Progetto di Reti Logiche

Pasquale Scarmozzino 10582003

31 marzo 2020

Indice

1 Introduzione

Il componente progettato realizza la codifica di un indirizzo immesso su un bus indirizzi secondo il metodo "Working Zone". Nello spazio degli indirizzi si individuano *Nwz*, in questo caso 8, indirizzi base per un intervallo di dimensione fissa *Dwz*, in questo caso 4. Il primo bit da sinistra, detto WZ_BIT, specifica se l'indirizzo è stato ricodificato o meno. L'indirizzo immesso sul bus viene trasformato nel seguente modo prima di essere ritrasmesso:

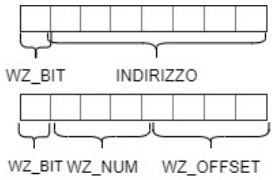


Figura 1: Suddivisione bit indirizzo, sopra non codificato, sotto ricodificato

1. se l'indirizzo non appartiene a una working zone, viene ritrasmesso così com'è, con WZ_BIT = 0;
2. se l'indirizzo appartiene a una working zone viene ricodificato:
   - si asserisce WZ_BIT = 1;
   - i successivi $\lceil \log_2 (\text{Nwz}) \rceil$ bit, in questo caso dal bit 6 al bit 4, contengono la codifica in binario naturale del numero di working zone WZ_NUM;
   - i rimanenti bit sono utilizzati per la codifica *one hot* della posizione dell'indirizzo nella working zone WZ_OFFSET, con la posizione 0 corrispondente a 0001, 1 a 0010, 2 a 0100, 3 a 1000.

Il componente inizia la codifica quando un segnale START (`i_start`) viene asserito. Una volta conclusa la fase di elaborazione porta alto un segnale DONE (`o_done`) precedentemente basso finché il segnale START non viene abbassato, infine il segnale DONE viene ribassato come nello schema della figura 2 e si aspetta un nuovo segnale di START.

Gli indirizzi di base delle working zone sono contenuti in una memoria nelle posizioni da 0 a 7, l'indirizzo da codificare è in posizione 8 e la codifica deve essere scritta nella posizione 9. Per leggere un valore da memoria occorre indicare l'indirizzo che lo contiene `o_address` di 16 bit e porre `o_en` = '1'. Per scrivere un indirizzo sulla memoria occorre porre `o_en` = '1', `o_we` = '1' e indicare con `o_address` l'indirizzo di memoria del dato desiderato. Il dato arriva all'entrata `i_data` del componente, mentre l'indirizzo codificato esce su `o_data`. Altri segnali sono `i_rst` per resettare il componente e `i_clk` che è il segnale di clock, con un periodo maggiore o uguale a 100 ns.

Il protocollo della memoria utilizzato è quello di una RAM single-port e write-first. I valori degli indirizzi base delle working zone contenuti in essa non cambiano se non in presenza di un segnale di reset, mentre l'indirizzo di codificare può

cambiare ogni volta che START viene abbassato. Ogni volta che si attivano i segnali di enable adatti e si carica un indirizzo valido la RAM rende disponibile alle sue uscite il valore desiderato entro l'inizio del ciclo di clock successivo.

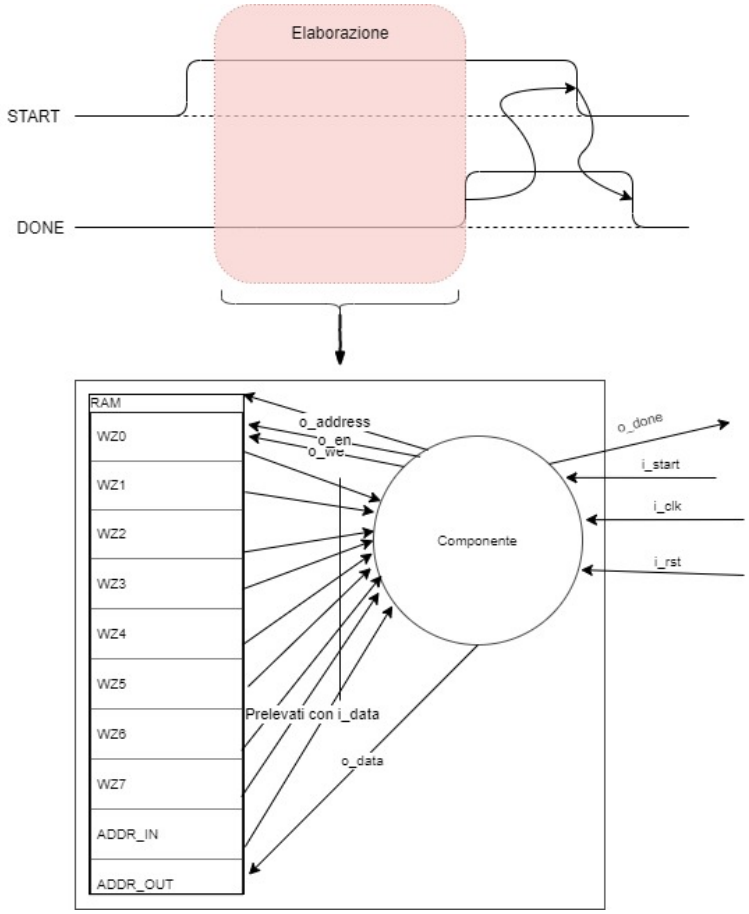


Figura 2: Andamento dei segnali e schema interazione tra RAM e componente

2 Architettura

Dal punto di vista architetturale il componente è strutturato come una macchina a stati finiti, i cui dettagli sono forniti in seguito, che preleva gli indirizzi delle working zone di base e li memorizza in 8 registri, successivamente preleva l'indirizzo da codificare e, una volta codificato, lo invia alla memoria e asserisce `o_done` tenendolo alto finché `i_start` non viene posto a 0. La logica che calcola la codifica è un modulo a parte (`output_logic1`) che seleziona l'uscita corretta tra quelle prodotte da 8 moduli `addrmap` contenuti in esso. La logica del modulo `output_logic1` e di ciò che contiene è interamente combinatoria per comprimere il tempo necessario a fornire l'indirizzo codificato e i moduli addrmap lavorano in parallelo. Questo comporta un costo in termini di area maggiore di una realizzazione sequenziale, ma non vi erano vincoli sulla area utilizzabile sul dispositivo. I registri servono per avere una memoria cache che riduca il numero di accessi in lettura alla RAM e, quindi, il tempo necessario a produrre un risultato. Le dimensioni di tutti i bus, registri, entrate e uscite sono state parametrizzate in modo da poter riutilizzare il codice anche per indirizzi di dimensione diversa con le dovute accortezze. Si deduce che il componente si struttura di una parte di memoria, ossia i registri, una parte di logica sequenziale, ossia la FSM e una parte di logica combinatoria di output che si integrano semplicemente e in modo chiaro, senza sovrapposizioni. La scelta di rendere la macchina a stati e la logica di output dei moduli autonomi è stata effettuata per poter agire sull'uno senza modificare l'altro per interventi che non modifichino in modo sostanziale la filosofia seguita. La figura a pag. 6 contiene la rappresentazione a piena pagina dei moduli principali.

