Progetto di Reti Logiche

A.A. 2023-2024

Francesco Foresti 10677417

1. Introduzione

Obbiettivo del progetto

L' obbiettivo del progetto è quello di realizzare un componente hardware descritto in VHDL in grado di leggere un elenco di valori interi intervallati da zeri in una memoria e complementarli con dei valori indicanti la credibilità degli stessi oppure nel caso di valori pari a zero riportare l'ultimo valore, valido letto in precedenza, e riportarlo con una credibilità ridotta.

Specifica nel dettaglio

Il procedimento ha inizio in seguito ad un segnale i_start, a quel punto il componente riceve un segnale i_k indicante il numero di valori da controllare, un segnale i_add indicante l'indirizzo di memoria da cui i valori, che sono scritti uno di seguito all'altro, vanno letti.

Le operazioni in memoria vengono eseguite tramite i segnali o_mem_en e o_mem_we, che abilitano rispettivamente le operazioni di lettura e scrittura, i_mem_data contente il dato letto, o_mem_data contenente il dato da scrivere e o_mem_addr indicante l'indirizzo di memoria con cui si vuole interagire.

Il componte continua l'elaborazione fino a quando non ha affidato un valore di credibilità a tutti gli i_k valori, comunicandolo esternamente con un segnale o_done oppure fino a quando non riceve un segnale asincrono di restart i_rst.

La modalità con cui viene assegnato il valore di credibilità C è la seguente: se il vettore letto da i_mem_data(che è un vettore di 8 bit interpretato come un intero senza segno) è un numero qualsiasi diverso da zero allora $C = 31$ (00011111) altrimenti il valore letto viene sostituito con l'ultimo valore accettato dal componente(che in caso di una prima iterazione è zero) seguito da 31 meno il numero di volte di fila per cui l'ultimo valore è stato già utilizzato invece del valore effettivamente letto.

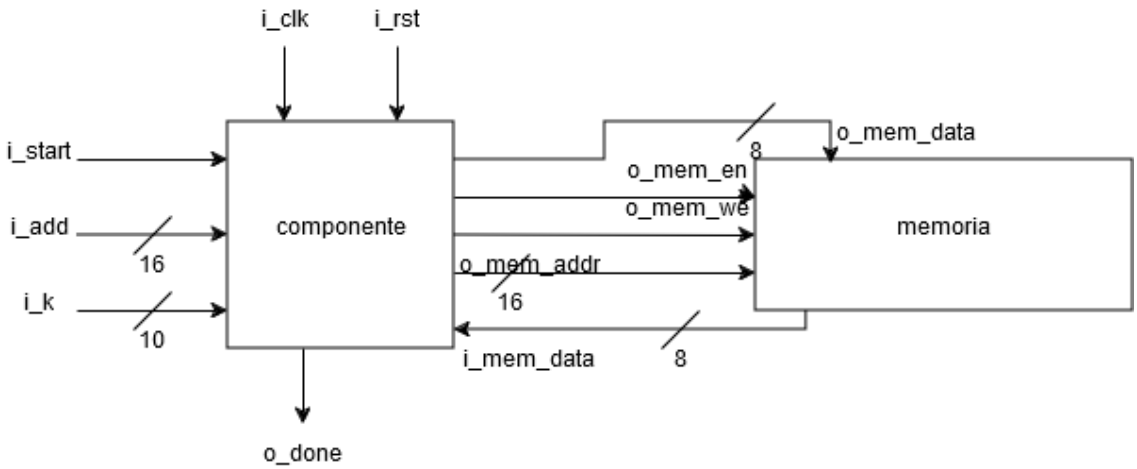
es:

12 0 15 0 0 0 12 0 0 0 0 0:

12 31 15 31 15 30 12 31 12 30 12 29

In memoria i numeri ed i valori di credibilità sono salvati come vettori ad 8 bit, l'indirizzo

i_add come un vettore a 16 bit e i_k come un vettore a 10 bit.




```
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;
```

