Prova Finale (Progetto) di Reti Logiche

Relazione del lavoro svolto

Andrea Caravano

Anno Accademico 2022–23

Indice

1 Introduzione

1.1 Descrizione ad alto livello

Nell'ambito della Prova Finale (Progetto) di Reti Logiche, è richiesto lo sviluppo di un componente hardware implementabile su FPGA (dispositivo logico programmabile), quali quelle della serie Artix-7 di Xilinx.
Il software che si richiede di utilizzare per lo sviluppo, simulazione e sintesi (simulata virtualmente) è anch'esso di Xilinx, Vivado.

Il componente ha accesso (attraverso segnali opportuni) ad una memoria, che provvede, fornito un indirizzo da 16 bit, ad esporre in uscita i corrispettivi 8 bit di dato, parallelamente.
Esso fa inoltre uso dei segnali principali `START` e `W`, caratterizzanti, rispettivamente, la validità della sequenza di bit e la sequenza effettiva, suddivisa in `Identificativo del canale` (2 *bit*, sempre presenti) e seguiti dall'`Indirizzo di memoria` (16 *bit*, al più).
Se i bit di indirizzo non sono completi, il componente provvede a completarli con dei bit di `padding`, anticausalmente alla sequenza (in cima).

L'identificativo di canale fornito è utilizzato per esporre, su uno dei 4 segnali di uscita (Z_0, Z_1, Z_2, Z_3) il dato letto dalla memoria all'indirizzo specificato.
L'elaborazione necessaria avviene entro 20 cicli di clock, con l'attivazione (per un solo ciclo di clock, quello in cui il dato viene esposto) di un segnale `DONE`, che viene disattivato nuovamente al termine.
Durante questa fase, il componente si "ricorda" dei dati precedentemente esposti sui 4 segnali di uscita (dall'ultimo reset) e, se non sovrascritti dall'ultima lettura, li espone nuovamente.
Non vi sono verifiche ulteriori da parte del componente relativamente alla violazione della specifica (una sequenza `W` più corta di 2 bit o un indirizzo più lungo di 16 bit).

1.2 Un primo esempio

Nell'esempio mostrato, (esteso ulteriormente in seguito), viene prelevato dalla memoria il dato presente all'indirizzo 19 (in binario, 10011) ed esposto sul canale Z_0 (identificato da `00`, in binario).
In memoria è stato predisposto il valore 134 (esemplificativo) a tale indirizzo.
A seguito del segnale di reset, il componente fa uso del seguente schema:

`START` = 1	
Identificativo del canale	Indirizzo di memoria

Quindi, in questo caso:

`START` = 1	
00 (Z_0)	10011 (19)

Il segnale di `START` torna dunque a 0.
Entro 20 cicli, il componente si occupa di fornire alla memoria l'indirizzo da cui è intenzionato a prelevare il dato e lo preleva, esponendolo sull'uscita corrispettiva all'identificativo di cancale richiesto.
La fase di valorizzazione dei segnali di uscita dura 1 ciclo di clock e segue lo schema seguente:

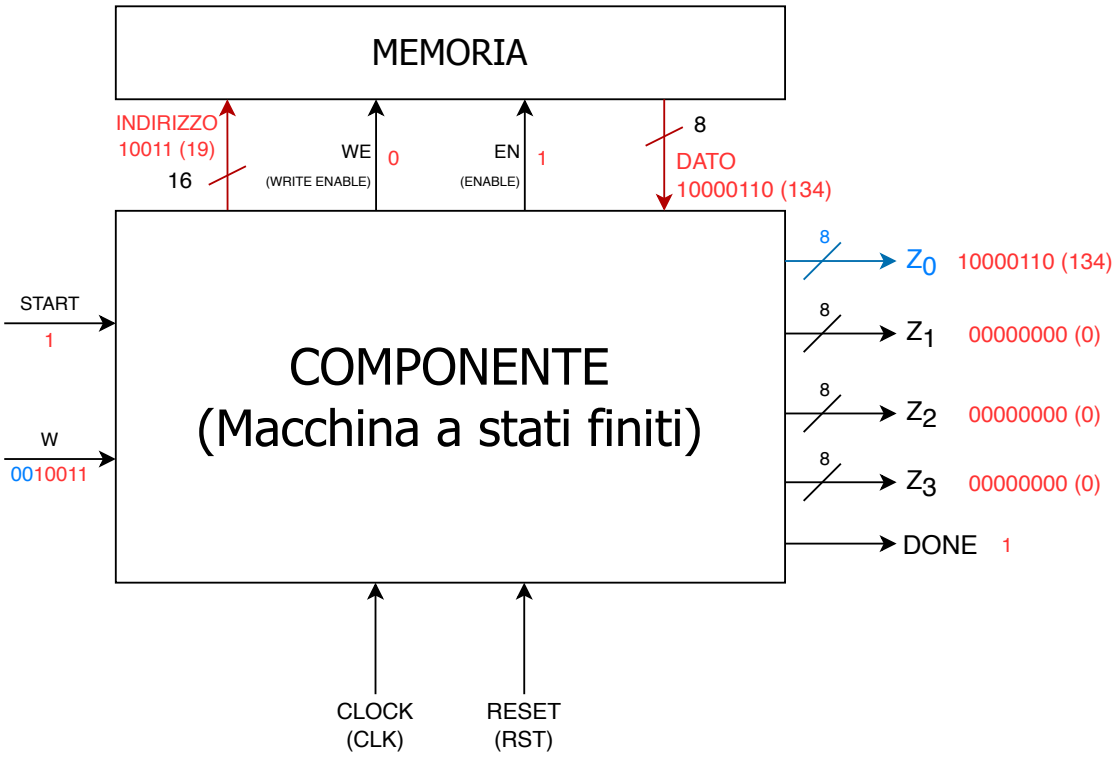
`DONE` = 1			
Z_0	Z_1	Z_2	Z_3
x	y	z	w

dove
x = valore assunto da Z_0 l'ultima volta che il segnale di `DONE` è stato = 1 (0 altrimenti).
y = valore assunto da Z_1 l'ultima volta che il segnale di `DONE` è stato = 1 (0 altrimenti).
z = valore assunto da Z_2 l'ultima volta che il segnale di `DONE` è stato = 1 (0 altrimenti).
w = valore assunto da Z_3 l'ultima volta che il segnale di `DONE` è stato = 1 (0 altrimenti).

Quindi, nel nostro caso:

`DONE` = 1			
Z_0	Z_1	Z_2	Z_3
10000110 (134)	00000000 (0)	00000000 (0)	00000000 (0)

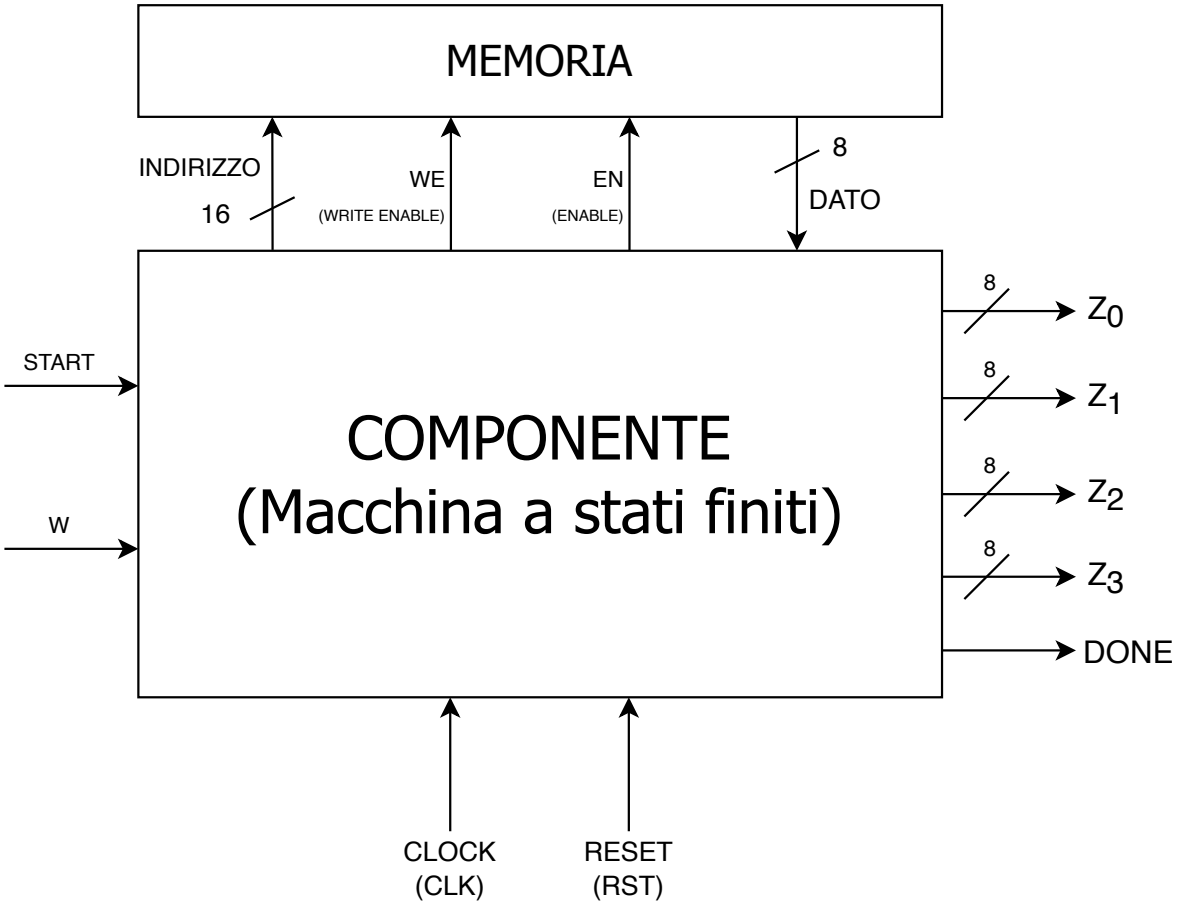
Viene infine mostrato di seguito lo schema riepilogativo durante l'esposizione del dato (`DONE` = 1):