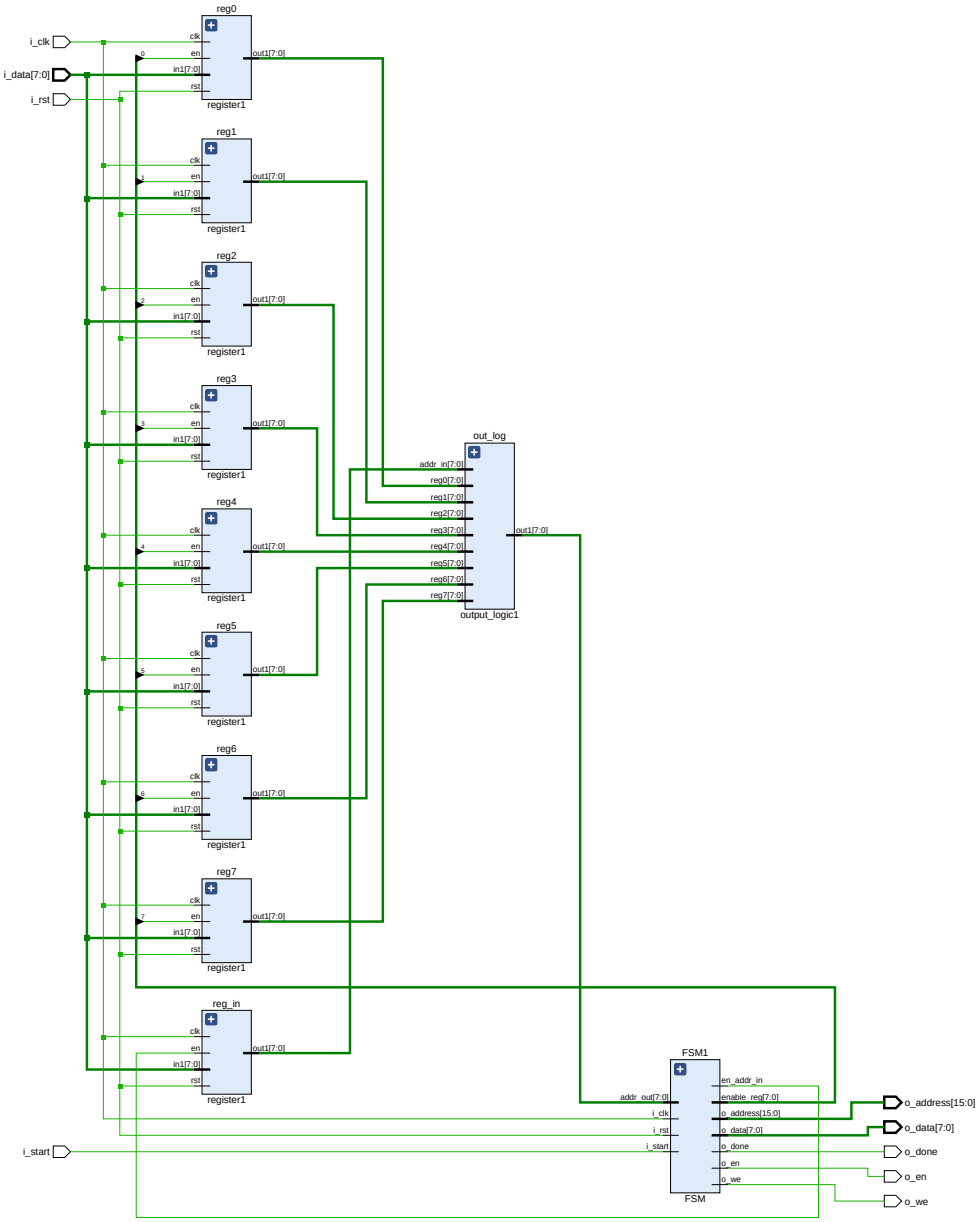


Figura 3: Architettura componente

2.1 project_reti_logiche

Questo modulo, centrale, descrive strutturalmente come si connettono gli altri moduli tra di loro e con l'interfaccia esterna. Strutturalmente vi sono 9 registri `reg0`, `reg1`, ..., `reg7` per gli indirizzi base delle working zone e `reg_in` per memorizzare l'indirizzo da codificare, una istanza di `output_logic1`, `out_log` e una istanza della FSM contenente la logica sequenziale. I registri, con enable come specificato sotto, sono collegati a `i_data` e opportunamente attivati per memorizzare i valori opportuni. Le uscite dei registri sono indicate con i `signal ad0`, ..., `ad7` e `addr_in`. Gli enable dei registri degli indirizzi base sono resi con il segnale `enable_reg` di 8 bit, uno per registro, mentre l'enable per il registro dell'indirizzo da codificare è `enable_addr_in`.

2.2 FSM1

Questo modulo concentra la logica sequenziale del componente in una macchina a stati finiti con 15 stati specificata in due `process`. Il processo `state_reg` attivato su `i_rst` e `i_clk` si occupa di selezionare lo stato di reset corretto e di aggiornare lo stato sul fronte ascendente del clock (il reset è invece asincrono). Il processo `FSM`, sensibile a `current_state` e `i_start` fonde le funzioni di stato prossimo e di uscita. I segnali di interfaccia `o_done`, `o_en` e `o_we` sono sempre definiti, `o_data` e `o_address` sono non specificati dove possibile per permettere ottimizzazioni nella sintesi. I segnali `enable_reg` e `enable_addr_in` corrispondono a quelli del modulo precedente. I suoi stati sono:

S0 stato di reset. Gli enable dei registri e `o_data` e `o_address` sono indefiniti (in questa fase non importa cosa contengono) mentre gli altri segnali di output sono pari a '0'. Si rimane in questo stato finche `i_start` non viene alzato, in tal caso si passa nello stato S1.

