66  begin
67      o_address <= count_addr;
68      state_reg : process(i_clk)
69      begin
70          if (i_clk'event and i_clk='1') then
71              if (i_rst = '1') then
72                  current_state <= RST;
73              else
74                  current_state <= next_state;
75              end if;
76          end if;
77      end process;
78
```

- Il secondo processo, di tipo combinatorio, ha lo scopo di calcolare lo stato prossimo e le varie uscite. Il processo preposto al calcolo dello stato successivo è basato su un costrutto di tipo case-when. Anche in questo caso l'unico segnale presente nella lista di sensitività è il clock, questo per evitare che le troppe variabili potessero generare conflitti ed errori ed anche in quanto, tale segnale, deve essere l'unico che può determinare l'effettiva modifica di stato o di segnali utilizzati durante il processo.

- Il primo stato presente nel costrutto case-when è RST, ovvero lo stato iniziale chiamato solo nel momento in cui si ha un segnale i_rst alto (i_rst =1).
  Tale stato è quello di attesa tra la messa in funzione del componente e l'inizio vero e proprio del processo. È caratterizzato da un costrutto if-then-else:
  -La prima condizione, a riga 85, è soddisfatta se si riceve in ingresso un segnale i_start =1; in caso affermativo si pone next_state pari allo stato prossimo S0 e si inizializzano tutte le variabili.
  L'inizializzazione è qui svolta per garantire la correttezza dei valori iniziali ogniqualvolta sia richiesta l'esecuzione di un nuovo processo o vi sia un segnale di reset durante il processo in atto.
  -La seconda serve come controllo nel caso in cui si torni allo stato di reset senza che ci sia stato un segnale i_rst = 1.
  -La terza lascia che lo stato prossimo permanga uguale a quello corrente e pone il segnale di controllo keep_going = 0 per segnalare l'interruzione del processo in atto.

```
79  delta_lambda: process(i_clk)
80  begin
81    if (i_clk'event and i_clk='1') then
82      case current_state is
83          when RST =>                     --stato di inizializzazione
84              if(i_start = '1') then      --il seganle di start alto indica l'inizio effettivo della computazione
85                  next_state <= S0;       --lo stato prossimo è S0
86                  o_en <= '1';            --porre l'uscita o_en alta fa si che possano essere effettuate delle operaioni sulla memoria
87                  o_we <= '0';            --l'uscita o_we = 0 segnala il fatto che si voglia leggere la memoria
88                  keep_going <='1';       --dalla riga 89 alla riga 98 si inizializzano vari segnali
89                  count_col <= "00000000";
90                  count_row <= "00000000";
91                  i <= "0000000000000001";
92                  count_addr <= "0000000000000010";
93                  inc <= "00000001";
```

- Il costrutto prosegue con lo stato S0 nel quale vengono letti i primi tre dati presenti in memoria:
  - il primo, posizionato all'indirizzo 0x02, è la dimensione M delle colonne;
  - il secondo, posizionato all'indirizzo 0x03, è la dimensione N delle righe;
  - il terzo, posto all'indirizzo 0x04, corrisponde alla soglia con cui si devono confrontare tutti gli elementi della matrice.

  In origine, all'interno di questo stato, sono stati rilevati dei ritardi nell'emissione del nuovo dato; per ovviare a tale problema sono stati utilizzati alcuni parametri per rallentare il processo e consentirne il corretto andamento.
  Si passa allo stato S1 quando count_addr è uguale all'indirizzo 0x05 ovvero quando il dato in ingresso è il primo elemento della matrice.