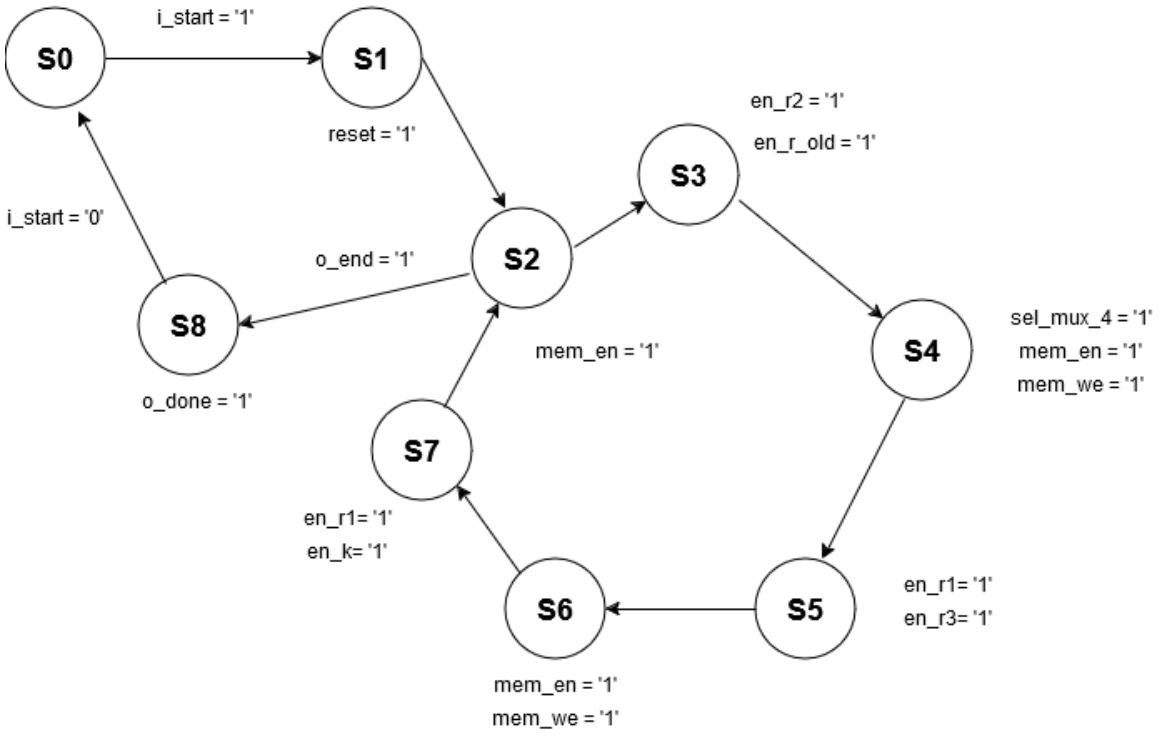
2. Architettura

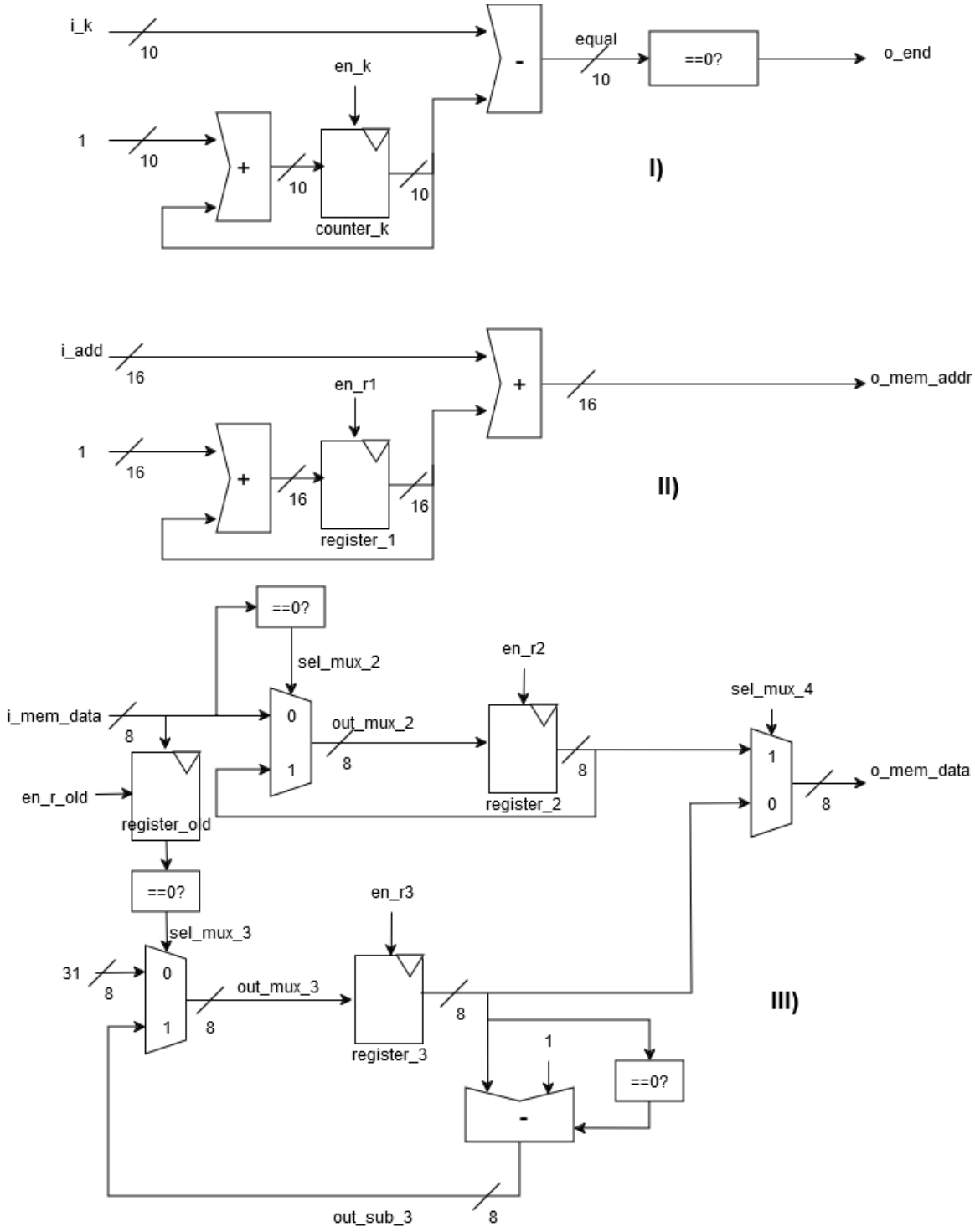
Struttura Generale

Il componente è stato realizzato con due diverse entità comunicanti fra di loro ognuana con la propria architettura:

Un **DATAPATH** contente i collegamenti per i vettori di ingresso ed uscita, registri in cui salvare i valori e tutto ciò che riguarda la manipolazione degli stessi(sommatori, comparatori…).

Una **MACCHINA A STATI FINITI** responsabile invece della logica e del controllo del datapath tramite l'ausilio di segnali.





Descrizione del datapath

Il datapath può essere diviso in tre parti, ognuna con uno specifico scopo(partendo dall'alto):

I) In questa parte viene letto il vettore **i_k**(10 bit) che siccome rimane in ingresso per tutta la durata del processo(i_start alto) non viene salvato.

Si utilizza un registro, **counter_k**(attivato da **en_k**), per salvare il contenuto di un contatore inizializzato a zero che viene incrementeto ad ogni ciclo completo della FSM, l'uscita dello stesso è sottratta ad **i_k** ed il risultato viene costantemente comparato a zero. Nel caso l'uguaglianza sia verificata viene mandato un segnale **o_end** alla fsm che a sua volta emetterà un segnale **o_done** alto.

Per descrivere contatore e registro sono stati utilizzati dei processi sequenziali, mentre per la sottrazione e successiva comparazione a zero sono stati utilizzati componenti combinatori.

II) In questa parte viene letto il segnale **i_add**(16 bit) e vengono svolte le operazioni per decidere il segnale **o_mem_addr** necessario per le successive operazioni di lettura e scrittura.

Anche in questo caso si sfrutta la costante presenza di **i_add** in ingresso ed il fatto che gli indirizzi sono uno di seguito all'altro.

Viene inizializzato **register_1** a zero volto a salvare il contenuto di un altro contatore(tramite il segnale **en_r1**), il cui valore è costantemente sommato ad **i_add** e portato all'uscita **o_mem_addr**.

Per descivere contatore e registro viene usato un processo sequenziale mentre la somma è descritta da un componente combinatorio.

III) In questa ultima parte vengono svolte due operazioni:

1) dato l'ingresso in memoria si sceglie se riportarlo in uscita oppure se utilizzare il valore salvato in precedenza.