S1 si richiede l'indirizzo della working zone 0. Quindi `o_en`='1', `o_done`='0', `o_we`='0' e `o_address`="0000000000000000" con gli enable indefiniti. Lo stato prossimo è S2.

S2-S8 si richiedono ordinatamente gli indirizzi di memoria da 1 a 7 e si memorizzano quelli da 0 a 6. `en_addr_in` è indefinito mentre `enable_reg` viene di volta in volta aggiornato al valore opportuno: se lo stato è Si, `enable_reg` vale 2^{i-2}. Per gli altri segnali si ha `o_en`='1', `o_done`='0' e `o_we`='0'. Lo stato prossimo è quello di indice successivo.

S9 si richiede il valore contenuto nell'indirizzo 8 della RAM e si memorizza il valore dell'indirizzo base della working zone 7. Il resto vale come al punto prima. Lo stato prossimo è S10.

S10 si memorizza l'indirizzo da codificare nell'opportuno registro senza scrivere o leggere nulla dalla RAM. Non si modificano i valori dei registri degli indirizzi base, cioè i loro enable sono messi a 0 mentre `en_addr_in`='1'. Lo stato prossimo è S11.

S11 si scrive l'indirizzo opportunamente codificato in memoria richiedendo di scrivere in posizione 9 quanto prodotto da out_log. `o_en`='1', `o_done`='0', `o_we`='1'. I valori di tutti i registri non sono modificati. Lo stato successivo è S12.

S12 si asserisce `o_done` il resto rimane come nello stato precedente.

S13 in questo stato si aspetta che `i_start` venga abbassato. Non si chiede né di leggere né di scrivere da memoria (`o_data` e `o_address` sono perciò indefiniti). Se `i_start` viene abbassato `o_done` viene abbassato e si passa in S14, altrimenti si mantiene `o_done` alto e si rimane in questo stato.

S14 si legge dalla RAM l'indirizzo da codificare: finché `i_start` è basso si rimane in questo stato, se viene alzato si riprende la computazione da S10.

La macchina è strutturata in modo tale da garantire, in caso di diverse richieste in sequenza, una codifica in tempi brevi. Tutte le uscite sono definite per tutti gli stati per evitare che siano inferiti dei latch, possibili cause di problemi nei circuiti reali come visto a lezione per via della trasparenza in ingresso e sostanzialmente inutili in quanto la memoria è contenuta nello stato corrente della macchina a stati.

2.3 out_log

Unica istanza del component `output_logic1` contiene a sua volta 8 istanze di `address_map`, da `addrmap0` per la prima working zone fino a `addrmap7` per l'ultima. Sia le entrate che l'uscita sono di ADDR_DIM:=8 bit. L'uscita è `out1`, mentre le entrate sono `addr_in` per l'indirizzo da codificare e 8 entrate per i valori contenuti nei registri `reg0`, `reg1`, ..., `reg7`. La descrizione è parzialmente strutturale con le già citate 8 istanze di `address_map` che producono partendo da `reg0`, `reg1`, ..., `reg7` e `addr_in` `addr_out0`, `addr_out1`, ..., `addr_out7`. Inoltre, è anche parzialmente dataflow. Si definisce un vettore `first_bits` di 8 bit che contiene i bit 7 in uscita dalla `addrmap` i in posizione i che va in input a un decoder, che effettua la trasformazione onehot-binario producendo `wz_number`, e a un demultiplexer che seleziona il segnale opportuno tra `addr_in` e la concatenazione

dell'uscita dalla `addrmap` corretta e di WZ_BIT e WZ_NUM ('1' & `wz_number`). Si è scelta tale soluzione per parallelizzare la computazione dell'uscita senza ricorrere a stati aggiuntivi. La scelta di una descrizione dataflow anziché comportamentale favorisce la semplicità del circuito e diminuisce il percorso critico non utilizzando un numero elevato di multiplexer e avendo delle tavole di verità ricche di condizioni di indeterminatezza (nel caso della conversione).

2.4 address_map

Questi moduli, istanziati progressivamente come addrmap$i$ con i il numero di working zone con base l'indirizzo nel registro al quale ciascun modulo è collegato, eseguono il mappaggio parziale tra l'indirizzo da codificare e la sua codifica. Le entrate e l'uscita, tutte di 8 bit, sono `addr_in` che contiene l'indirizzo da codificare, `reg` che indica il registro al quale si valuta l'appartenenza dell'indirizzo e `addr_out` meglio specificato nel seguito. Il modulo esegue la differenza `diff` tra `addr_in` e `reg`. Se tale differenza è compresa tra 0 e 3 e, quindi, l'indirizzo in ingresso appartiene alla working zone identificata dal registro si pone il bit 7 di `addr_out` a 1 e gli ultimi 4 bit al codice di offset appropriato lasciando i rimanenti indefiniti. Se l'indirizzo non appartiene alla working zone si pone a 0 il bit 7 e il resto viene lasciato non definito. Il leftmost bit è sempre specificato per poter definire il vettore `first_bits` nel modulo superiore. Il resto rimane indefinito per poter effettuare le opportune ottimizzazioni.

2.5 Registri

I registri usati sono registri parallelo-parallelo con un segnale di enable `en`, una entrata `in1`, una uscita `out1` e i canonici segnali di reset, `rst`, e clock, `clk`. La dimensione, modificabile con la costante ADDR_DIM, è di 8 bit. Il componente è specificato comportalmente con un processo con lista di sensitività `clk`, `rst`, `en`. Il reset è asincrono a '00000000' e l'uscita viene aggiornata sul fronte ascendente del clock solo se en = '1'.

3 Risultati sperimentali

3.1 Sintesi

Il circuito è sintetizzabile. Di seguito le parti salienti del report di sintesi.