```
109     when S0 =>
110         if (sign = '1') then
111             if (count_addr = "0000000000000010") then       --se l'indirizzo è 0x02 si sta prelevando il dato corrispondente al numero di colonne
112                 next_state <= S0;       --si impone che lo stato successivo siauguale a quello corrente
113                 col <= i_data;          --il segnale col è posto uguale al dato letto
114                 o_en <= '1';
115                 sx_col <= i_data;       --si inizializza il segnale sx_col ponendolo uguale al dato in ingresso
116                 dx_col <= "00000000";   --si inizializza il segnale dx-col ponendolo uguale a 0
117             elsif (count_addr = "0000000000000011") then    --se l'indirizzo è 0x03 si sta prelevando il dato corrispondente al numero di righe
118                     next_state <= S0;
119                     row <= i_data;          --il segnale row è posto uguale al dato letto
120                     high_row <= i_data;     --si inizializza il segnale high_row ponendolo uguale al dato in ingresso
121                     low_row <= "00000000";  --si inizializza il segnale low_row ponendolo uguale a 0
122             elsif (count_addr = "0000000000000100") then    --se l'indirizzo è 0x04 si sta prelevando il dato corrispondente al valore di soglia
123                     next_state <= S0;
124                     threshold <= i_data;    --il segnale threshold è posto uguale al dato letto
125                     r <= '1';
126             end if;
127         sign <= '0';
128         count_addr <= count_addr +inc;  --si incrementa il contatore dell'indirizzo di un'unità
```

```
129                 else
130                     sign <= '1';
131                     if (count_addr = "0000000000000101") then   --se l'indirizzo è 0x05 si sta prelevando il primo elemento della matrice
132                         if( r = '1') then    --nelle letture iniziali si è individuato un ritardo; questo controllo serve per impedire
133                             next_state <= S0;           --che si passi allo stato successivo come primo valore quello di soglia
134                             sign <= '0';
135                             r <= '0';
136                         else
137                             if(col = "00000000" or row = "00000000") then  --caso limite matrice nulla, si settano i 4 valori dei lati a 0x00
138                                 dx_col <= "00000000";
139                                 sx_col <= "00000000";
140                                 high_row <= "00000000";
141                                 low_row <= "00000000";
142                                 d <= '1';           --d ed r sono parametri usati nello stato S2, si settano i valori che garantiscano il
143                                 r <= '0';           --corretto funzionamento dello stato prossimo
144                                 next_state <= S2;   -- si passa allo stato S2 in quanto non vi è alcun dato in memoria da leggere
145                             else
146                                 next_state <= S1;   --si passa allo stato successivo in quanto si è concluso il primo step del processo
147                             end if;
148                         end if;
149                     end if;
```

- Lo stato S1 contiene l'algoritmo atto a leggere gli elementi della matrice, a valutarne il valore rispetto alla soglia e a cambiare, eventualmente i 4 segnali corrispondenti ai lati estremi della figura cercata. I contatori partono da 0 pertanto si giunge all'ultima colonna quando count_col = col - 1, ovvero al valore della dimensione M-1, e all'ultima riga quando count_row = row - 1, cioè pari alla dimensione N-1. Una volta letto l'ultimo elemento dell'ultima riga e dell'ultima colonna si effettua, tra la riga 179 e la riga 184, un controllo atto a porre i valori si sx_col e low_row pari a zero nel caso in cui non vi sia alcun elemento con valore pari o superiore alla soglia.

```
154     when S1 =>
155         if (count_col < col) then                   --se il dato in ingresso ha un valore pari o superiore a quello della soglia
156             if (i_data >= threshold) then           --si provede a ricalcolare i 4 valori necessari per calcolare i lati dell'immagine
157                 if(count_row <= high_row ) then     --se il contatore delle righe ha valore inferiore a quello di high_row allora
158                     high_row <= count_row;          --si sostituisce quest'ultimo con il valore di count_row
159                 end if;
160                 if(count_row > low_row) then        --se il contatore delle righe ha valore uguale o superiore a quello di low_row allora
161                     low_row <= count_row;           --si sostituisce quest'ultimo con il valore di count_row
162                 end if;
163                 if (count_col < sx_col) then        --se il contatore delle colonne ha valore inferiore a quello di sx_col allora
164                     sx_col <= count_col;            --si sostituisce quest'ultimo con il valore di count_col
