Progetto di Reti Logiche (Prova Finale)

Studente: Francesco Romanò (Codice persona 10619531 - Matricola 888514)

Anno accademico 2019/2020

Prof. Gianluca Palermo

Indice

1. Introduzione

Lo scopo del progetto è la realizzazione di un componente hardware descritto tramite linguaggio VHDL. Il componente deve codificare un indirizzo qualsiasi appartenente ad una RAM che comprende otto aree di memoria non sovrapposte fra loro dette *Working Zone* (WZ). La codifica da effettuare è ispirata alla codifica *Working Zone* ed è descritta più dettagliatamente di seguito.

Ogni indirizzo ha 8 bit (compreso l'indirizzo codificato) e la RAM ha esattamente 127 indirizzi (MSB sempre a 0).
In particolare se l'indirizzo rientra in uno di quelli appartenenti a una WZ (ogni WZ è formata da 4 indirizzi), esso sarà codificato nel modo:

WZ_BIT (1 bit)	WZ_NUM (3 bit)	WZ_OFFSET (4 bit)

dove WZ_BIT è uguale a '1' per segnalare che l'indirizzo è in una WZ, WZ_NUM è il numero della WZ in codifica binaria e WZ_OFFSET è la distanza dall'indirizzo base (ossia l'indirizzo piú piccolo dei quattro) della WZ in codifica "one hot".

Se invece l'indirizzo da codificare non fa parte di nessuna WZ sarà codificato nel modo:

WZ_BIT (1 bit)	ADDR (7 bit)

dove WZ_BIT è uguale a '0' per segnalare che l'indirizzo non è in una WZ e ADDR rappresenta l'indirizzo da codificare (che è sempre lungo 7 bit).

I dati necessari a tale operazione sono salvati su una memoria che contiene rispettivamente tutti gli indirizzi base delle 8 WZ (indirizzi da 0 a 7) e l'indirizzo di cui si vuole fare la codifica (indirizzo 8), la quale viene scritta nella cella successiva (indirizzo 9).

0	Indirizzo base WZ_0
1	Indirizzo base WZ_1
2	Indirizzo base WZ_2
3	Indirizzo base WZ_3
4	Indirizzo base WZ_4
5	Indirizzo base WZ_5
6	Indirizzo base WZ_6
7	Indirizzo base WZ_7
8	Indirizzo da codificare
9	Indirizzo codificato

Figura 1.1: Rappresentazione della memoria su cui lavora il componente

Il componente dopo aver estratto tutti gli indirizzi base è pronto per: estrarre l'indirizzo da codificare, effettuare la traduzione, scrivere in memoria l'indirizzo codificato e eventualmente continuare in ripetute codifiche di altri indirizzi.