2) viene assegnato un valore di credibilità al valore letto.

L'ingresso **i_mem_data** viene immediatamente salvato in **register_old**(attivato **en_r_old**), quindi l'esito di una sua comparazione a zero viene usata come selettore del **multiplexer_2** la cui uscita è salvata nel **register_2**.

Una comparazione a zero del contenuto di **register_old** è utilizzata come

selettore per il **multiplexer_3** la cui uscita è salvata in **register_3** (attivato da **en_r3**)il cui contenuto può essere decrementato e salvato in **out_sub_3** il secondo ingresso di mux_3.

Infine l'uscita **o_mem_data** viene controllata dal **multiplexer_4** che alterna fra il contenuto di **register_2** e **register_3**.

Per descrivere i registri ed il sottrattore vengono utilizati processi sequenziali, mentre i comparatori a zero, i selettori ed i multiplexer sono destritti come componenti combinatori.

Descrizione macchina a stati finiti

Al datapath è affiancata una macchina a stati finiti volta a controllare i segnali interni del modulo:

S0 è lo stato di partenza di idle in cui la macchina resta in attesa di un segnale i_start o a cui viene riportata in seguito ad un segnale i_rst.

S1 la macchina produce un segnale interno di reset (diverso da i_rst) che riporta al valore iniziale i vari registri. É stato introdotto per gestire sia un riavvio della macchina dovuto ad un restart sia una seconda richiesta di computazione.

S2 attiva la memoria in lettura, se la MSF ha ricevuto dal datapath un segnale interno o_end imposta come stato successivo S8 altrimenti prosegue su S3

S3 vengono salvati i contenuti in ingresso ai registri r2 ed r_old

S4 viene attivata la memoria in scrittura e viene selezionato l'ingresso 1 del mux_4 destinato all'uscita o_mem_data: viene riscritto il valore appena letto oppure quello precedente nel caso quest'ultimo sia zero

S5 vengono salvati i contenuti in ingresso ai registri r1 ed r3

S6 viene attivata la memoria in scrittura: viene scritto il valore di credibilità 31 nel caso di un nuovo numero oppure l'utimo c decrementato di 1(se diverso da zero)

S7 vengono salvati i contenuti in ingresso ai registri r1 ed rk, in particolare

con r_k contente il numero di iterazioni del ciclo da S2 a S7 svolte finora

S8 viene mandato il segnale o_done indicante la terminazione della computazione. Per ritornare a S0 è necessario che il segnale i_start, che da specifica rimane alto lungo tutta la durata della computazione, si abbassi

3. Risultati Sperimentali

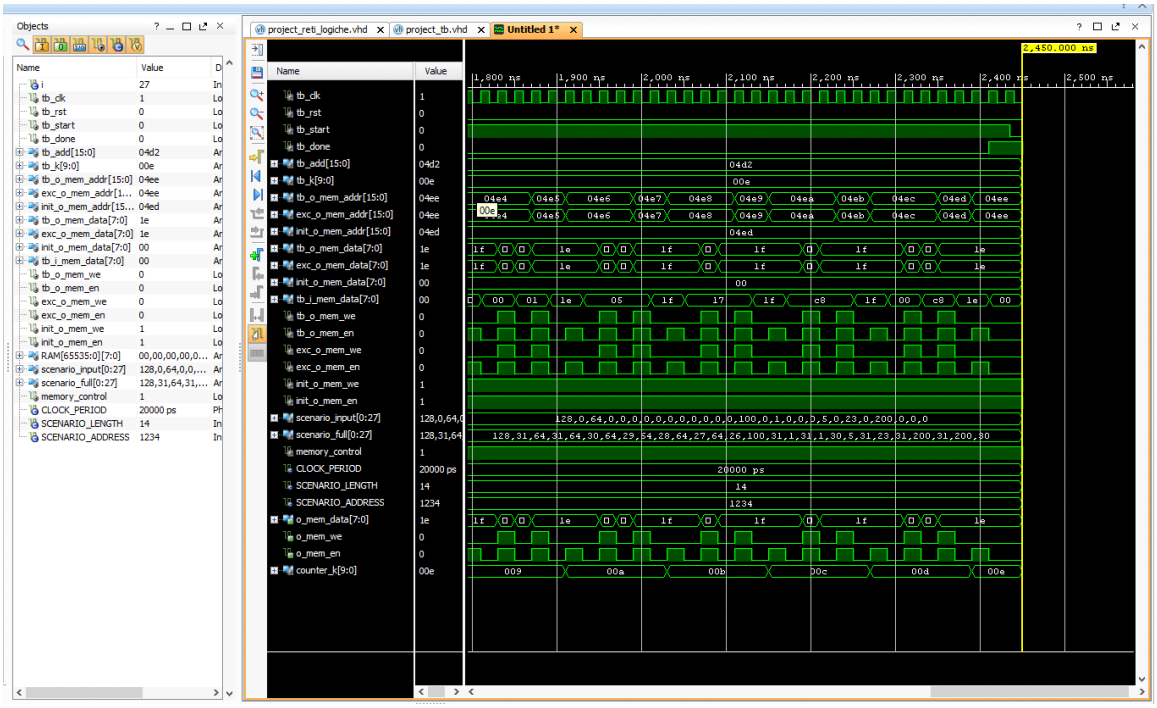
Conclusa la descrizione del componente si è proceduto con l'eseguire una simulazione in presintesi(behavioral) con i test banch forniti.

