Progetto di Reti Logiche AA 2019-2020

Studenti: Luca Redaelli, Matteo Riboldi
Matricole: 888179, 888305
Codici Persona: 10570738, 10570198

Introduzione

La seguente documentazione vuole descrivere in maniera dettagliata il processo volto alla risoluzione del problema proposto in forma di Prova Finale per l'Anno Accademico 2019-2020 dell'insegnamento di Reti Logiche.
Viene inizialmente presentata la specifica del progetto, completa di indicazioni dei Docenti e arricchita con esempi pratici per poter comprendere al meglio la richiesta.
Segue una prima analisi sull'implementazione scelta (illustrata con grafici per favorirne la lettura). Vengono infine analizzati i dati ottenuti dall'esecuzione di alcuni banchi di test – sia quelli forniti dai Docenti, sia alcuni preparati appositamente per verificare il corretto funzionamento di casi specifici.

Specifica

La specifica della Prova finale (Progetto di Reti Logiche) 2019 è ispirata al metodo di codifica a bassa dissipazione di potenza denominato "Working Zone".
Il metodo di codifica Working Zone è un metodo pensato per il Bus Indirizzi che si usa per trasformare il valore di un indirizzo quando questo viene trasmesso, se appartiene a certi intervalli (detti appunto working-zone). Una working-zone è definita come un intervallo di indirizzi di dimensione fissa (Dwz) che parte da un indirizzo base. All'interno dello schema di codifica possono esistere multiple working-zone (Nwz).
Lo schema modificato di codifica da implementare è il seguente:

- se l'indirizzo da trasmettere (ADDR) non appartiene a nessuna Working Zone, esso viene trasmesso così come è, e un bit addizionale rispetto ai bit di indirizzamento (WZ_BIT) viene messo a 0. In pratica dato ADDR, verrà trasmesso WZ_BIT=0 concatenato ad ADDR (WZ_BIT & ADDR, dove & è il simbolo di concatenazione);
- se l'indirizzo da trasmettere (ADDR) appartiene ad una Working Zone, il bit addizionale WZ_BIT è posto a 1, mentre i bit di indirizzo vengono divisi in 2 sotto campi rappresentanti:
    - Il numero della working-zone al quale l'indirizzo appartiene WZ_NUM, che sarà codificato in binario
    - L'offset rispetto all'indirizzo di base della working zone WZ_OFFSET, codificato come one-hot (cioè il valore da rappresentare è equivalente all'unico bit a 1 della codifica).

  In pratica dato ADDR, verrà trasmesso WZ_BIT=1 concatenato ad WZ_NUM e WZ_OFFSET (WZ_BIT & WZ_NUM & WZ_OFFSET, dove & è il simbolo di concatenazione)

Nella versione da implementare il numero di bit da considerare per l'indirizzo da codificare è 7. Il che definisce come indirizzi validi quelli da 0 a 127. Il numero di working-zone è 8 (Nwz=8) mentre la dimensione della working-zone è 4 indirizzi incluso quello base (Dwz=4). Questo comporta che l'indirizzo codificato sarà composto da 8 bit: 1 bit per WZ_BIT + 7 bit per ADDR, oppure 1 bit per WZ_BIT, 3 bit per codificare in binario a quale tra le 8 working zone l'indirizzo appartiene, e 4 bit per codificare one hot il valore dell'offset di ADDR rispetto all'indirizzo base.

Il modulo da implementare leggerà l'indirizzo da codificare e gli 8 indirizzi base delle working-zone e dovrà produrre l'indirizzo opportunamente codificato.

Dati

I dati ciascuno di dimensione 8 bit sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0. Anche l'indirizzo che è da specifica di 7 bit viene memorizzato su 8 bit. Il valore dell'ottavo bit sarà sempre zero.

- Le posizioni in memoria da 0 a 7 sono usati per memorizzare gli otto indirizzi base delle working-zone:
    - 0 - Indirizzo Base WZ 0
    - 1 - Indirizzo Base WZ 1
    - ...
    - 7 - Indirizzo Base WZ 7
- La posizione in memoria 8 avrà al suo interno il valore (indirizzo) da codificare (ADDR);
- La posizione in memoria 9 è quella che deve essere usata per scrivere, alla fine, il valore codificato secondo le regole precedenti.