2 Architettura

2.1 Astrazione

L'implementazione astratta del componente è realizzata come mostrato:

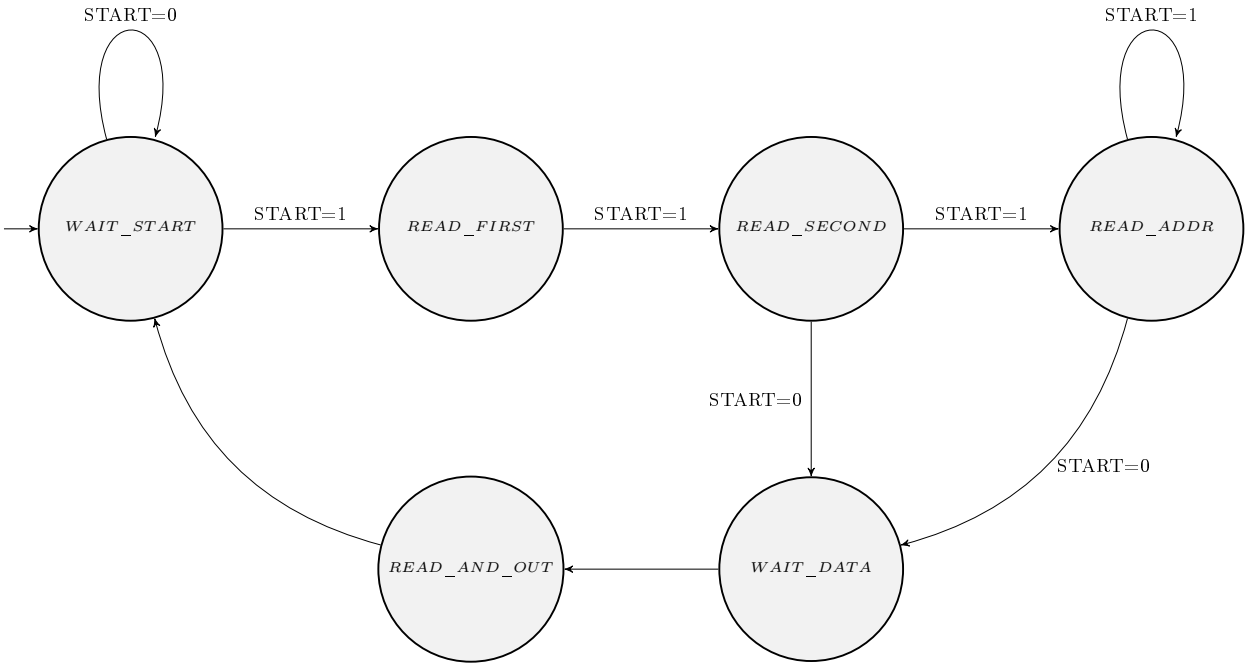


I cui segnali di riferimento sono:

Segnale	Dimensione (*bit*)	Funzione
`CLOCK (CLK)`	1	Segnale di clock
`RESET (RST)`	1	Segnale di reset
`START`	1	Delimitatore di validità della sequenza (`W`)
`W`	1	Sequenza aggregata effettiva, suddivisa in `Identificativo del canale` e `Indirizzo di memoria`
Indirizzo	16	Trasmesso alla memoria, derivante dalla Sequenza aggregata (eventualmente completato)
`Write Enable(WE)`	1	Attivo quando è richiesto di scrivere dati in memoria. Nell'ambito di questo progetto, è sempre disattivo (impostato a 0).
`Enable (EN)`	1	Attivo quando è richiesto in generale l'uso della memoria. Nell'ambito di questo progetto, è sempre attivo (impostato a 1).
Dato	8	Letto dalla memoria, in corrispondenza dell'indirizzo indicato. Esso viene reso effettivamente disponibile dopo un ciclo di clock rispetto all'indicazione alla memoria dell'indirizzo.
Z_0	8	Canale di uscita corrispondente all'identificativo di canale `00`
Z_1	8	Canale di uscita corrispondente all'identificativo di canale `01`
Z_2	8	Canale di uscita corrispondente all'identificativo di canale `10`
Z_3	8	Canale di uscita corrispondente all'identificativo di canale `11`
`DONE`	1	Attivo per un ciclo di clock durante la fase di esposizione del dato trovato in memoria all'indirizzo fornito (ed eventualmente degli ultimi valori esposti precedentemente in questa fase)

2.2 Macchina a stati finiti

La macchina a stati finiti fa uso del seguente diagramma di stato:



Implementati come descritto nel seguito.

2.2.1 `WAIT_START`

Stato iniziale del ciclo, i segnali di `DONE`, i canali di uscita, il buffer dell'indirizzo e il contatore della relativa lunghezza vengono tutti reimpostati al valore predefinito (0).
Tale stato viene mantenuto finché il segnale di `START` (delimitatore di validità della sequenza) rimane 0.
In caso contrario, si prosegue nella lettura del primo bit componente l'`Identificativo del canale` di uscita.