3.1.1 synth_1_synth_synthesis_report_0 - synth_1

```
#-----------------------------------------------------------
# Vivado v2019.2.1 (64-bit)
# SW Build 2729669 on Thu Dec  5 04:49:17 MST 2019
# IP Build 2729494 on Thu Dec  5 07:38:25 MST 2019
# Start of session at: Tue Feb 25 16:57:17 2020
# Process ID: 13652
#-----------------------------------------------------------

---------------------------------------------------------------------------------
Starting Synthesize : Time (s): cpu = 00:00:05 ; elapsed = 00:00:06 . Memory (MB): peak = 574.484 ; gain = 244.891
---------------------------------------------------------------------------------

---------------------------------------------------------------------------------
Finished Synthesize : Time (s): cpu = 00:00:07 ; elapsed = 00:00:07 . Memory (MB): peak = 647.723 ; gain = 318.129
---------------------------------------------------------------------------------

---------------------------------------------------------------------------------
Finished Constraint Validation : Time (s): cpu = 00:00:07 ; elapsed = 00:00:08 . Memory (MB): peak = 647.723 ; gain =
    318.129
---------------------------------------------------------------------------------

---------------------------------------------------------------------------------
Start Loading Part and Timing Information
---------------------------------------------------------------------------------
Loading part: xc7a200tfbg484-1
---------------------------------------------------------------------------------
Finished Loading Part and Timing Information : Time (s): cpu = 00:00:07 ; elapsed = 00:00:08 . Memory (MB): peak = 647.723
     ; gain = 318.129
---------------------------------------------------------------------------------
INFO: [Device 21-403] Loading part xc7a200tfbg484-1
INFO: [Synth 8-802] inferred FSM for state register 'current_state_reg' in module 'FSM'
-----------------------------------------------------------------------------------------------------
                   State |                     New Encoding |                Previous Encoding
-----------------------------------------------------------------------------------------------------
                      s0 |                000000000000001 |                            0000
                      s1 |                000000000000010 |                            0001
                      s2 |                000000000000100 |                            0010
                      s3 |                000000000001000 |                            0011
                      s4 |                000000000010000 |                            0100
                      s5 |                000000000100000 |                            0101
                      s6 |                000000001000000 |                            0110
                      s7 |                000000010000000 |                            0111
                      s8 |                000000100000000 |                            1000
                      s9 |                000001000000000 |                            1001
                     s10 |                000010000000000 |                            1010
                     s11 |                000100000000000 |                            1011
                     s12 |                001000000000000 |                            1100
                     s13 |                010000000000000 |                            1101
                     s14 |                100000000000000 |                            1110
-----------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'one-hot' in module 'FSM'
---------------------------------------------------------------------------------
Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:08 ; elapsed = 00:00:08 . Memory (MB): peak = 647.723 ; gain =
    318.129
---------------------------------------------------------------------------------

Report RTL Partitions:
+-+--------------+-------------+----------+
| |RTL Partition |Replication  |Instances |
+-+--------------+-------------+----------+
+-+--------------+-------------+----------+
No constraint files found.
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
          3 Input       8 Bit        Adders := 8
+---Registers :
                        8 Bit     Registers := 9
+---Muxes :
         15 Input      16 Bit         Muxes := 1
         15 Input      15 Bit         Muxes := 1
          2 Input      15 Bit         Muxes := 3
          2 Input       8 Bit         Muxes := 8
          4 Input       8 Bit         Muxes := 8
         15 Input       1 Bit         Muxes := 3
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
```

```
Hierarchical RTL Component report
Module register1
Detailed RTL Component Info :
+---Registers :
                 8 Bit     Registers := 1
Module address_map
Detailed RTL Component Info :
+---Adders :
           3 Input       8 Bit        Adders := 1
+---Muxes :
           2 Input       8 Bit         Muxes := 1
           4 Input       8 Bit         Muxes := 1
Module FSM
Detailed RTL Component Info :
+---Muxes :
          15 Input      16 Bit         Muxes := 1
          15 Input      15 Bit         Muxes := 1
           2 Input      15 Bit         Muxes := 3
          15 Input       1 Bit         Muxes := 3
------------------------------------------------------------------------------
Finished RTL Hierarchical Component Statistics
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Start Part Resource Summary
------------------------------------------------------------------------------
Part Resources:
DSPs: 740 (col length:100)
BRAMs: 730 (col length: RAMB18 100 RAMB36 50)
------------------------------------------------------------------------------
Finished Part Resource Summary
------------------------------------------------------------------------------
No constraint files found.
------------------------------------------------------------------------------
Report RTL Partitions:
+-+--------------+-------------+----------+
| |RTL Partition |Replication |Instances |
+-+--------------+-------------+----------+
+-+--------------+-------------+----------+

Report Check Netlist:
+------+--------------------+--------+----------+--------+--------------------+
|      |Item                |Errors  |Warnings  |Status  |Description         |
+------+--------------------+--------+----------+--------+--------------------+
|1     |multi_driven_nets   |       0|         0|Passed  |Multi driven nets   |
+------+--------------------+--------+----------+--------+--------------------+
------------------------------------------------------------------------------

Report Cell Usage:
+------+--------+------+
|      |Cell    |Count |
+------+--------+------+
|1     |BUFG    |     1|
|2     |CARRY4  |    16|
|3     |LUT2    |    68|
|4     |LUT3    |     3|
|5     |LUT4    |     9|
|6     |LUT5    |     1|
|7     |LUT6    |    34|
|8     |FDCE    |    86|
|9     |FDPE    |     1|
|10    |IBUF    |    11|
|11    |OBUF    |    27|
+------+--------+------+

Report Instance Areas:
+------+---------------+----------------+------+
|      |Instance       |Module          |Cells |
+------+---------------+----------------+------+
|1     |top            |                |   257|
|2     |  out_log      |output_logic1   |    41|
|3     |    addrmap0   |address_map     |     5|
|4     |    addrmap1   |address_map_8   |     3|
|5     |    addrmap2   |address_map_9   |     4|
|6     |    addrmap3   |address_map_10  |     5|
|7     |    addrmap4   |address_map_11  |     4|
|8     |    addrmap5   |address_map_12  |     7|
|9     |    addrmap6   |address_map_13  |     6|
|10    |    addrmap7   |address_map_14  |     2|
|11    |  FSM1         |FSM             |    27|
|12    |  reg0         |register1       |     8|
|13    |  reg1         |register1_0     |     8|
|14    |  reg2         |register1_1     |     8|
|15    |  reg3         |register1_2     |     8|
```

```
|16     |   reg4         |register1_3     |      8|
|17     |   reg5         |register1_4     |      8|
|18     |   reg6         |register1_5     |      8|
|19     |   reg7         |register1_6     |      8|
|20     |   reg_in       |register1_7     |     86|
+-------+----------------+----------------+-------+
```