Note ulteriori sulla specifica

1. Nella codifica 1 hot si consideri il bit 0 come il meno significativo. In pratica:
    - WZ_OFFSET = 0 è codificato one hot come 0001;
    - WZ_OFFSET = 1 è codificato one hot come 0010;
    - WZ_OFFSET = 2 è codificato one hot come 0100;
    - WZ_OFFSET = 3 è codificato one hot come 1000;
2. Riprendendo ancora la specifica, il valore codificato sarà così composto:
    - Bit 7: valore del singolo bit di WZ_BIT;
    - Bit 6-4: valore codificato binario di WZ_NUM;
    - Bit 3-0: valore codificato one-hot di WZ_OFFSET
3. Se serve, si consideri che gli indirizzi base delle working-zone non cambieranno mai all'interno della stessa esecuzione;
4. Il modulo partirà nella elaborazione quando un segnale START in ingresso verrà portato a 1. Il segnale di START rimarrà alto fino a che il segnale di DONE non verrà portato alto; Al termine della computazione (e una volta scritto il risultato in memoria), il modulo da progettare deve alzare (portare a 1) il segnale DONE che notifica la fine dell'elaborazione. Il segnale DONE deve rimanere alto fino a che il segnale di START non è riportato a 0. Un nuovo segnale start non può essere dato fin tanto che DONE non è stato riportato a zero. Se a questo punto viene rialzato il segnale di START, il modulo dovrà ripartire con la fase di codifica.

ESEMPIO:
La seguente sequenza di numeri mostra un esempio del contenuto della memoria al termine di una elaborazione. I valori che qui sono rappresentati in decimale, sono memorizzati in memoria con l'equivalente codifica binaria su 8 bit senza segno.

CASO 1 CON VALORE NON PRESENTE IN NESSUNA WORKING-ZONE

Indirizzo Memoria	Valore	Commento
0	4	// Indirizzo Base WZ 0
1	13	// Indirizzo Base WZ 1
2	22	// Indirizzo Base WZ 2
3	31	// Indirizzo Base WZ 3
4	37	// Indirizzo Base WZ 4
5	45	// Indirizzo Base WZ 5
6	77	// Indirizzo Base WZ 6
7	91	// Indirizzo Base WZ 7
8	42	// ADDR da codificare
9	42	// Valore codificato con in OUTPUT

CASO 2 CON VALORE PRESENTE IN UNA WORKING-ZONE

Indirizzo Memoria	Valore	Commento
0	4	// Indirizzo Base WZ 0
1	13	// Indirizzo Base WZ 1
2	22	// Indirizzo Base WZ 2
3	31	// Indirizzo Base WZ 3
4	37	// Indirizzo Base WZ 4
5	45	// Indirizzo Base WZ 5
6	77	// Indirizzo Base WZ 6
7	91	// Indirizzo Base WZ 7
8	33	// ADDR da codificare
9	180	// Valore codificato con in OUTPUT (1 - 011 - 0100)

Il componente da descrivere deve avere la seguente interfaccia.

```
entity 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 project_reti_logiche;
```

In particolare:

- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_start è il segnale di START generato dal Test Bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_data è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_address è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- o_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.

APPENDICE: Descrizione Memoria
NOTA: **La memoria è già istanziata all'interno del Test Bench e non va sintetizzata**