Ovviamente il progetto iniziale era piuttosto diverso da quello finale, molti dei cambiamenti sono stati svolti in seguito a fallimenti nei test.

In particolare l'introduzione del segnale interno di o_end che permettesse al datapath di comunicare la terminazione alla MSF e del segnale interno di reset comodo per gestire richieste di computazione seriali.

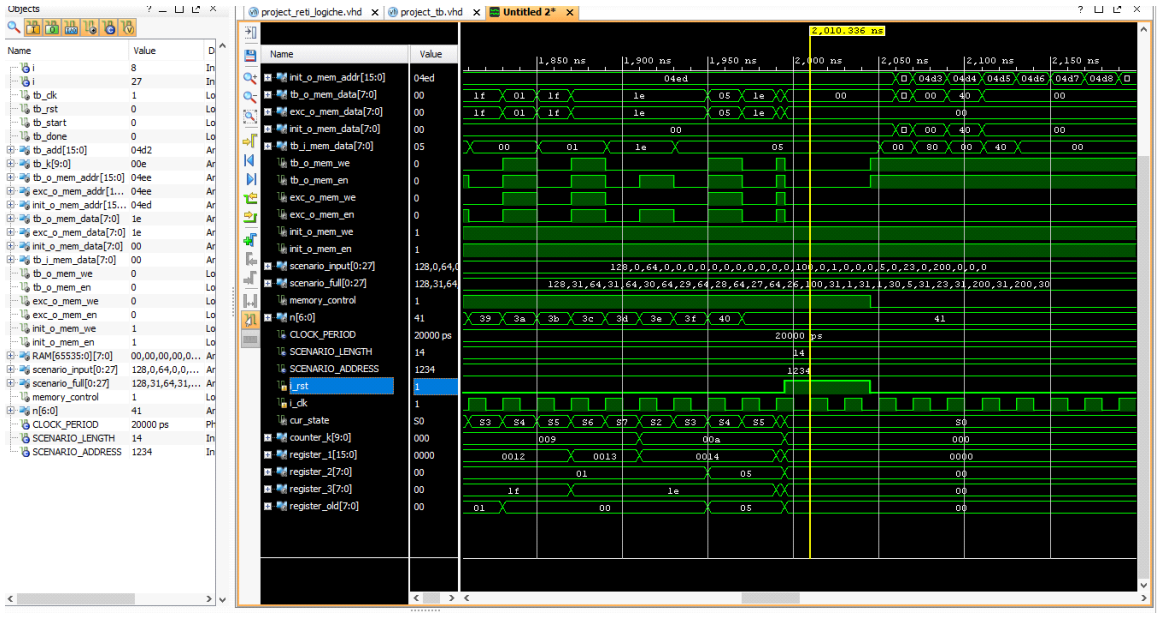
Ecco di seguito alcuni dei test più significativi svolti:

project_tb(il primo fornito):
contiene un'unica richiesta con dei valori standard e non molto numerosi.