165                 end if;
166                 if ( count_col >= dx_col) then      --se il contatore delle colonne ha valore uguale o superiore a quello di dx_col allora
167                     dx_col<= count_col;             --si sostituisce quest'ultimo con il valore di count_col
168                 end if;
169             end if;
170         end if;
171         o_en <= '1';
172         if (count_col = (col-inc)) then             --se il contatore delle colonne ha valore pari alla dimensione M della matrice
173             count_col <= "00000000";                --si deve azzerare tale contatore ed incrementare quello delle righe
174             if(count_row < (row- inc)) then         --fintantochè il contatore delle righe è inferiore alla dimensione N della matrice
175                 count_row <= count_row + inc;       --si incrementa di uno
176                 count_addr <= count_addr + i;       --si incrementa di uno count_addr
177                 next_state <= S1;                   --lo stato prossimo corrisponde allo stato corrente
178             else
179                 if (sx_col > dx_col) then           --se il contatore delle righe è pari alla dimensione N della matrice si è giunti
180                 sx_col <= dx_col;                --all'ultimo elemento della matrice
181             end if;
182             if (high_row > low_row) then
183                 high_row <= low_row;
184             end if;
185             count_col <= col;
186             next_state <= S2;                   --lo stato prossimo corrisponde a S2 in quanto è terminata la lettura della matrice
187             count_addr <= "0000000000000000";   --count_addr = 0x00 in quanto questo è il primo indirizzo in cui si scrive
188             end if;
189         elsif (count_col < (col-inc)) then      --se il contatore delle colonne ha valore inferiore alla dimensione M della matrice
190             count_addr <= count_addr + i;       --incremento sia il suo valore che quello dell'indirizzo di uno
191             count_col <= count_col + inc;
192             next_state <= S1;                   --lo stato prossimo corrisponde allo stato corrente
193         end if;
194         if (i_rst = '1') then                   --nel caso in cui ci sia uno stato di reset = 1 lo stato prossimo corrisponde a RST
195             next_state <= RST;
196         end if;
```

- Lo stato S2 è solo uno di passaggio nel quale viene calcolata l'area. È bene notare che nel calcolo dell'area debbano essere tenute in considerazione anche le righe e le colonne perimetrali. Per tale ragione durante il calcolo di base ed altezza risulta necessario incrementare i due risultati di uno ed escludere dall'if a riga 202 il caso in cui col= 0x01 o row = 0x01 (in caso contrario la matrice N=1 ed M=1 risulterebbe sempre nulla a prescindere).

Alla riga 213 si pone l'uscita o_we alta in quanto, nello stato successivo, si compie l'azione di scrittura in memoria che richiede, per la specifica, che o_we sia pari a 1 .

```
197     when S2 =>
198         if( d ='1') then                        --parametro introdotto per il caso della matrice nulla, se d = 1
199             count_addr <= "0000000000000000";   --si pone count_addr e di conseguenza o_addr = 0x00, primo indirizzo di scrittura
200             d <= '0';                           --tale ciclo deve essere percorso solo una volta
201         end if;
202         if(sx_col = "00000000" and dx_col = "00000000" and low_row = "00000000" and high_row = "00000000" and (col>"00000001" or col="00000000" and (row>"00000001" or row="00000000")) then
203             lenght <= dx_col - sx_col;          --se non vi sono elementi con valore pari o superiore alla soglia base ed altezza sono nulli
204             high <= low_row - high_row;
205             area <= "0000000000000000";
206         else
207             lenght <= dx_col - sx_col + inc;    --in caso contrario si trovano due valori diversi da zero
208             high <=  low_row - high_row + inc;
209             area <= high * lenght;                  --si calcola l'area moltiplicando base per altezza
210         end if;
211
212         s<= '1';
213         o_we <= '1';                            --si alza il segnale o_we in quanto l'operazione successiva prevede la scrittura in memoria
214         next_state <= S3;                       --lo stato prossimo corrisponde a S3
215         if (i_rst = '1') then
216             next_state <= RST;                  --nel caso in cui ci sia uno stato di reset = 1 lo stato prossimo corrisponde a RST
217         end if;
```

- Quest'ultimo stato si occupa del salvataggio dell'area in memoria; dati i ritardi che si generano sia in lettura che in scrittura si è deciso di allungare i tempi tornando dopo ciascuna operazione allo stato precedente.
  Il processo si conclude solo dopo che il segnale o_done è rimasto alto per un intero ciclo di clock.
  A questo punto il componente torna nello stato di RST e si mette in attesa di ricevere un nuovo segnale di start.