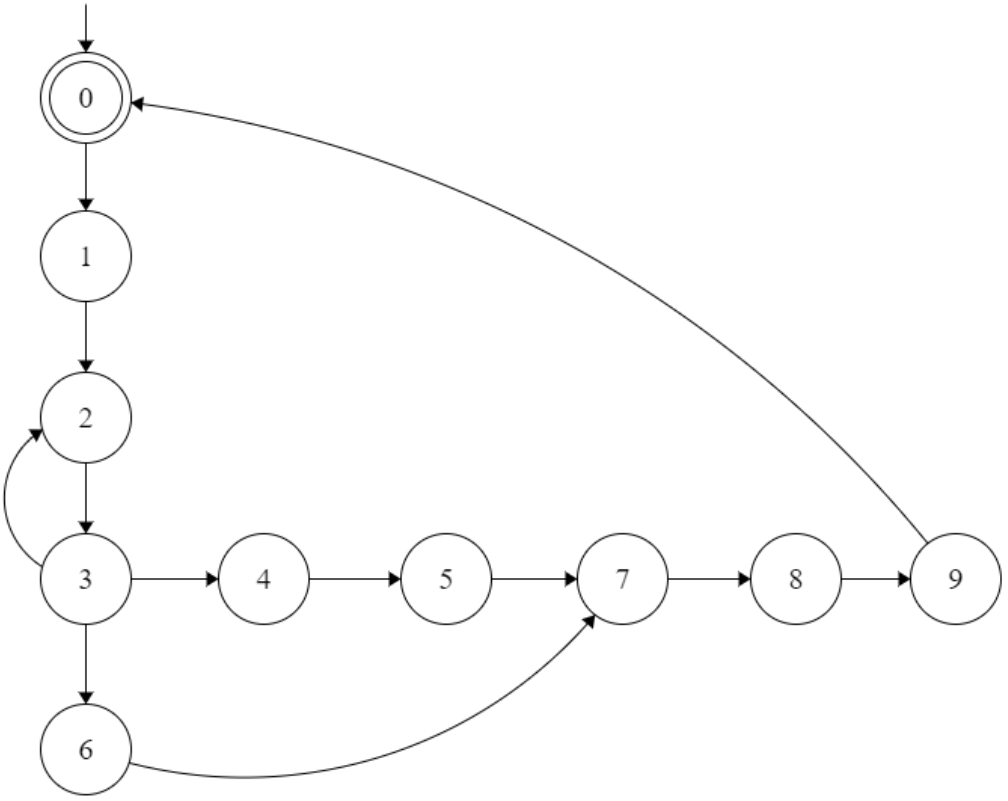
La memoria e il suo protocollo può essere estratto dalla seguente descrizione VHDL che fa parte del test bench e che è derivata dalla User guide di VIVADO disponibile al seguente link:
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado-synthesis.pdf

```
-- Single-Port Block RAM Write-First Mode (recommended template)
--
-- File: rams_02.vhd
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rams_sp_wf is
port(
  clk  : in  std_logic;
  we   : in  std_logic;
  en   : in  std_logic;
  addr : in  std_logic_vector(15 downto 0);
  di   : in  std_logic_vector(7 downto 0);
  do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;

architecture syn of rams_sp_wf is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM : ram_type;
begin
  process(clk)
    begin
    if clk'event and clk = '1' then
      if en = '1' then
        if we = '1' then
          RAM(conv_integer(addr)) <= di;
          do                      <= di;
        else
          do <= RAM(conv_integer(addr));
        end if;
      end if;
    end if;
  end process;
end syn;
```

Analisi della specifica

Dopo un'attenta analisi e lettura della specifica risulta ragionevole semplificare il processo di codifica dell'indirizzo tramite la realizzazione di una macchina a stati finiti.
Viene qui proposta una rappresentazione grafica della macchina. Sono stati utilizzati dieci stati, ai quali è necessario aggiungerne un undicesimo per consentire la corretta gestione della memoria (ricezione e scrittura dei dati).



La seguente tabella vuole presentare al lettore l'associazione stato-nome e una breve descrizione dello stesso.

Stato	Nome	Descrizione
0	START	Stato iniziale della macchina.
1	ADDR_LOADER	Carica l'indirizzo da analizzare.
2	WZ_BASE_LOADER	Carica l'indirizzo base della ennesima Working Zone.
3	DIFFERENCE_CALC	Calcola la differenza tra ADDR e WZ__BASE, valutando se l'indirizzo appartenga (o meno) alla Working Zone.
4	ONE_SHOT_CALC	Calcola gli ultimi 4 bit dell'indirizzo finale in formato one shot.
5	ADDR_WZ	Produce l'indirizzo nel caso in cui ADDR appartenga ad una Working Zone.
6	ADDR_NWZ	Produce l'indirizzo nel caso in cui ADDR non appartenga ad alcuna Working Zone.
7	ADDR_WRITER	Scrive in memoria l'indirizzo prodotto.
8	DONE	Aggiorna il segnale o_done a 1.
9	DONE_WAIT	Attende il segnale i_start a 0, quindi resetta il segnale o_done a 0 e riporta la macchina nel suo stato iniziale.