2.2.2 `READ_FIRST`

In corrispondenza dell'attivazione del segnale di `START`, viene letto il primo bit che compone l'`Identificativo del canale` di uscita.

2.2.3 `READ_SECOND`

Viene letto il secondo bit componente l'`Identificativo del canale` di uscita.
Come da specifica, non sono previste transizioni nel caso in cui i segnali ricevuti non rispettino le indicazioni sulla durata minima (o massima) della sequenza `W`, che si tradurrebbero comunque nella permanenza in uno stato di errore. Si noti inoltre che l'indirizzo può essere vuoto (0 *bit*).

2.2.4 `READ_ADDR`

Fino a quando il segnale di `START` rimane a 1 (indicante la validità della sequenza `W`), vengono memorizzati i bit che compongono l'indirizzo in un buffer temporaneo, seguendo l'ordine di arrivo.
In questa fase non vengono effettuate traslazioni dell'indirizzo per eventuali `bit di padding`, che si rimandano invece alla successiva.

2.2.5 `WAIT_DATA`

Stato di attesa del ciclo di clock necessario alla memoria per esporre il dato sul corrispondente segnale.
L'indirizzo fornito è comprensivo di eventuali `bit di padding` necessari per completare i 16 richiesti.

2.2.6 READ_AND_OUT

Fase in cui il segnale `DONE` è attivo.
Il dato trovato in memoria all'indirizzo specificato viene esposto sul canale scelto, gli altri canali (solo in questa fase) espongono l'ultimo valore esposto in questa fase, 0 altrimenti.
Le uscite seguono dunque lo schema esposto nell'esempio introduttivo.
Al termine, il segnale `DONE` viene posto nuovamente a 0 e il componente ritorna nella fase iniziale (`WAIT_START`).

3 Risultati sperimentali

La specifica del componente è stata implementata in VHDL su Xilinx Vivado 2018.3.1, attraverso la piattaforma di Virtual Desktop Citrix offerta dal Politecnico.
Sempre in Vivado sono state realizzate varie configurazioni di test bench, come esposto nel seguito.

3.1 Sintesi

Di seguito, viene riportato il report della sintesi, in cui viene mostrato il numero di registri utilizzato.
Si noti inoltre, come raccomandato a lezione, l'assenza di Latch, che potrebbero indicare problemi nella gestione dei segnali del componente.

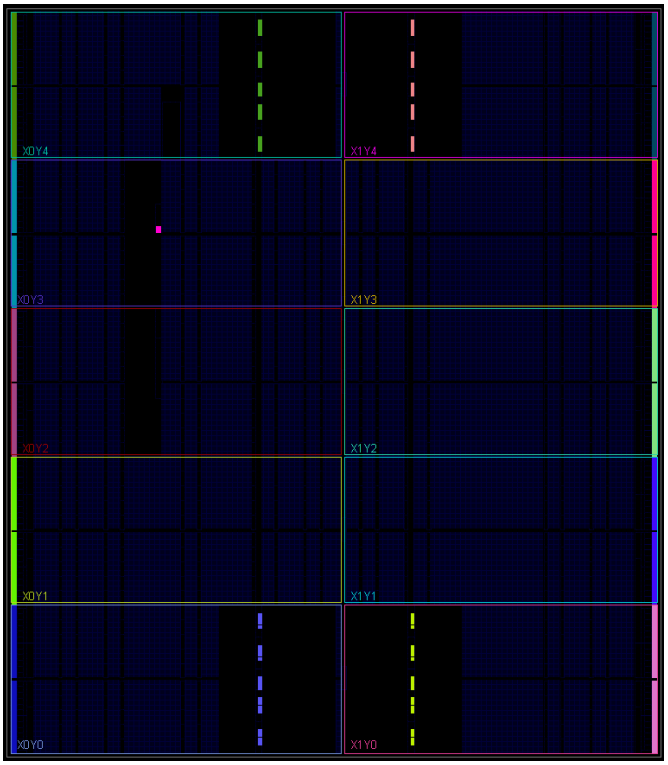
Site Type	Used	Fixed	Available	Util%
Slice LUTs*	108	0	134600	0.08
LUT as Logic	108	0	134600	0.08
LUT as Memory	0	0	46200	0.00
Slice Registers	107	0	269200	0.04
Register as Flip Flop	107	0	269200	0.04
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Inoltre, il componente non presenta problemi legati al timing, come mostrato:

```
Timing Report

Slack (MET) :             96.937ns  (required time - arrival time)
```

Dalla sintesi, completata con successo, deriva lo schema seguente:



3.2 Simulazioni

Sono state realizzate diverse configurazioni di casi di test, a partire dal test bench di riferimento fornito. Nel seguito, ne vengono mostrati alcuni esempi.