```
218             when S3 =>
219                 if( s ='1') then
220                     if (count_addr =  "0000000000000000" and r = '0') then      --scrivo gli 8bit meno significativi all'indirizzo 0x00
221                         o_data <= area(7 downto 0);         --il dato in uscita corrisponde agli 8bit meno significativi
222                         next_state <= S2;                   --lo stato prossimo corrisponde a S2
223                         s <= '0';
224                         r <= '1';
225                     elsif (count_addr =  "0000000000000000" and r = '1') then
226                         count_addr <= count_addr + i;       --incremento l'indirizzo di uno
227                         next_state <= S2;                   --lo stato prossimo corrisponde a S2
228                         s <= '0';
229                         k <= '1';
230                     elsif (count_addr = "0000000000000001" and k = '1') then    --scrivo gli 8bit più significativi all'indirizzo 0x01
231                         o_data <= area(15 downto 8);        --il dato in uscita corrisponde agli 8bit più significativi
232                         next_state <= S2;                   --lo stato prossimo corrisponde a S2
233                         s <= '0';
234                         k <= '0';
235                     elsif (count_addr = "0000000000000001" and k = '0') then
236                         c <= '1';
237                         s <= '0';
238                         o_en <= '0';    --non dovendo più compiere alcuna azione con la memoria risulta più neessario tenere alto o_en
239                         o_we <= '0';
240                         o_done <= '1';  --avendo completato tutte le operazioni richieste si può alzare o_done
241                     end if;
242                 end if;
243                 if ( c = '1') then
244                     o_done <= '0';      --passato un ciclo di clock si può porre o_done = 0
245                     count_addr <= count_addr +i;                --incremento di uno l'indirizzo
246                     next_state <= RST;                          --si torna allo stato iniziale
247                 end if;
248                 if (i_rst = '1') then
249                     next_state <= RST;  --nel caso in cui ci sia uno stato di reset = 1 lo stato prossimo corrisponde a RST
250                 end if;
251         end case;
252       end if;
253     end process;
254 end FSM;
```

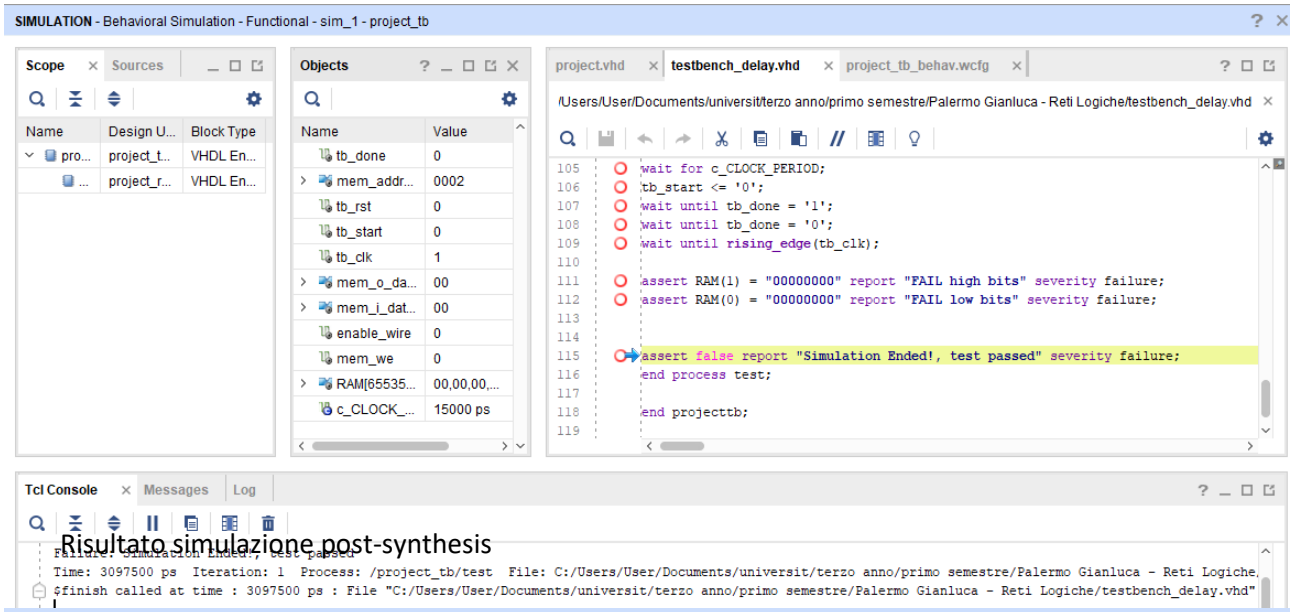
Testing

TestBench

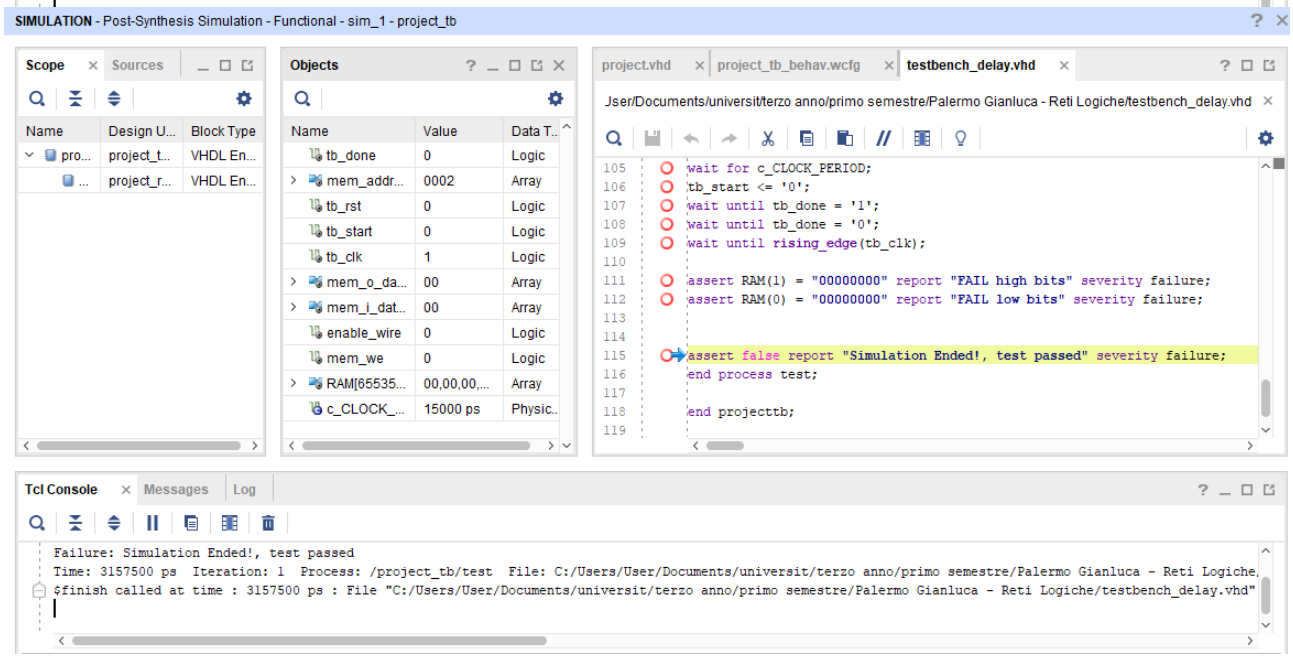
Il componente risulta idoneo alla specifica in quanto supera con successo i 4 testbench propositi sia in simulazione behavioral che in quella post-sintesi. Di seguito sono riportati gli screenshot a dimostrazione di quanto affermato.