Tale approccio consente di codificare correttamente l'indirizzo richiesto sia nel caso in cui questo non appartenga ad alcuna Working Zone sia nel caso in cui, invece, vi appartenga.
Risulta importante analizzare alcuni passaggi chiave della codifica:

1. Calcolo della differenza tra ADDR e WZ_BASE
2. Calcolo della codifica one shot per gli ultimi 4 bit dell'indirizzo finale
3. Produzione dell'indirizzo finale

Partiamo dal presupposto che, durante la codifica, verranno utilizzate cinque variabili:

- ADDR: intero atto a contenere il valore da analizzare
- WZ_BASE: intero atto a contenere gli indirizzi base delle Working Zone
- ONE_SHOT: vettore atto a contenere la codifica one shot dell'indirizzo
- counter: contatore (per tenere traccia del numero di cicli eseguiti)
- difference: intero (differenza tra ADDR e WZ_BASE)

La prima operazione viene elaborata nello stato 3 della macchina presentata in precedenza, DIFFERENCE_CALC. Nel caso in cui la differenza algebrica tra ADDR e WZ_BASE (precedentemente caricati da memoria e salvati in opportune variabili) sia pari a 0, 1, 2 o 3 – ADDR necessita codifica Working Zone, appartenendo ad una di esse – la macchina viene indirizzata correttamente verso lo stato 4, ONE_SHOT_CALC, per poter proseguire con la trasformazione dell'indirizzo. Il valore della differenza viene opportunamente salvato in una variabile denominata difference.
Se invece la differenza tra ADDR e WZ_BASE non dovesse rientrare nei parametri per la codifica in Working Zone la macchina verrebbe indirizzata:

- allo stato 2, WZ_BASE_LOADER, nel caso in cui la variabile counter non dovesse ancora aver raggiunto il valore di 7 (sono 7 i valori di base Working Zone da verificare), per poter raccogliere un ulteriore indirizzo base
- allo stato 6, ADDR_NWZ, nel caso in cui la variabile counter dovesse aver raggiunto il valore di 7 – tutte le Working Zone sono state analizzate e ADDR non risulta appartenere ad alcuna di esse, pertanto non necessita ulteriore codifica

La seconda operazione viene invece effettuata nello stato 4 della macchina, ONE_SHOT_CALC. Va ricordato che tale stato risulta raggiungibile se e solo se ADDR appartiene a una Working Zone. A seconda del valore differenza calcolato in precedenza, la macchina attribuisce alla variabile ONE_SHOT uno specifico valore:

- "0001" nel caso in cui difference sia uguale a 0
- "0010" nel caso in cui difference sia uguale a 1
- "0100" nel caso in cui difference sia uguale a 2
- "1000" nel caso in cui difference sia uguale a 3

L'ultima operazione, invece, viene prodotta su due stati differenti, a seconda del tipo di indirizzo da generare.

- Indirizzo non in Working Zone: stato 6, ADDR_NWZ. In tal caso il valore di ADDR viene convertito in un vettore di 7 bit preceduto con concatenazione da un bit posto a 0, come da specifica.
- Indirizzo in Working Zone: stato 5, ADDR_WZ. In tal caso ad un bit posto opportunamente ad 1, come da specifica, vengono concatenati, nell'ordine, il valore della variabile counter (che determina il numero della Working Zone di appartenenza, da 0 a 7, convertito in un vettore di 3 bit) e il valore della variabile ONE_SHOT.

Durante la modellazione della macchina a stati sono emerse alcune criticità. Andremo ora ad analizzarle e a spiegare in che modo esse siano state correttamente risolte.