3.2.1 Test bench n. 1: esempio introduttivo

Implementa l'esempio introduttivo.
In particolare, a seguito del segnale di reset, ci si aspetta, a seguire i due casi di test iniziali, il primo introdotto:

START = 1	
00 (Z_0)	10011 (19)

DONE = 1			
Z_0	Z_1	Z_2	Z_3
10000110 (134)	01011000 (88)	10100010 (162)	00000000 (0)

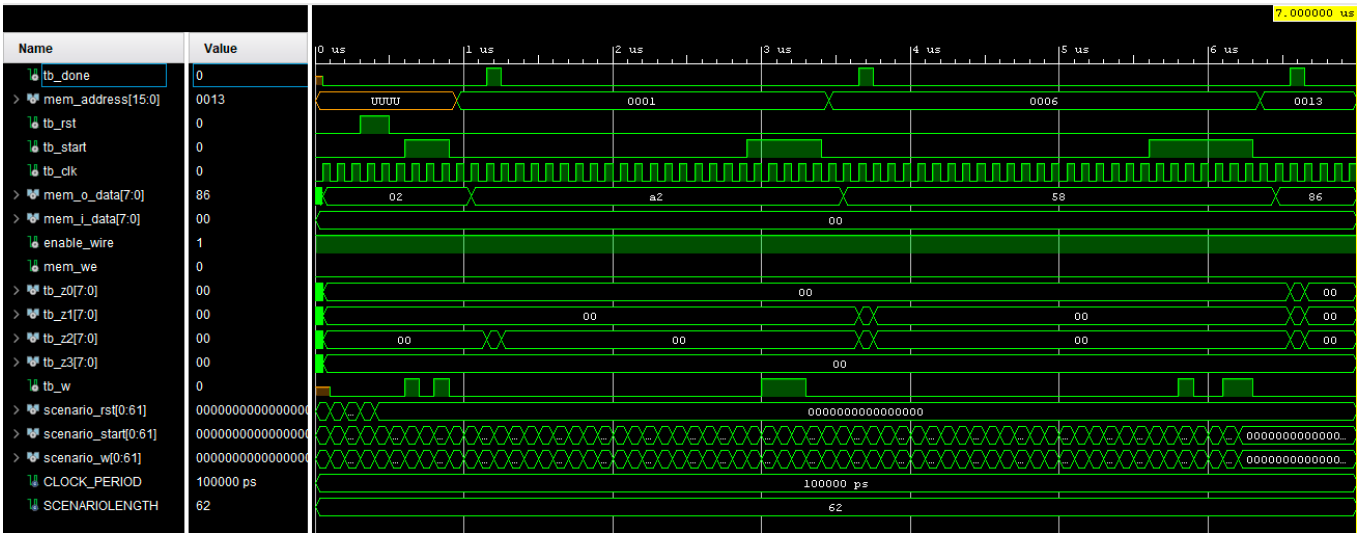
Che corrisponde alla seguente combinazione di segnali:

```
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)      := "00110" & "000" &
↪  "00000000000000000000" & "0000000" & "00000000000000000000" & "0000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1)    := "00000"
                        & "111" & "00000000000000000000"
                        & "1111100" & "00000000000000000000"
                        & "1111111";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)        := "00000"
                        & "101" & "00000000000000000000"
                        & "0111000" & "00000000000000000000"
                        & "0010011";
```

E con memoria istanziata:

```
TYPE ram_type IS ARRAY (65535 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : ram_type := (  0 => STD_LOGIC_VECTOR(to_unsigned(2, 8)),
                            1 => STD_LOGIC_VECTOR(to_unsigned(162, 8)),
                            2 => STD_LOGIC_VECTOR(to_unsigned(75, 8)),
                            3 => STD_LOGIC_VECTOR(to_unsigned(175, 8)),
                            6 => STD_LOGIC_VECTOR(to_unsigned(88, 8)),
                            19 => STD_LOGIC_VECTOR(to_unsigned(134, 8)), -- aggiunto il
                            ↪  valore 134 all'indirizzo 19
                            OTHERS => "00000000"-- (OTHERS => '0')
                        );
```

L'esecuzione porta dunque al risultato atteso:



3.2.2 Test bench n. 2: indirizzo vuoto e sovrascrittura

Si testa ora la capacità del componente di reagire correttamente ad un indirizzo vuoto (0 *bit*).
Ci si aspetta esso corrisponda all'indirizzo 0 di memoria, presso cui è stato posto il valore di test 135.
Si sceglie inoltre di utilizzare il canale `10` (Z_2), verificando anche la corretta sovrascrittura del valore precedente (162).
Quindi, ci si aspetta che la nuova sequenza sia:

START = 1	
10 (Z_2)	ƀ (0)

DONE = 1			
Z_0	Z_1	Z_2	Z_3
10000110 (134)	01011000 (88)	10000111 (135)	00000000 (0)