- TestBench_delay

Risultato simulazione Behavioral

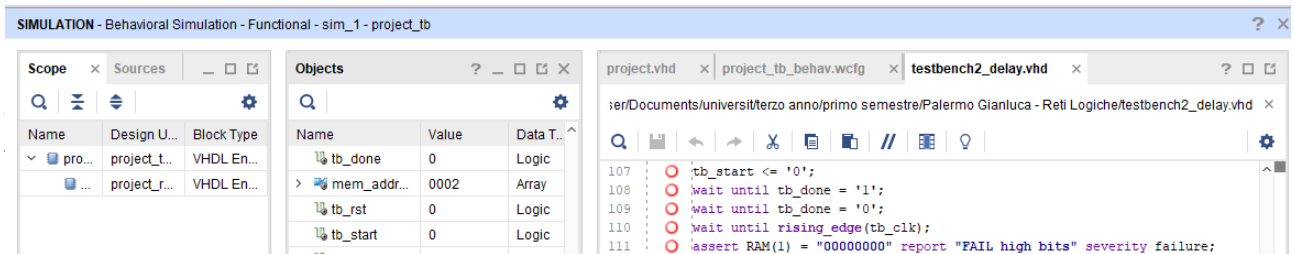


Risultato simulazione post-synthesis

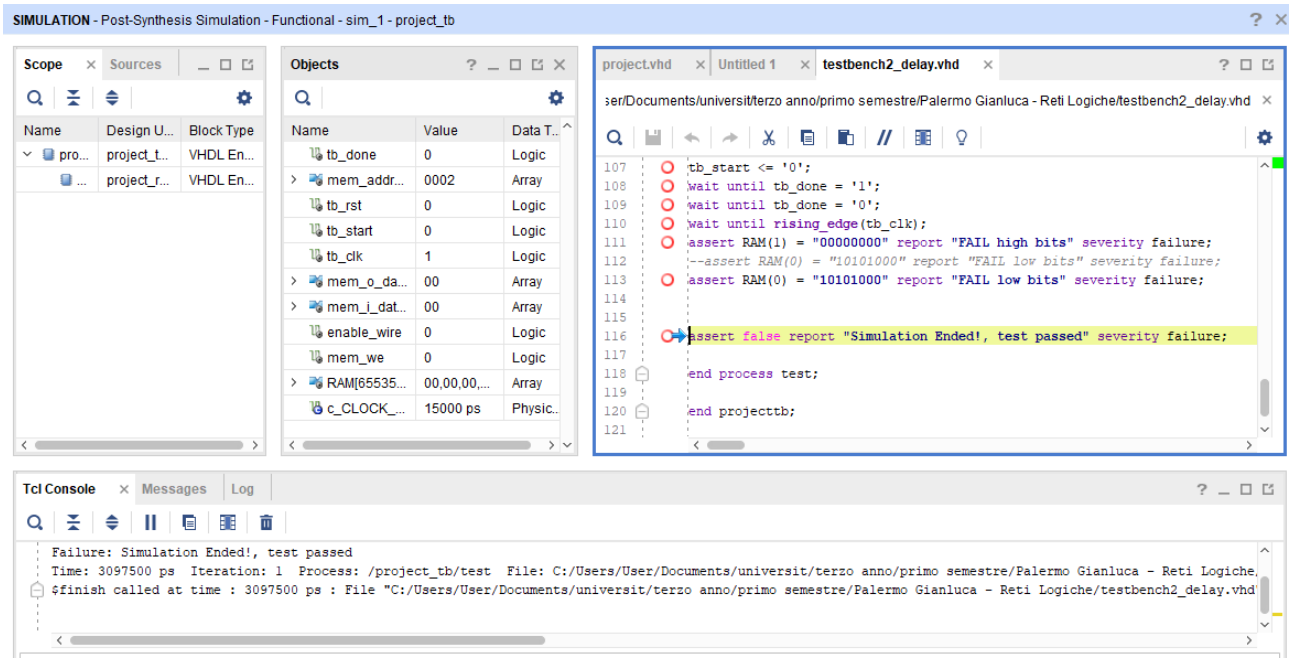


- TestBench2_delay

Risultato simulazione Behavioral

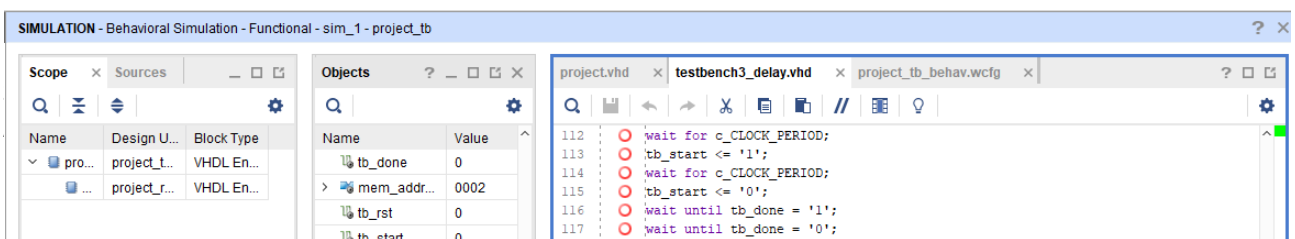


Risultato simulazione post-synthesis

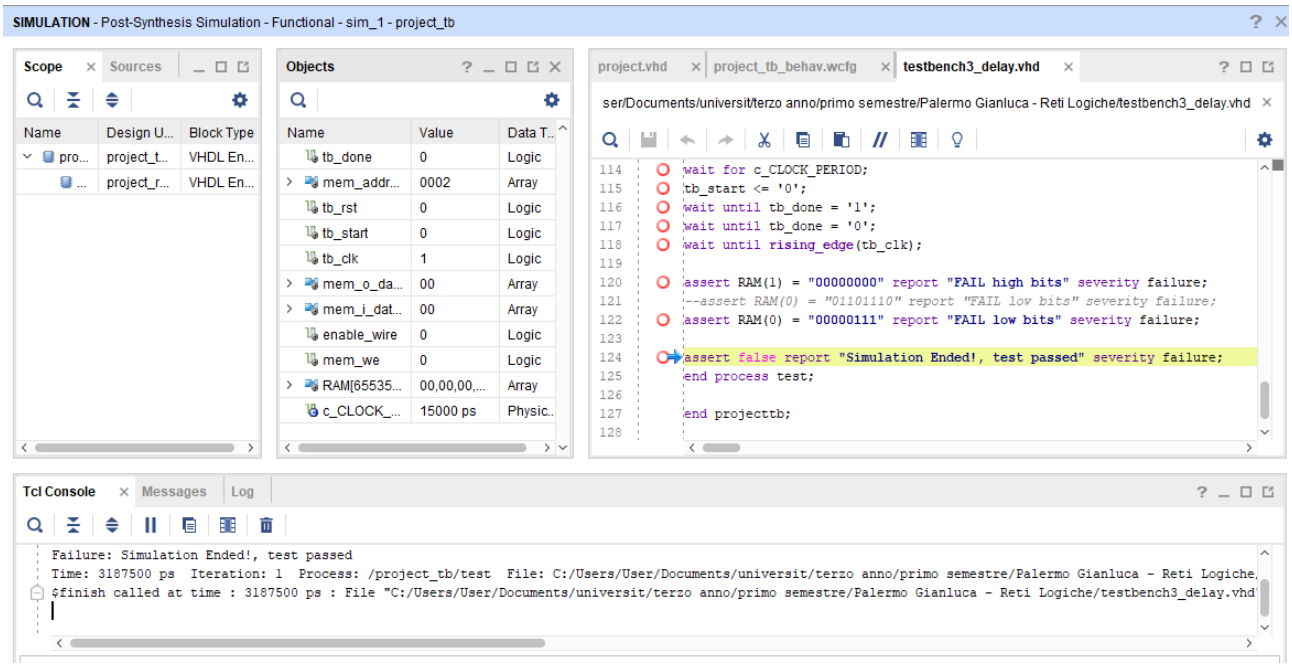


- TestBench3_delay

Risultato simulazione Behavioral

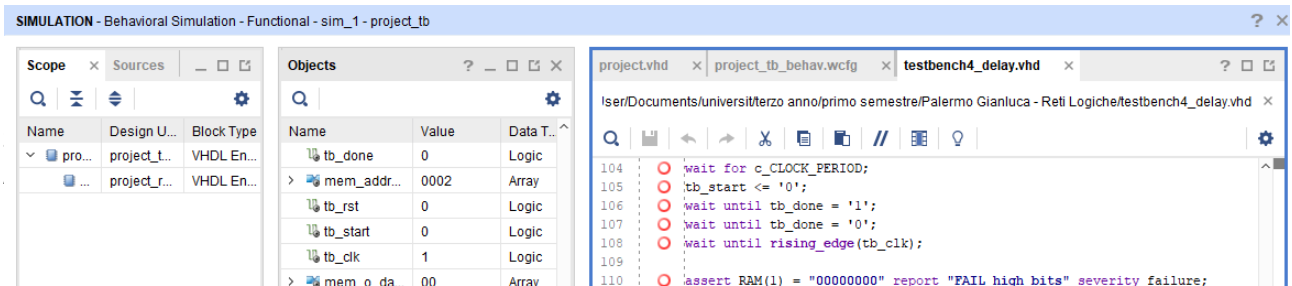


Risultato simulazione post-synthesis

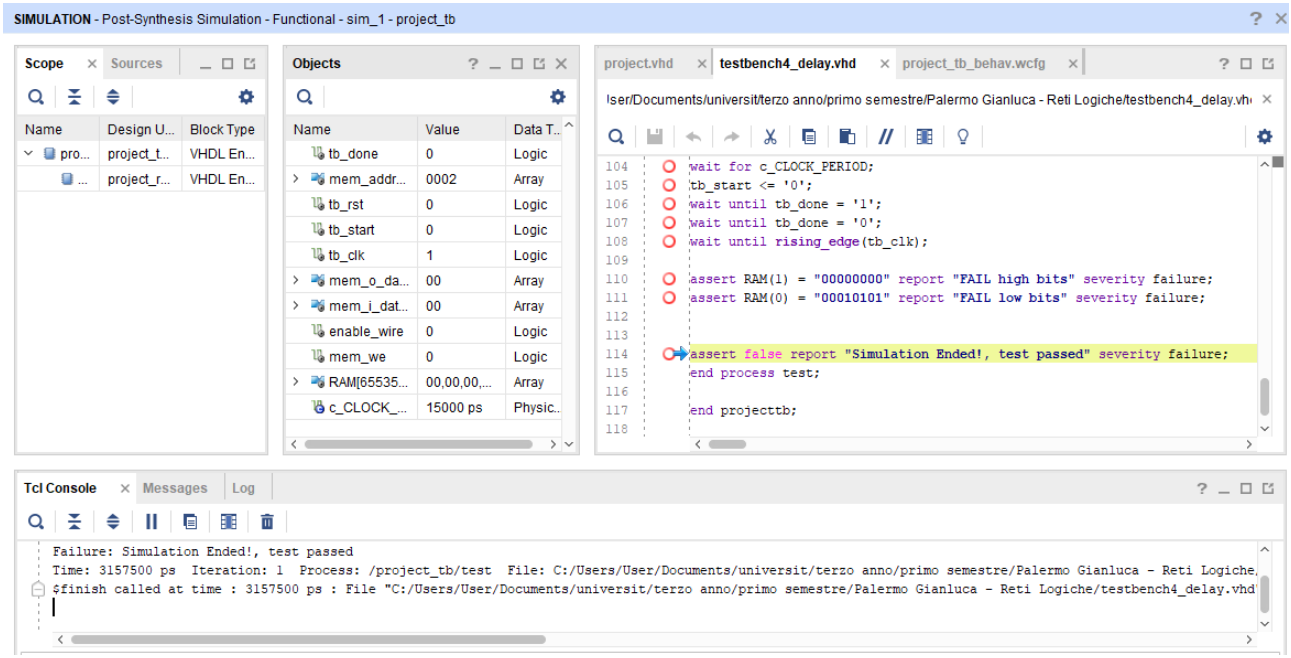


- TestBench4_delay

Risultato simulazione Behavioral



Risultato simulazione post-synthesis



Ulteriori Test