1. Gestione del segnale `i_rst`
2. Gestione della memoria

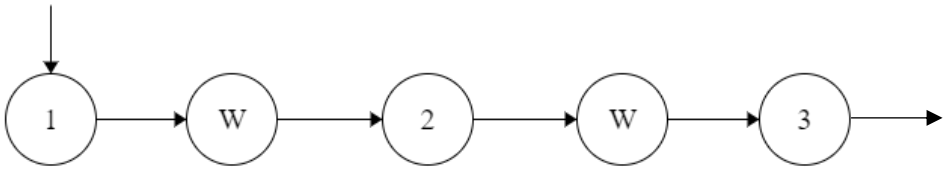
`i_rst`, come da specifica, è il segnale che permette di inizializzare la macchina pronta per ricevere il suo primo segnale `i_start`. Pertanto, ogniqualvolta il segnale `i_rst` viene portato ad 1, la macchina deve essere correttamente riportata nello stato di `START` e le sue variabili devono essere inizializzate. Il problema è pertanto risolto nel seguente modo:

```
if (i_rst = '1') then
    o_en <= '0';            Riporta il segnale di reading da memoria allo stato zero.
    o_we <= '0';            Riporta il segnale di writing su memoria allo stato zero.
    o_done <= '0';          Riporta il segnale di completamento allo stato zero.
    ADDR := 0;              Resetta l'indirizzo da analizzare letto.
    WZ_BASE := 0;           Resetta l'indirizzo base della Working Zone letto.
    ONE_SHOT := "0000";     Resetta l'indirizzo base della codifica one shot.
    counter := 0;           Resetta il contatore di cicli.
    difference := 0;        Resetta il valore della differenza.
    CURR_STATE <= START;    Riporta la macchina allo stato iniziale.
    STATE <= START;         Riporta la macchina allo stato iniziale.
end if;
```

Il secondo problema, invece, è stato risolto tramite l'aggiunta di uno stato intermedio `WAIT_MEM` interposto tra ogni coppia di stati che faccia uso della memoria esterna (per leggere o scrivere). Tale stato consente infatti alla memoria di inviare/ricevere correttamente i dati, tramite un delay artificiale.

```
when WAIT_MEM =>
     if (CURRENT_STATE = START) then
          STATE <= ADDR_LOADER;
     elsif (CURRENT_STATE = ADDR_LOADER) then
          STATE <= WZ_BASE_LOADER;
     elsif (CURRENT_STATE = ADDR_WZ) then
          STATE <= ADDR_WRITER;
     elsif (CURRENT_STATE = ADDR_NWZ) then
          STATE <= ADDR_WRITER;
     elsif (CURRENT_STATE = ADDR_WRITER) then
          STATE <= DONE;
     else
          STATE <= DONE_WAIT;
     end if;
```

Viene quindi mostrato un esempio grafico del processo generato:



Testbenching

Per verificare il corretto funzionamento del componente vengono utilizzate delle testbenches. Alcune di queste sono state fornite dai Docenti, altre sono state opportunamente progettate per verificare funzionalità specifiche del componente stesso, per stressarlo e per garantirne la validità.

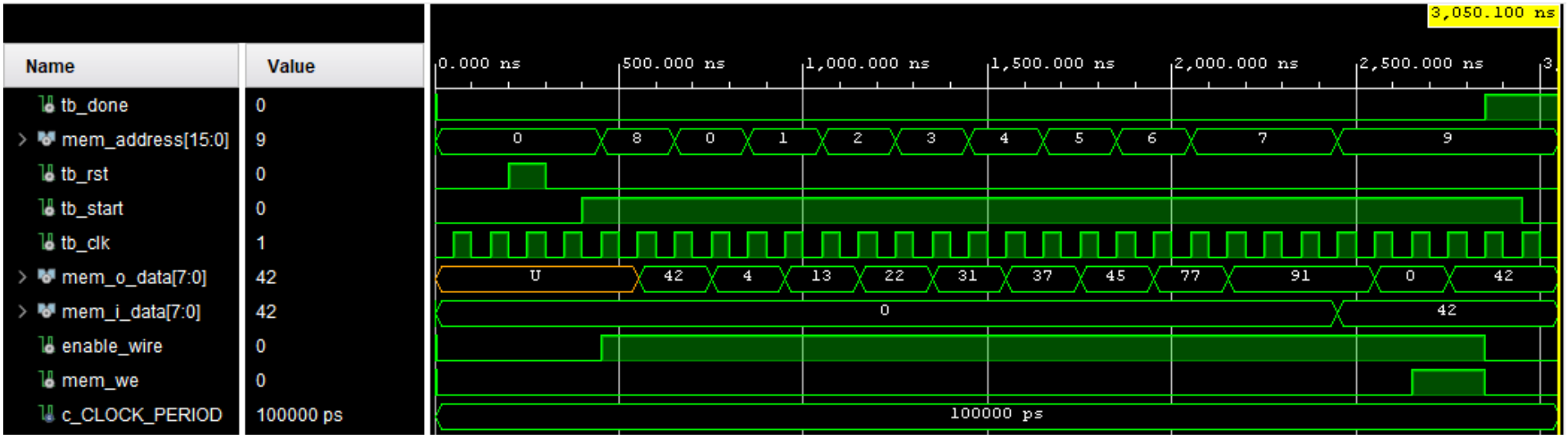
Prima di tutto sono effettuate delle simulazioni post-sintesi: di queste risulta particolarmente interessante l'analisi dei diagrammi Waveform per accertarsi che tutti i segnali inviati e ricevuti siano corretti. Vengono quindi effettuate anche le simulazioni behavioural, dalle quali si riconosce l'andamento delle variabili nel corso del tempo.