3.1.2 synth_1_synth_report_utilization_0

```
Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------------------------------------------------------------

| Tool Version : Vivado v.2019.2.1 (win64) Build 2729669 Thu Dec  5 04:49:17 MST 2019
| Date         : Tue Feb 25 16:58:24 2020
| Host         : DESKTOP-7BRK978 running 64-bit major release  (build 9200)
| Command      : report_utilization -file project_reti_logiche_utilization_synth.rpt -pb
    project_reti_logiche_utilization_synth.pb
| Design       : project_reti_logiche
| Device       : 7a200tfbg484-1
| Design State : Synthesized
------------------------------------------------------------------------------------------------------------------------------------------------------


Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  111 |     0 |    134600 |  0.08 |
|   LUT as Logic          |  111 |     0 |    134600 |  0.08 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |   87 |     0 |    269200 |  0.03 |
|   Register as Flip Flop |   87 |     0 |    269200 |  0.03 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |    0 |     0 |     67300 |  0.00 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt_design
    after synthesis, if not already completed, for a more realistic count.


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            - |           - |            - |
| 0     |            - |           - |          Set |
| 0     |            - |           - |        Reset |
| 0     |            - |         Set |            - |
| 0     |            - |       Reset |            - |
| 0     |          Yes |           - |            - |
| 1     |          Yes |           - |          Set |
| 86    |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 0     |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+


2. Memory
---------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       365 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       365 |  0.00 |
|   RAMB18       |    0 |     0 |       730 |  0.00 |
+----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one
    FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1
```

3. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	740	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	38	0	285	13.33
Bonded IPADs	0	0	14	0.00
Bonded OPADs	0	0	8	0.00
PHY_CONTROL	0	0	10	0.00
PHASER_REF	0	0	10	0.00
OUT_FIFO	0	0	40	0.00
IN_FIFO	0	0	40	0.00
IDELAYCTRL	0	0	10	0.00
IBUFDS	0	0	274	0.00
GTPE2_CHANNEL	0	0	4	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	40	0.00
PHASER_IN/PHASER_IN_PHY	0	0	40	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	500	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	285	0.00
OLOGIC	0	0	285	0.00

5. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	1	0	32	3.13
BUFIO	0	0	40	0.00
MMCME2_ADV	0	0	10	0.00
PLLE2_ADV	0	0	10	0.00
BUFMRCE	0	0	20	0.00
BUFHCE	0	0	120	0.00
BUFR	0	0	40	0.00

6. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
PCIE_2_1	0	0	1	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDCE	86	Flop & Latch
LUT2	68	LUT
LUT6	34	LUT
OBUF	27	IO
CARRY4	16	CarryLogic
IBUF	11	IO
LUT4	9	LUT

LUT3	3	LUT
LUT5	1	LUT
FDPE	1	Flop & Latch
BUFG	1	Clock

8. Black Boxes

Ref Name	Used

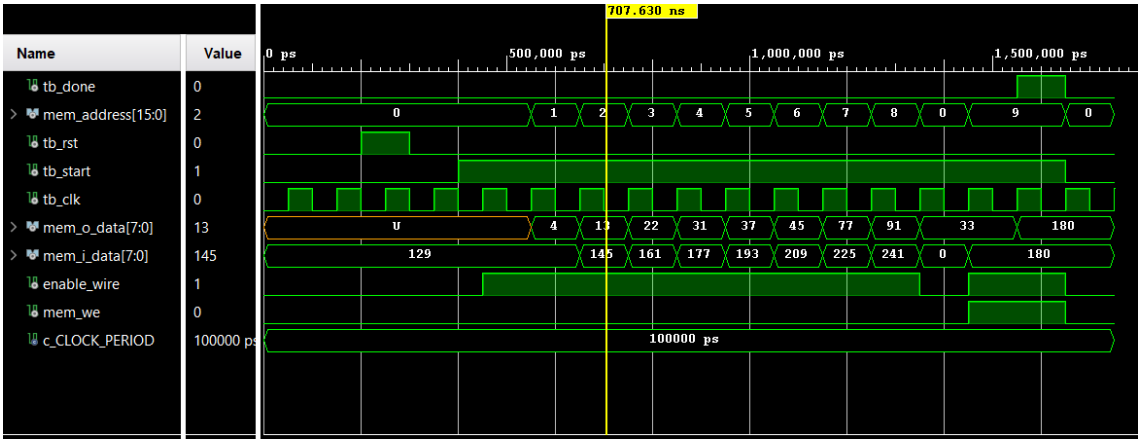
9. Instantiated Netlists

Ref Name	Used

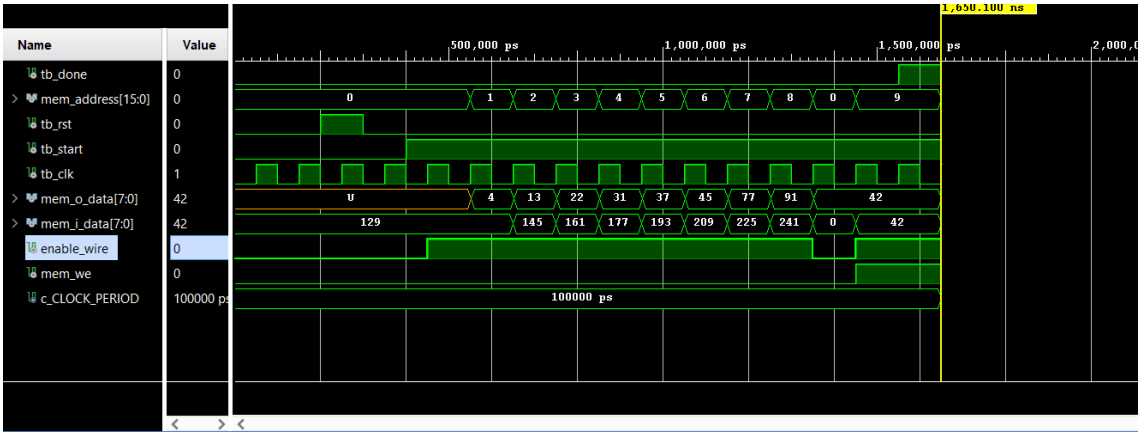
3.2 Simulazioni

Il componente ha prodotto gli output corretti nei testbench forniti e in quelli definiti sia nella simulazione Behavioural che nella Functional Post-Synthesis. Di seguito i grafici relativi alle simulazioni.