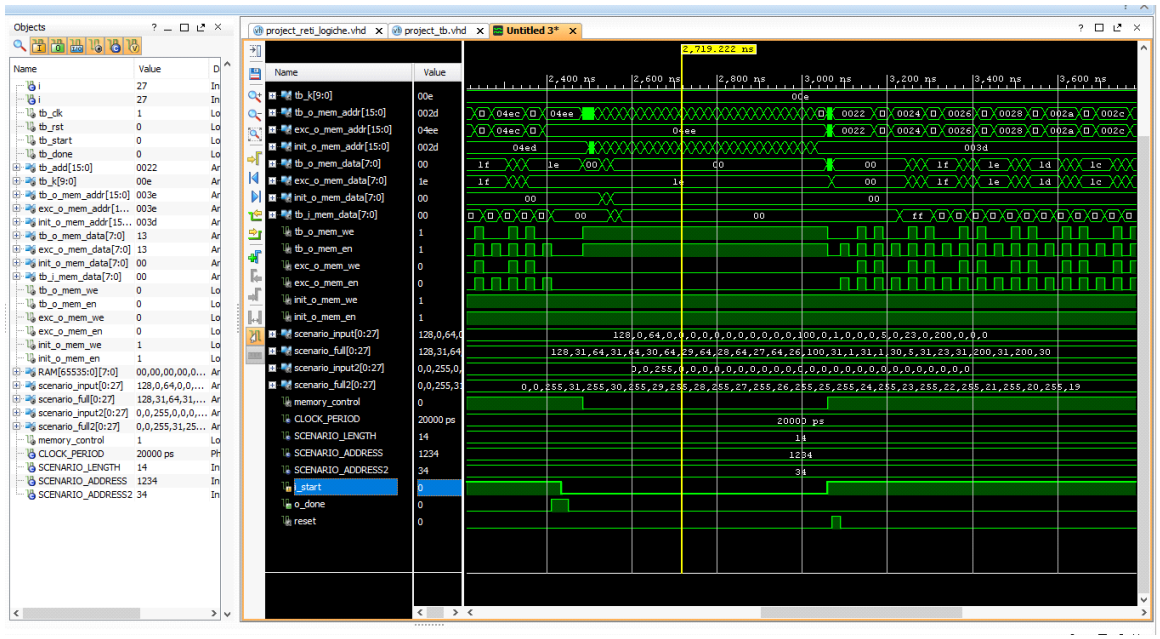
i_rst_tb

Testbench per verificare la capacità di riprendere una nuova esecuzione in seguito al riconoscimento di un segnale i_rst asincrono.



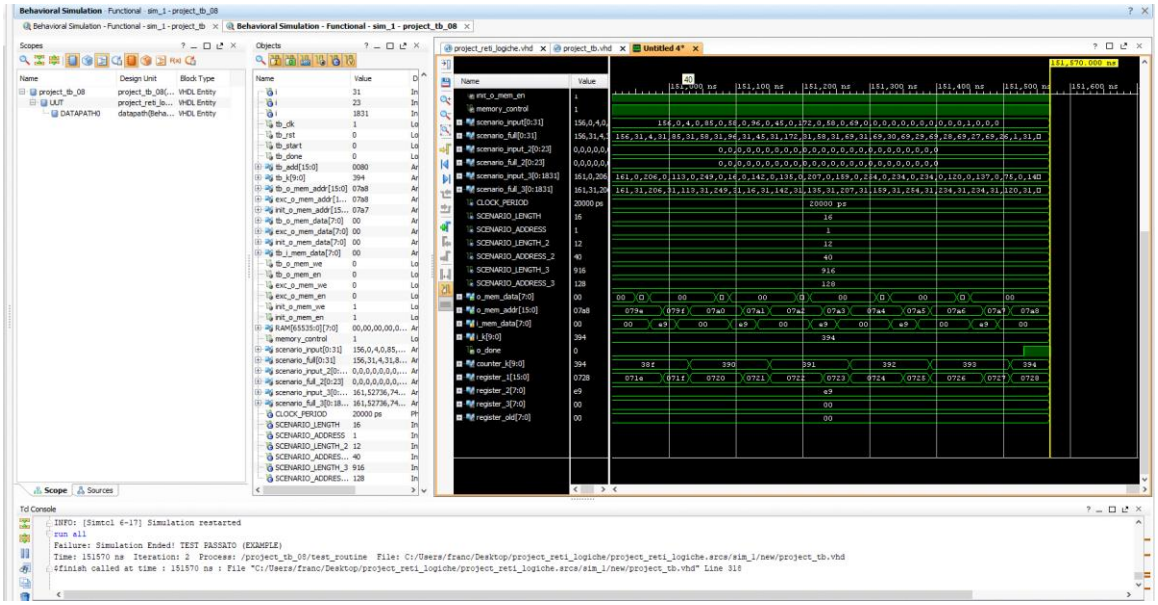
i_start_tb

Testbench per verificare la capacità di elaborare più scenari in ingresso in sequenza.