Ogni testbench prodotta è una variazione delle testbench ufficiali fornite dai Docenti.
Di seguito sono riportati i nomi ed una breve descrizione delle testbench utilizzate:

`TESTBENCH UFFICIALE 1`	Verifica il caso in cui ADDR non appartenga ad alcuna Working Zone. Accompagnata da analisi
`TESTBENCH UFFICIALE 2`	Verifica il caso in cui ADDR appartenga ad una Working Zone. Accompagnata da analisi
`RESET`	Verifica il corretto funzionamento del segnale di reset
`INVERSED`	Verifica il corretto funzionamento del componente nel caso in cui gli indirizzi base delle Working Zone siano disposti al contrario
`RANDOM`	Verifica il corretto funzionamento del componente nel caso in cui gli indirizzi basi delle Working Zone siano disposti in ordine casuale. Accompagnata da analisi
`BEFORE FIRST`	Verifica il corretto funzionamento del componente nel caso in cui ADDR preceda il primo valore di WZ_BASE
`AFTER LAST`	Verifica il corretto funzionamento del componente nel caso in cui ADDR sia successivo all'ultimo valore di WZ_BASE
`DIFFERENCE 0`	Verifica il corretto funzionamento del componente nel caso in cui la differenza tra ADDR e WZ_BASE sia pari a 0
`DIFFERENCE 1`	Verifica il corretto funzionamento del componente nel caso in cui la differenza tra ADDR e WZ_BASE sia pari a 1
`DIFFERENCE 2`	Verifica il corretto funzionamento del componente nel caso in cui la differenza tra ADDR e WZ_BASE sia pari a 2
`DIFFERENCE 3`	Verifica il corretto funzionamento del componente nel caso in cui la differenza tra ADDR e WZ_BASE sia pari a 3

I report statement per le simulazioni in modalità behavioural sono stati resi sottoforma di commento nella versione ufficiale del progetto. Per poterli visualizzare correttamente in console devono essere necessariamente decommentati.
Il progetto e le relative testbench sono disponibili al seguente indirizzo (https://github.com/lucaredaelli/Progetto-di-Reti-Logiche) e saranno rese pubbliche a partire dal 15 settembre 2020 onde evitare plagi di alcun tipo.