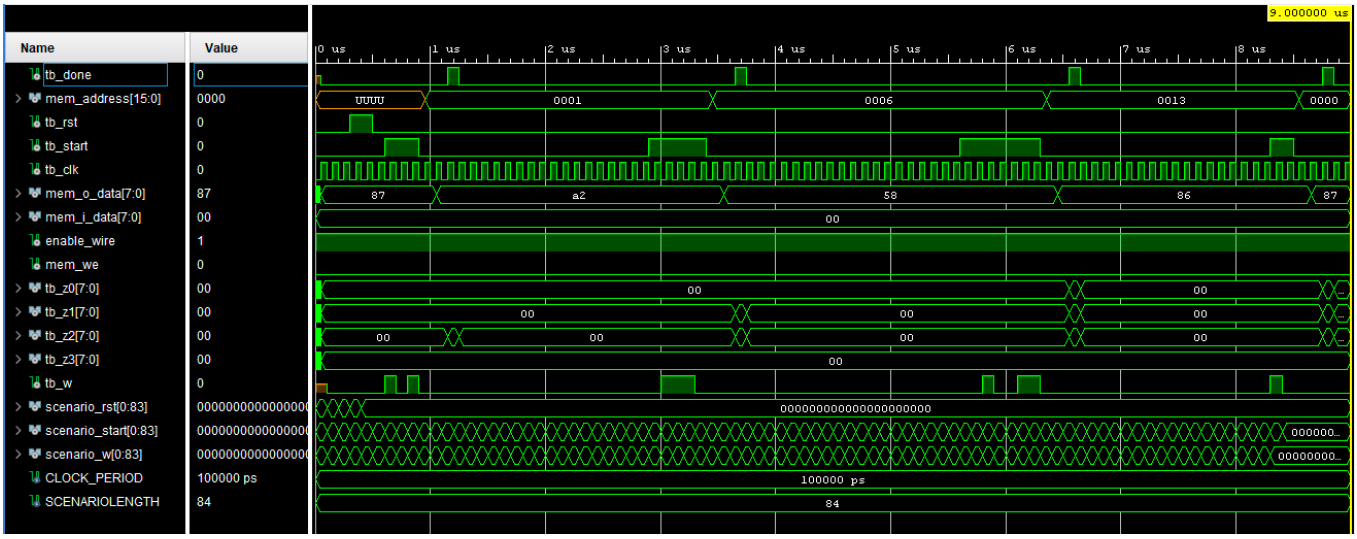
Che corrisponde alla seguente combinazione di segnali:

```
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)      := "00110" & "000" &
↪  "00000000000000000000" & "0000000" & "00000000000000000000" & "0000000" &
↪  "00000000000000000000" & "00";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1)    := "00000"
                        & "111" & "00000000000000000000"
                        & "1111100" & "00000000000000000000"
                        & "1111111" & "00000000000000000000"
                        & "11";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)        := "00000"
                        & "101" & "00000000000000000000"
                        & "0111000" & "00000000000000000000"
                        & "0010011" & "00000000000000000000"
                        & "10";
```

E con memoria istanziata:

```
TYPE ram_type IS ARRAY (65535 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : ram_type := (  0 => STD_LOGIC_VECTOR(to_unsigned(135, 8)), -- posto in
↪  memoria 135 all'indirizzo 0
                            1 => STD_LOGIC_VECTOR(to_unsigned(162, 8)),
                            2 => STD_LOGIC_VECTOR(to_unsigned(75, 8)),
                            3 => STD_LOGIC_VECTOR(to_unsigned(175, 8)),
                            6 => STD_LOGIC_VECTOR(to_unsigned(88, 8)),
                            19 => STD_LOGIC_VECTOR(to_unsigned(134, 8)),
                            OTHERS => "00000000"-- (OTHERS => '0')
                          );
```

L'esecuzione porta dunque al risultato atteso:



3.2.3 Test bench n. 3: indirizzo completo e reset

Si testa ora la capacità del componente di reagire correttamente ad un indirizzo completo (16 *bit*).
Ci si aspetta esso corrisponda all'indirizzo 1000100111111011 (35323) di memoria, presso cui è stato posto il valore di test 142.
Si sceglie di utilizzare il canale `11` (Z_3).
Si noti che l'indirizzo scelto è volutamente non simmetrico o palindromo.
Prima di procedere, tuttavia, si testa anche l'abilità del componente di reagire correttamente ad un secondo reset, antecedente la lettura di quest'ultima sequenza.
Quindi, ci si aspetta che la nuova sequenza sia:

START = 1	
11 (Z_3)	1000100111111011 (35323)

Prima del reset, ci si aspetta sui canali di uscita:

DONE = 1			
Z_0	Z_1	Z_2	Z_3
10000110 (134)	01011000 (88)	10000111 (135)	00000000 (0)

A seguire il reset, invece:

DONE = 1			
Z_0	Z_1	Z_2	Z_3
00000000 (0)	00000000 (0)	00000000 (0)	10001110 (142)