Una volta osservato il corretto funzionamento del componente nei casi proposti si è provveduto ad aumentare la copertura dei test e quindi ad appurare il suo corretto funzionamento anche in presenza di eventi limite.

Studentessa: Alessandra Pasini
A.S.: 2017/2018

Si è ritenuto opportuno testare, con successo, i seguenti casi:

- Matrice vuota con valore N=0 e M=0; con N=0 e M!=0 e con N!=0 e M=0.
- Matrice dimensione massima, ovvero con N= 255 righe ed M= 255 colonne.
- Matrice di dimensione generica con valori superiori alla soglia su una sola riga.
- Matrice di dimensione generica con valori superiori alla soglia solo su una colonna.
- Immagine con area=1, ovvero un unico valore uguale o superiore alla soglia all'interno della matrice
- Immagine formata da due soli valori.
- immagine con forma triangolare, anche in questo caso viene correttamente individuata l'area massima rettangolare o quadrata che racchiude la figura.
- Inserimento di un reset durante l'esecuzione: quando, durante la computazione, si riscontra un segnale di reset alto, indipendentemente dallo stato in cui il componente si trova, si ha un'interruzione immediata del processo e ritorno allo stato inziale. Il processo inizia nuovamente solo nel momento in cui si riceve un segnale di start = 1.
- Avvio consecutivo di due esecuzioni sia con l'utilizzo di reset e start che con il solo segnale di start.