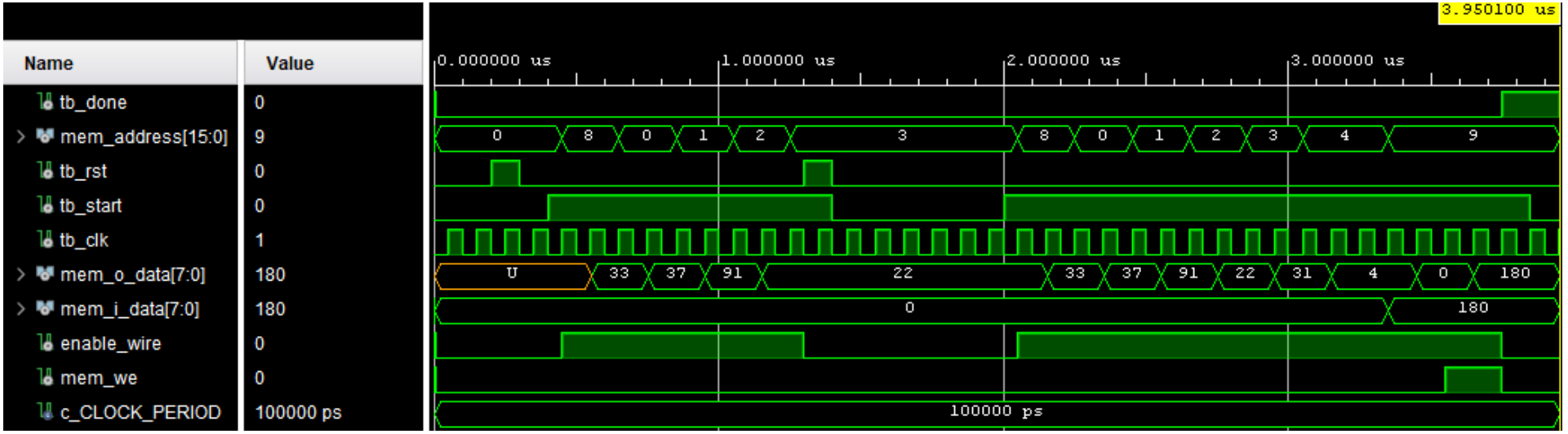
Testbench Ufficiale 1 – NO WZ



Si noti come il componente, dopo aver letto il valore di ADDR, cicli su tutti i valori di WZ_BASE. Non essendo il valore 42 appartenente ad alcuna Working Zone scrive in memoria, all'indirizzo 9, il valore 42 preceduto da uno 0.
Il processo ha richiesto in tutto 31 cicli di clock.

Note della modalità behavioural
Note: Reset!
Note: ADDR: 42
Note: WZ_BASE in analisi: 4
Note: WZ_BASE in analisi: 13
Note: WZ_BASE in analisi: 22
Note: WZ_BASE in analisi: 31
Note: WZ_BASE in analisi: 37
Note: WZ_BASE in analisi: 45
Note: WZ_BASE in analisi: 77
Note: WZ_BASE in analisi: 91
Note: ADDR non appartiene ad alcuna Working Zone.
Note: Processo completato.
Failure: Simulation Ended!, TEST PASSATO