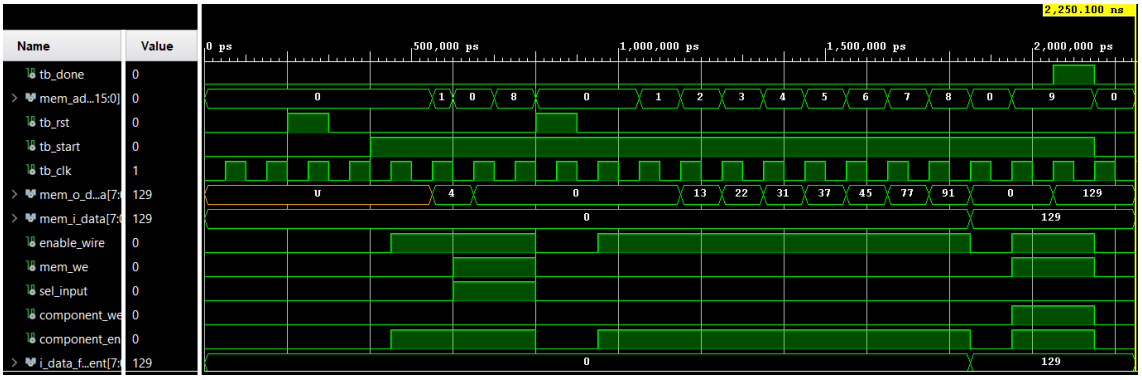
3.2.1 Indirizzo appartenente a una working zone



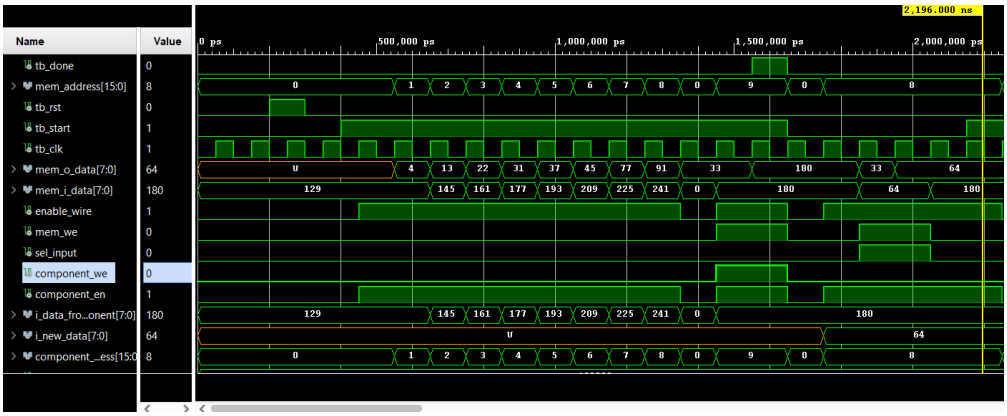
3.2.2 Indirizzo non appartenente a una working zone



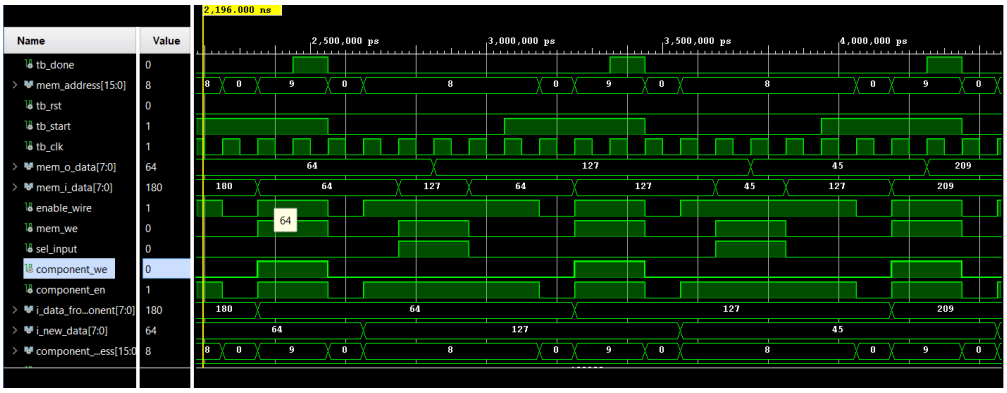
3.2.3 Reset multipli



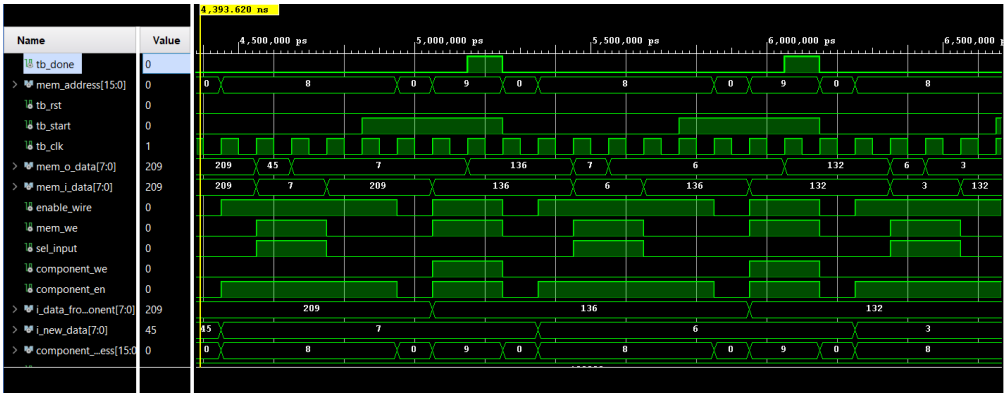
3.2.4 Richieste multiple



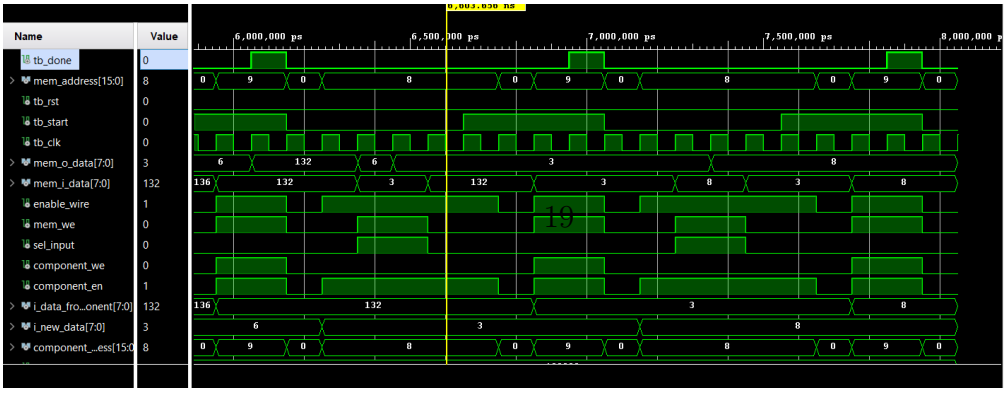
(a)