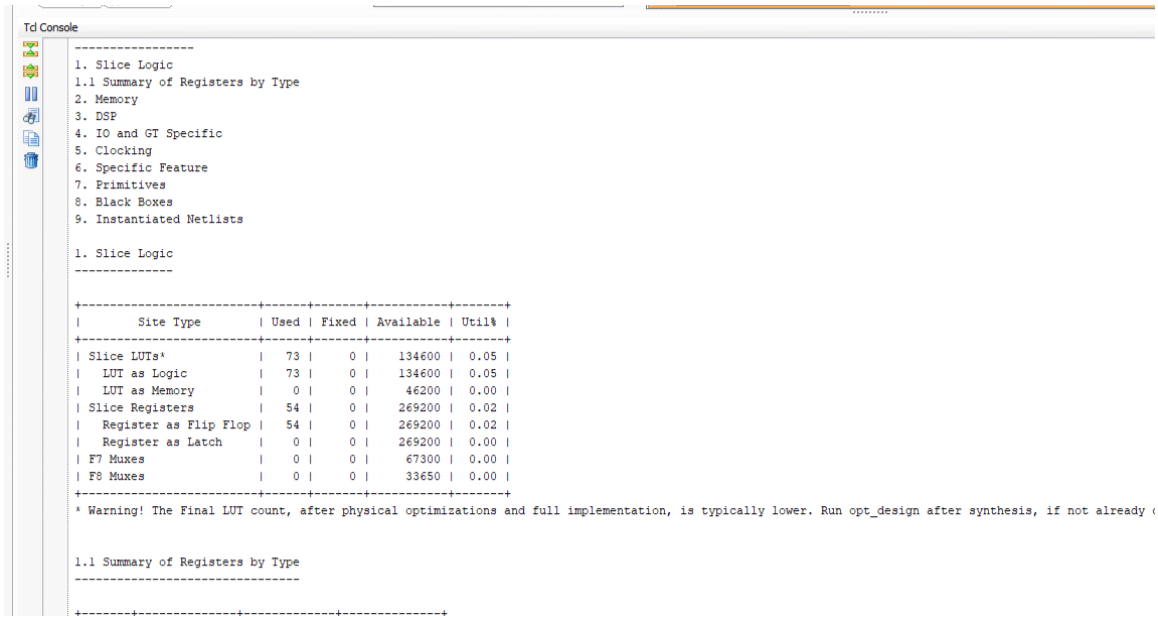


tb_8

Un test banch contente tre serie piuttosto lunghe di valori



In seguito al superamento dei test bench si è provveduto alla sintesi del componente e ai relativi test. La sintesi è avvenuta con successo e non vengono generati latch durante la stessa, il componente è stato testato anche in post sintesi(con I medesimi test bench) senza riscontrare problemi.



Infine analizzando il tempo di utilizzo si è riscontrato uno slack time di 16.233 ns.

```
Timing Report

Slack (MET) :             16.233ns  (required time - arrival time)
  Source:                 DATAPATH0/register_3_reg[4]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=20.000ns})
  Destination:            DATAPATH0/register_3_reg[0]/CE
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=20.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns - clock rise@0.000ns)
  Data Path Delay:        3.385ns  (logic 0.999ns (29.513%)  route 2.386ns (70.487%))
  Logic Levels:           3  (LUT2=1 LUT6=2)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 22.100 - 20.000 )
    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
```

4. Conclusioni

Il componente è stato realizzato cercando di mantenere il più possibile una caratteristica di atomicità nelle parti che lo compongo questo si è rivelato estremamente utile in fase di testing per trovare casi limite o “pezzi” mal progettati.

Il lavoro si è svolto in maniera iterativa partendo dal circuito su carta per procedere alla descrizione in vhdl, ed in seguito alla sintesi.