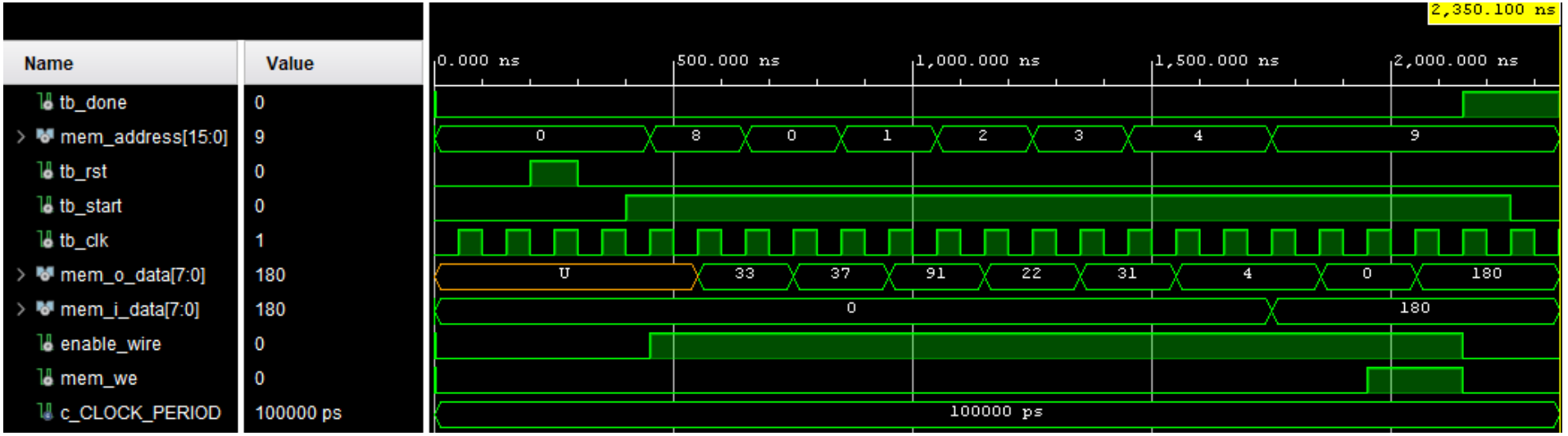
Testbench Ufficiale 2 – IN WZ



Si noti come il componente, dopo aver letto il valore di ADDR, cicli su tutti i valori di WZ_BASE (considerando eventuali reset richiesti dal testbench) e non appena verifica che ADDR appartenga alla Working Zone 3 ne calcola il valore codificato corretto – in questo caso 180 – andando a scriverlo in memoria.
Il processo ha richiesto in tutto 40 cicli di clock.

Note della modalità behavioural
Note: Reset!
Note: ADDR: 33
Note: WZ_BASE in analisi: 4
Note: WZ_BASE in analisi: 13
Note: Reset!
Note: ADDR: 33
Note: WZ_BASE in analisi: 4
Note: WZ_BASE in analisi: 13
Note: WZ_BASE in analisi: 22
Note: WZ_BASE in analisi: 31
Note: ADDR appartiene alla seguente Working Zone: 3
Note: Processo completato.
Failure: Simulation Ended!, TEST PASSATO

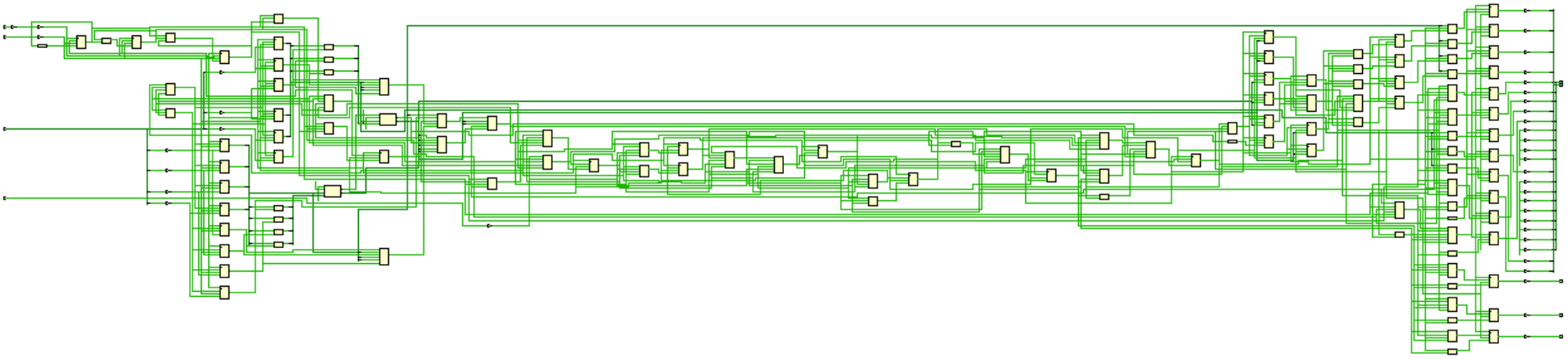
Random



Rispetto al test precedente, in questo sono stati rimossi dei reset superflui e i valori in memoria di WZ_BASE sono stati resi casuali.
Si noti come il componente, dopo aver letto il valore di ADDR, cicli su tutti i valori di WZ_BASE e non appena verifica che ADDR appartenga alla Working Zone 3 ne calcola il valore codificato corretto – in questo caso 180 – andando a scriverlo in memoria.
Il processo ha richiesto in tutto 24 cicli di clock.

Note della modalità behavioural
Note: Reset!
Note: ADDR: 33
Note: WZ_BASE in analisi: 37
Note: WZ_BASE in analisi: 91
Note: WZ_BASE in analisi: 22
Note: WZ_BASE in analisi: 31
Note: ADDR appartiene alla seguente Working Zone: 3
Note: Processo completato.
Failure: Simulation Ended!, TEST PASSATO

Schematic Implementation



What's Inside?