(b)



(c)



(d)

4 Test Bench

Oltre ai test di esempio che testano la correttezza del comportamento del componente in caso di indirizzo contenuto in una working zone e è stato approntato un altro test bench:

4.1 tb_pfrl_2020_repeated

Verifica che in caso di richiesta di una nuova codifica per un indirizzo in posizione 8 della RAM funzioni correttamente. Verifica, quindi, sia il caso che l'indirizzo appartenga a una working zone che il caso che non appartenga. Si testano i seguenti casi limite:

1. indirizzo non appartenente ad alcuna working zone;
2. indirizzo appartenente a una working zone in posizione 0001;
3. indirizzo appartenente a una working zone in posizione 0010;
4. indirizzo appartenente a una working zone in posizione 0100;
5. indirizzo appartenente a una working zone in posizione 1000;
6. indirizzo non appartenente a una working zone ma precedente all'indirizzo base di qualcuna;
7. indirizzo non appartenente a una working zone ma immediatamente successivo ad essa.

In questo modo si verifica, inoltre, che la working zone abbia esattamente dimensione 4, che gli indirizzi interni a essa siano ben codificati e che quelli immediatamente precedenti o successivi non siano inclusi in essa.

Listing 1: tb_pfrl_2020_repeated.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity project_tb is
end project_tb;

architecture projecttb of project_tb is
constant c_CLOCK_PERIOD         : time := 100 ns;
signal   tb_done                : std_logic;
signal   mem_address            : std_logic_vector (15 downto 0) := (others => '0');
signal   tb_rst                 : std_logic := '0';
signal   tb_start               : std_logic := '0';
signal   tb_clk                 : std_logic := '0';
signal   mem_o_data,mem_i_data  : std_logic_vector (7 downto 0);
signal   enable_wire            : std_logic;
signal   mem_we                 : std_logic;
signal   sel_input, component_we, component_en: std_logic := '0'; --propri
signal   i_data_from_component, i_new_data : std_logic_vector(7 downto 0);
```

```
signal component_address : std_logic_vector(15 downto 0);

type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);

-- come da esempio su specifica
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 4 , 8)),
                         1 => std_logic_vector(to_unsigned( 13 , 8)),
                         2 => std_logic_vector(to_unsigned( 22 , 8)),
                         3 => std_logic_vector(to_unsigned( 31 , 8)),
                         4 => std_logic_vector(to_unsigned( 37 , 8)),
                         5 => std_logic_vector(to_unsigned( 45 , 8)),
                         6 => std_logic_vector(to_unsigned( 77 , 8)),
                         7 => std_logic_vector(to_unsigned( 91 , 8)),
                         8 => std_logic_vector(to_unsigned( 33 , 8)),
                         others => (others =>'0'));

component project_reti_logiche is
port (
      i_clk         : in  std_logic;
      i_start       : in  std_logic;
      i_rst         : 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 component project_reti_logiche;


begin
UUT: project_reti_logiche
port map (
          i_clk         => tb_clk,
          i_start       => tb_start,
          i_rst         => tb_rst,
          i_data        => mem_o_data,
          o_address     => component_address,
          o_done        => tb_done,
          o_en          => component_en,
          o_we          => component_we,
          o_data        => i_data_from_component
          );

with sel_input select
    mem_i_data <= i_data_from_component when '0',
                  i_new_data when others;

with sel_input select
    mem_we <=  component_we when '0',
               '1' when others;

with sel_input select
    enable_wire <= component_en when '0',
                   '1' when others;

with sel_input select
    mem_address <= component_address when '0',
                   std_logic_vector(to_unsigned(8, 16)) when others;

p_CLK_GEN : process is
begin
    wait for c_CLOCK_PERIOD/2;
    tb_clk <= not tb_clk;
end process p_CLK_GEN;


MEM : process(tb_clk)
begin
    if tb_clk'event and tb_clk = '1' then
        if enable_wire = '1' then
            if mem_we = '1' then
                RAM(conv_integer(mem_address))  <= mem_i_data;
                mem_o_data                      <= mem_i_data after 1 ns;
            else
                mem_o_data <= RAM(conv_integer(mem_address)) after 1 ns;
            end if;
        end if;
    end if;
end process;
```

```
test : process is
begin
    wait for 100 ns;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
    wait for 100 ns;

    -- Maschera di output = 1 - 011 - 0100
    assert RAM(9) = std_logic_vector(to_unsigned( 180 , 8)) report "TEST_FALLITO. Expected  180
        found " &
 integer'image(to_integer(unsigned(RAM(9))))  severity failure;


    i_new_data <= std_logic_vector(to_unsigned(64, 8));
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
    wait for 100 ns;


    -- Maschera di output = 0 - 1000000
    assert RAM(9) = std_logic_vector(to_unsigned( 64 , 8)) report "TEST_FALLITO. Expected  64
        found " &
 integer'image(to_integer(unsigned(RAM(9))))  severity failure;

    i_new_data <= std_logic_vector(to_unsigned(127, 8));
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
    wait for 100 ns;

    -- Maschera di output = 0 - 1111111
    assert RAM(9) = std_logic_vector(to_unsigned( 127 , 8)) report "TEST_FALLITO. Expected  209
        found " &
 integer'image(to_integer(unsigned(RAM(9))))  severity failure;


    i_new_data <= std_logic_vector(to_unsigned(45, 8));
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
    wait for 100 ns;

    -- Maschera di output = 1 - 101 - 0001
```

```
    assert RAM(9) = std_logic_vector(to_unsigned( 209 , 8)) report "TEST_FALLITO. _Expected__209
        __found_" &
integer'image(to_integer(unsigned(RAM(9))))  severity failure;


    i_new_data <= std_logic_vector(to_unsigned(7, 8));
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
    wait for 100 ns;