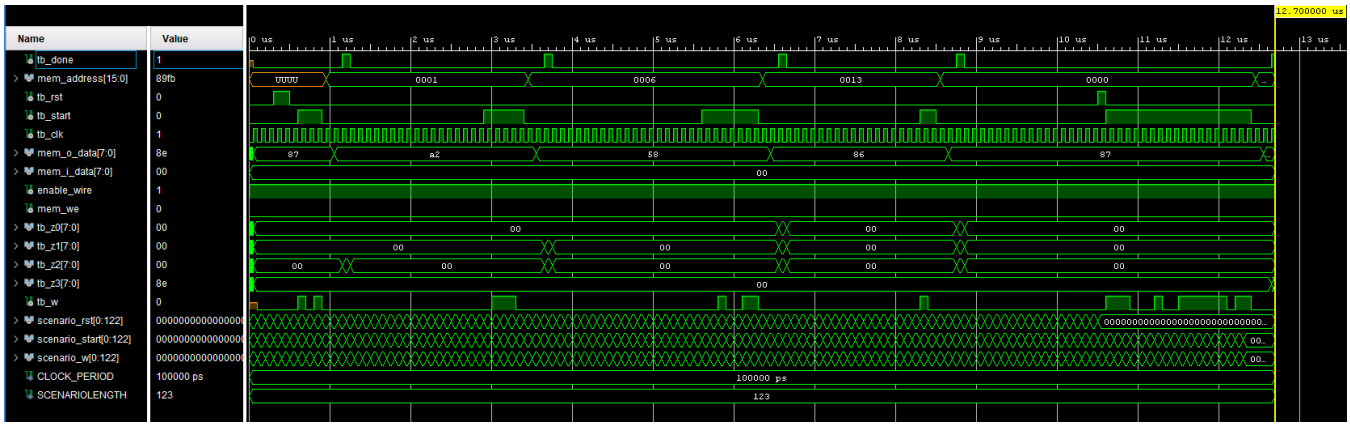
Che corrisponde alla seguente combinazione di segnali:

```
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)     := "00110"
                      & "000" & "00000000000000000000"
                      & "0000000" & "00000000000000000000"
                      & "0000000" & "00000000000000000000"
                      & "00" & "00000000000000000000"
                      & "1" -- reset
                      & "00000000000000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1)   := "00000"
                      & "111" & "00000000000000000000"
                      & "1111100" & "00000000000000000000"
                      & "1111111" & "00000000000000000000"
                      & "11" & "00000000000000000000"
                      & "0"
                      & "11111111111111111";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)       := "00000"
                      & "101" & "00000000000000000000"
                      & "0111000" & "00000000000000000000"
                      & "0010011" & "00000000000000000000"
                      & "10" & "00000000000000000000"
                      & "0"
                      & "11100010011111111011";
```

E con memoria istanziata:

```
TYPE ram_type IS ARRAY (65535 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : ram_type := (  0 => STD_LOGIC_VECTOR(to_unsigned(135, 8)),
                            1 => STD_LOGIC_VECTOR(to_unsigned(162, 8)),
                            2 => STD_LOGIC_VECTOR(to_unsigned(75, 8)),
                            3 => STD_LOGIC_VECTOR(to_unsigned(175, 8)),
                            6 => STD_LOGIC_VECTOR(to_unsigned(88, 8)),
                            19 => STD_LOGIC_VECTOR(to_unsigned(134, 8)),
                            35323 => STD_LOGIC_VECTOR(to_unsigned(142, 8)), --
                            ↪ posto 142 in memoria all'indirizzo pieno
                            OTHERS => "00000000"-- (OTHERS => '0')
                         );
```

L'esecuzione porta dunque al risultato atteso:



3.2.4 Test bench n. 4: combinazioni e reset multipli

Quest'ultimo test bench (come i precedenti) è cumulativo di tutti i test fino ad ora svolti e aggiunge un doppio reset, a valle del quale avviene un'ultimo tentativo di lettura in memoria.
Si tenta, infine, di leggere da un indirizzo di memoria presso il quale è posto il dato convenzionale di default (0).
Per primo, si sceglie di utilizzare come indirizzo in memoria `1111101000` (1000), presso cui ci si aspetta di trovare il valore di default 0.
Il canale di uscita scelto è `01` (Z_1).
Terminata tale lettura, si sceglie di utilizzare come indirizzo in memoria `10` (2), che è rimasto fino ad ora inutilizzato e a cui è posto il valore 75.
Il canale di uscita è nuovamente Z_1, presso cui si aspetta ora di trovare 75.
Infine, viene effettuato un altro reset, a valle del quale si sceglie di utilizzare come indirizzo in memoria `11` (3), che è rimasto fino ad ora inutilizzato e a cui è posto il valore 175.
Il canale di uscita scelto per quest'ultima lettura è `10` (Z_2).
Ci si aspetta gli altri canali espongano ora valore nullo.
Quindi, ci si aspetta che la nuova sequenza sia:

START = 1	
01 (Z_1)	1111101000 (1000)

Seguito da:

START = 1	
01 (Z_1)	10 (2)

A seguire l'ultimo reset:

START = 1	
10 (Z_2)	11 (3)

Sulle uscite, inizialmente:

DONE = 1			
Z_0	Z_1	Z_2	Z_3
00000000 (0)	00000000 (0)	00000000 (0)	10001110 (142)

Si noti che in Z_1 ci si aspetta ancora di trovare 0, perché anche in memoria si trova il valore 0.
Successivamente:

DONE = 1			
Z_0	Z_1	Z_2	Z_3
00000000 (0)	01001011 (75)	00000000 (0)	10001110 (142)

In seguito all'ultimo reset, infine:

DONE = 1			
Z_0	Z_1	Z_2	Z_3
00000000 (0)	00000000 (0)	10101111 (175)	00000000 (0)

Che corrisponde alla seguente combinazione di segnali:

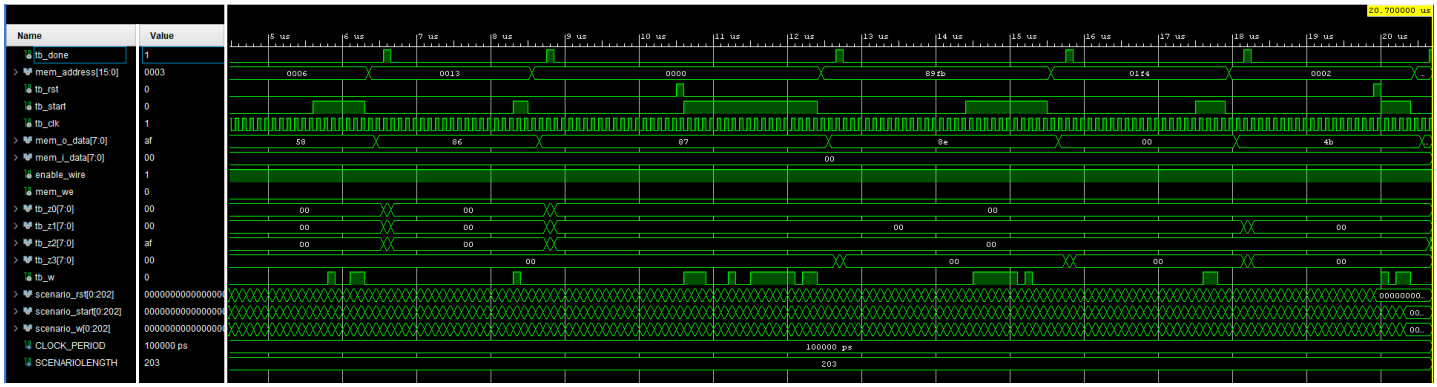
```
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)      := "00110"
                      & "000" & "0000000000000000000"
                      & "0000000" & "0000000000000000000"
                      & "0000000" & "0000000000000000000"
                      & "00" & "0000000000000000000"
                      & "1" -- reset
                      & "0000000000000000" & "0000000000000000000"
                      & "00000000000" & "0000000000000000000"
                      & "0000" & "0000000000000000000"
                      & "1" -- ulteriore reset
                      & "0000";
```

```
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1)    := "00000"
                        & "111" & "00000000000000000000"
                        & "1111100" & "00000000000000000000"
                        & "1111111" & "00000000000000000000"
                        & "11" & "00000000000000000000"
                        & "0"
                        & "1111111111111111" & "00000000000000000000"
                        & "11111111111" & "00000000000000000000"
                        & "1111" & "00000000000000000000"
                        & "0"
                        & "1111";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)        := "00000"
                        & "101" & "00000000000000000000"
                        & "0111000" & "00000000000000000000"
                        & "0010011" & "00000000000000000000"
                        & "10" & "00000000000000000000"
                        & "0"
                        & "11100010011111111011" & "00000000000000000000"
                        & "01111110100" & "00000000000000000000"
                        & "0110" & "00000000000000000000"
                        & "0"
                        & "1011";
```

E con memoria istanziata:

```
TYPE ram_type IS ARRAY (65535 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : ram_type := (  0 => STD_LOGIC_VECTOR(to_unsigned(135, 8)),
                            1 => STD_LOGIC_VECTOR(to_unsigned(162, 8)),
                            2 => STD_LOGIC_VECTOR(to_unsigned(75, 8)), -- valore
                            ↪  utilizzato per la lettura
                            3 => STD_LOGIC_VECTOR(to_unsigned(175, 8)), -- valore
                            ↪  utilizzato per la lettura
                            6 => STD_LOGIC_VECTOR(to_unsigned(88, 8)),
                            19 => STD_LOGIC_VECTOR(to_unsigned(134, 8)),
                            35323 => STD_LOGIC_VECTOR(to_unsigned(142, 8)),
                            OTHERS => "00000000"-- (OTHERS => '0')
                         );
```

L'esecuzione porta dunque al risultato atteso:



4 Conclusioni

I test bench utilizzati durante lo sviluppo e le fasi di testing, anche post-sintesi, pongono le basi sulle tecniche esposte nel capitolo precedente.

Scopo della Prova Finale è dunque stata la gestione del progetto completo di un componente effettivamente implementabile su una FPGA reale Xilinx, grazie a Vivado stesso.
Oltre alle fasi di comprensione ed esemplificazione della specifica si sono dunque utilizzati i paradigmi approfonditi a lezione per realizzare concretamente l'implementazione di una macchina a stati finiti (che trova infatti applicazione in un numero sconfinato di applicazioni professionali).
Si è inoltre svolta una fase approfondita di testing, naturalmente derivante dalle indicazioni della specifica e dei docenti, al fine di consolidare la comprensione e l'idoneità dell'implementazione nei confronti della specifica.