    -- Maschera di output = 1 - 000 - 1000
    assert RAM(9) = std_logic_vector(to_unsigned( 136 , 8)) report "TEST_FALLITO. _Expected__136
        __found_" &
integer'image(to_integer(unsigned(RAM(9))))  severity failure;


    i_new_data <= std_logic_vector(to_unsigned(6, 8));
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
    wait for 100 ns;

    -- Maschera di output = 1 - 000 - 0100
    assert RAM(9) = std_logic_vector(to_unsigned( 132, 8)) report "TEST_FALLITO. _Expected__132_
        _found_" &
integer'image(to_integer(unsigned(RAM(9))))  severity failure;


    i_new_data <= std_logic_vector(to_unsigned(3, 8));
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';
    wait for 100 ns;

    -- Maschera di output = 0 - 0000011
    assert RAM(9) = std_logic_vector(to_unsigned( 3, 8)) report "TEST_FALLITO. _Expected__132__
        found_" &
integer'image(to_integer(unsigned(RAM(9))))  severity failure;

    i_new_data <= std_logic_vector(to_unsigned(8, 8));
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
```

```
    wait until tb_done = '0';
    wait for 100 ns;

    -- Maschera di output = 0 - 0001000
    assert RAM(9) = std_logic_vector(to_unsigned( 8 , 8)) report "TEST FALLITO. Expected  8  
        found " &
 integer'image(to_integer(unsigned(RAM(9))))  severity failure;


    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
end process test;

end projecttb;
```

4.2 tb_pfrl_2020_multi_reset

Questo test valuta il comportamento del componente in caso di reset che ridefinisca le working zone. Dopo aver iniziato la fase di elaborazione riceve un reset che cambia l'indirizzo da codificare e quello di una working zone. Il modulo correttamente ricomincia la computazione ricaricando gli indirizzi base e quello da codificare.

Listing 2: tb_pfrl_2020_repeated.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity project_tb is
end project_tb;

architecture projecttb of project_tb is
constant c_CLOCK_PERIOD         : time := 100 ns;
signal   tb_done                : std_logic;
signal   mem_address            : std_logic_vector (15 downto 0) := (others => '0');
signal   tb_rst                 : std_logic := '0';
signal   tb_start               : std_logic := '0';
signal   tb_clk                 : std_logic := '0';
signal   mem_o_data,mem_i_data  : std_logic_vector (7 downto 0);
signal   enable_wire            : std_logic;
signal   mem_we                 : std_logic;
signal   sel_input, component_we, component_en: std_logic := '0'; --propri
signal   i_data_from_component, i_new_data : std_logic_vector(7 downto 0);
signal component_address, simulation_address : std_logic_vector(15 downto 0);

type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);

-- come da esempio su specifica
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 4 , 8)),
                         1 => std_logic_vector(to_unsigned( 13 , 8)),
                         2 => std_logic_vector(to_unsigned( 22 , 8)),
                         3 => std_logic_vector(to_unsigned( 31 , 8)),
                         4 => std_logic_vector(to_unsigned( 37 , 8)),
                         5 => std_logic_vector(to_unsigned( 45 , 8)),
                         6 => std_logic_vector(to_unsigned( 77 , 8)),
                         7 => std_logic_vector(to_unsigned( 91 , 8)),
                         8 => std_logic_vector(to_unsigned( 33 , 8)),
                         others => (others =>'0'));

component project_reti_logiche is
port (
      i_clk         : in  std_logic;
      i_start       : in  std_logic;
      i_rst         : 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 component project_reti_logiche;
```

```
begin
UUT: project_reti_logiche
port map (
          i_clk         => tb_clk,
          i_start       => tb_start,
          i_rst         => tb_rst,
          i_data        => mem_o_data,
          o_address     => component_address,
          o_done        => tb_done,
          o_en          => component_en,
          o_we          => component_we,
          o_data        => i_data_from_component
          );

with sel_input select
    mem_i_data <= i_data_from_component when '0',
                  i_new_data when others;

with sel_input select
    mem_we <=  component_we when '0',
               '1' when others;

with sel_input select
    enable_wire <= component_en when '0',
                   '1' when others;

with sel_input select
    mem_address <= component_address when '0',
                   simulation_address when others;

p_CLK_GEN : process is
begin
    wait for c_CLOCK_PERIOD/2;
    tb_clk <= not tb_clk;
end process p_CLK_GEN;


MEM : process(tb_clk)
begin
    if tb_clk'event and tb_clk = '1' then
        if enable_wire = '1' then
            if mem_we = '1' then
                RAM(conv_integer(mem_address))  <= mem_i_data;
                mem_o_data                      <= mem_i_data after 1 ns;
            else
                mem_o_data <= RAM(conv_integer(mem_address)) after 1 ns;
            end if;
        end if;
    end if;
end process;


test : process is
begin
    wait for 100 ns;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    sel_input <= '1';
    i_new_data <= std_logic_vector(to_unsigned(0, 8));
    simulation_address <= std_logic_vector(to_unsigned(0, 16));
    wait for c_CLOCK_PERIOD;
    i_new_data <= std_logic_vector(to_unsigned(0, 8));
    simulation_address <= std_logic_vector(to_unsigned(8, 16));
    wait for c_CLOCK_PERIOD;
    sel_input <= '0';
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
```

```
    wait until tb_done = '0';
    wait for 100 ns;


    -- Maschera di output = 1 - 000 - 0001
    assert RAM(9) = std_logic_vector(to_unsigned(129, 8)) report "TEST FALLITO. Expected  64  
        found " & integer'image(to_integer(unsigned(RAM(9))))  severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
end process test;

end projecttb;
```

5 Conclusioni

Il componente è stato progettato cercando di comprimere il tempo necessario per poter fornire la codifica richiesta, specie nel caso di richieste ripetute. Perciò si è preferito, dove possibile, adottare una logica combinatoria che parallelizzi la computazione. Questo comporta circuiti fisicamente più grandi, relativamente a una logica puramente sequenziale o che acceda di volta in volta alla RAM chiedendo l'indirizzo necessario. Ne consegue che i vantaggi maggiori si hanno in situazioni in cui il reset del componente e l'aggiornamento di tutti gli indirizzi in memoria sono relativamente meno frequenti rispetto a codifiche di diversi indirizzi, a parità di working zone, in sequenza. Il componente sembra comportarsi in modo corretto nelle condizioni considerate. Nel progetto si è utilizzato un approccio modulare che disaccoppia i componenti a logica prevalentemente combinatoria dalla memoria e dalla macchina a stati finiti dimodoché possano essere modificati in modo indipendente.