L'interfaccia del componente è, da specifica, la seguente:

```
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.

Figura 1.2: Dettagli tratti dalla specifica fornita

2. Architettura del componente progettato

Il segnale di ingresso i_rst prepara il componente a una nuova sessione di codifica, portando la FSM nello stato START nel quale vengono configurati i parametri interni nella configurazione iniziale. La

macchina rimane in tale stato fino a quando il segnale di ingresso i_start non viene portato a ‘1’, tale evento fa partire l’esecuzione. A esecuzione terminata il componente manda il segnale o_done a ‘1' e attende che i_start torni alto per eseguire una nuova codifica.

2.1 Stati della macchina progettata

La macchina, riportata in Figura 2.1, è composta da 16 stati, dei quale segue una descrizione sintetica:

START
Stato iniziale nel quale viene portata la FSM in seguito a un segnale alto di i_rst, qui ci si prepara a leggere l’indirizzo base della prima WZ e si abilita la lettura (o_en viene portato a ‘1').
Quando i_start viene portato a ‘1’, la FSM passa al prossimo stato.

LOAD_WZ0
Stato in cui il componente legge e salva come signal interno l’indirizzo base di WZ_0 e prepara la lettura di WZ_1.

Seguono gli stati del tutto analoghi utilizzati per leggere e memorizzare tutti gli indirizzi base: LOAD_WZ1, LOAD_WZ2, LOAD_WZ3, LOAD_WZ4, LOAD_WZ5, LOAD_WZ6, LOAD_WZ7.

LOAD_ADDR
Stato in cui viene letto l’indirizzo che si vuole codificare, chiamato ADDR_TARGET d’ora in poi.

CHECK_WZ
Stato in cui è concentrata la logica del componente, tramite dei confronti con gli indirizzi salvati precedentemente si prepara la codifica:
- Se ADDR_TARGET non appartiene a nessuna WZ lo stato prossimo sarà DONE_STATE.
- Se ADDR_TARGET appartiene a una WZ vengono impostati i valori di WZ_NUM e WZ_OFFSET come da specifica e lo stato prossimo sarà WZ_DONE_STATE. In particolare WZ_BIT sarà = ‘1’, WZ_NUM sarà il numero della WZ in binario (da "000" a "111") e WZ_OFFSET sarà la distanza dall’indirizzo base della WZ in questione in codifica “one-hot” (da "0001" a "1000").

DONE_STATE
Stato in cui viene preparata la scrittura in memoria degli indirizzi con codifica 0 & addr, che corrisponde a ADDR_TARGET (che ha sempre MSB a ‘0'). Il segnale o_we viene alzato per abilitare la scrittura.

WZ_DONE_STATE
Stato in cui viene preparata la scrittura in memoria degli indirizzi con codifica 1 & WZ_NUM & WZ_OFFSET. Il segnale o_we viene alzato per abilitare la scrittura.

TERMINATE_CONVERSION
Stato dove o_we viene abbassato e viene alzato o_done per segnalare la fine della codifica, inoltre il componente viene messo in “ascolto" sull’indirizzo di memoria dove è salvato ADDR_TARGET per rileggerlo in quanto questo potrebbe essere cambiato prima di una nuova esecuzione.

DONE
Stato in cui o_done viene riabbassato.

NEW_START
Stato di fine codifica dove il componente rimane fin quando non riceve un nuovo segnale alto di i_start. Quando ciò accade viene letto il nuovo ADDR_TARGET e il componente è pronto per una nuova codifica. Questo stato non coincide con START in quanto non viene rieffettuata la lettura degli indirizzi base delle WZ, che sono rimasti salvati alla prima codifica effettuata. Invece un eventuale segnale alto di i_rst riporta il componente nel primo stato START.

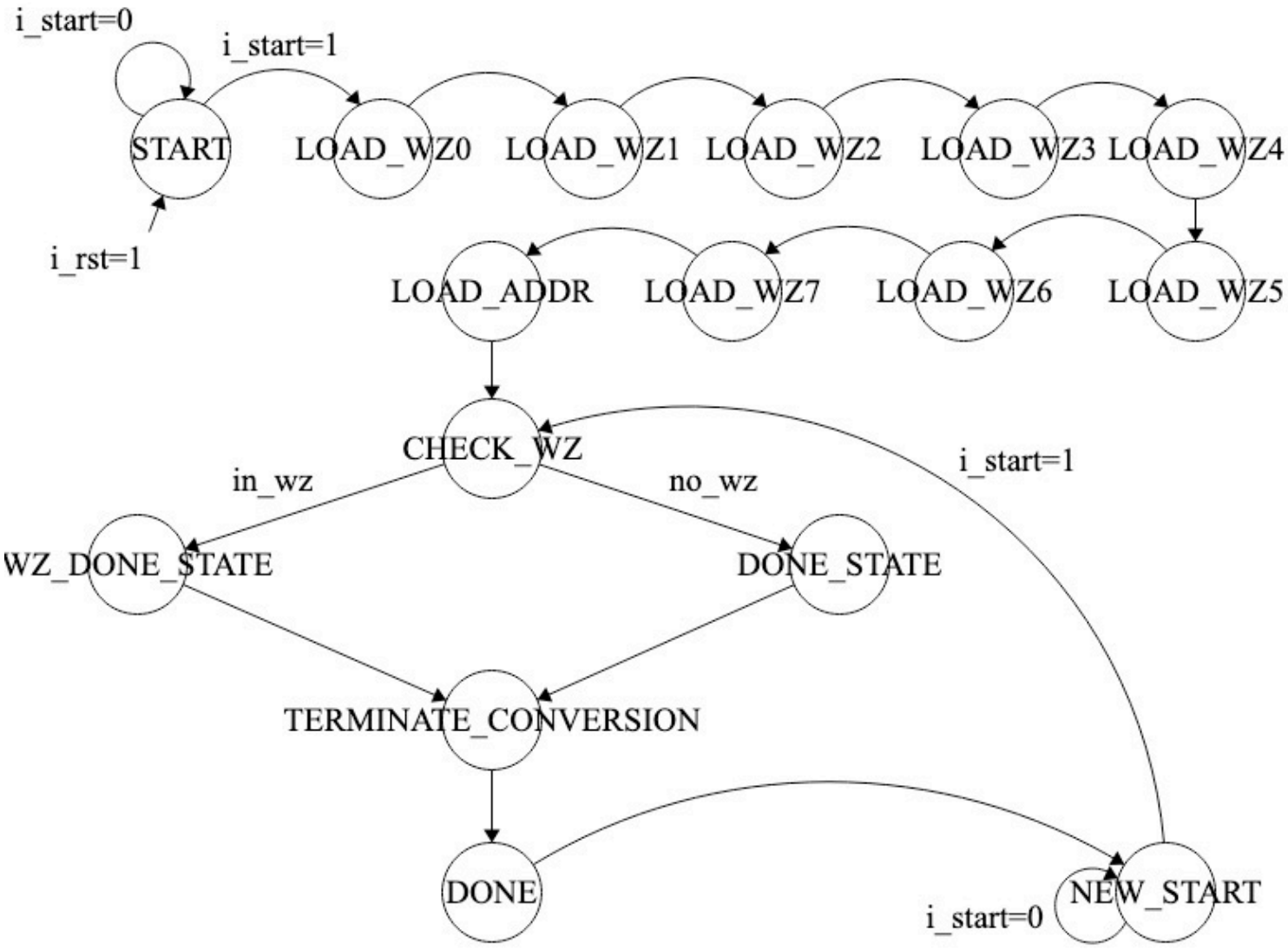

Figura 2.1: Diagramma degli stati della FSM progettata

2.2 Scelte progettuali

La principale scelta progettuale è stata quella di far salvare al componente tutti gli indirizzi base delle otto WZ. La scelta è stata presa ipotizzando numerose codifiche di diversi indirizzi all'interno di una stessa esecuzione: infatti questo approccio permette, una volta caricati e salvati tutti gli indirizzi base, di effettuare codifiche successive in modo molto rapido passando dallo stato NEW_START direttamente allo stato CHECK_WZ in seguito ad un segnale alto di i_start.

Un'altra scelta è stata quella di concentrare la logica di codifica in un unico stato in modo da effettuare la codifica vera e propria in un unico ciclo di clock. Si è dunque cercato di prediligere l'efficienza temporale su più codifiche come linea guida progettuale.

3. Risultati sperimentali

Una volta descritto il componente tramite codice VHDL è stata effettuata la sua sintesi e si è poi passati alla fase di testing. Di entrambe queste fasi segue un sintetico resoconto.

3.1 Report di sintesi

Il componente viene correttamente sintetizzato sul dispositivo consigliato nelle specifiche.
In particolare il report di sintesi riporta la seguente codifica degli stati:

State	New Encoding	Previous Encoding
start	0000000000000001	0000
load_wz0	0000000000000010	0001
load_wz1	0000000000000100	0010
load_wz2	0000000000001000	0011
load_wz3	0000000000010000	0100
load_wz4	0000000000100000	0101
load_wz5	0000000001000000	0110
load_wz6	0000000010000000	0111
load_wz7	0000000100000000	1000
load_addr	0000001000000000	1001
check_wz	0000010000000000	1010
wz_done_state	0000100000000000	1100
done_state	0001000000000000	1011
terminate_conversion	0010000000000000	1101
done	0100000000000000	1110
new_start	1000000000000000	1111

Inoltre vengono fornite le seguenti informazioni sui componenti base sintetizzati:

```
---------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input      8 Bit       Adders := 24
+---Registers :
                   16 Bit    Registers := 1
                    8 Bit    Registers := 10
                    4 Bit    Registers := 1
                    3 Bit    Registers := 1
                    1 Bit    Registers := 4
+---Muxes :
      16 Input     16 Bit         Muxes := 1
      50 Input     16 Bit         Muxes := 1
      16 Input      8 Bit         Muxes := 1
      32 Input      4 Bit         Muxes := 1
      16 Input      4 Bit         Muxes := 1
      32 Input      3 Bit         Muxes := 1
      16 Input      3 Bit         Muxes := 1
      32 Input      1 Bit         Muxes := 2
      16 Input      1 Bit         Muxes := 9
---------------------------------------------------------------
```

Infine il componente è risultato anche correttamente implementabile:

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
✓ synth_1	constrs_1	synth_design Complete!								290	110	0.0	0	0	2/24/20, 4:37 PM	00:00:32
✓ impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	0.455	0	286	110	0.0	0	0	2/24/20, 4:38 PM	00:01:40

Il numero piuttosto elevato di LUT e FF è giustificato dalla linea guida scelta di favorire e ottimizzare la velocità di esecuzioni multiple.

3.2 Simulazioni e test

Il componente sintetizzato supera correttamente entrambi i test pubblici forniti. Per assicurarsi del corretto funzionamento sono inoltre state testate 6 classi test che ne valutano casi notevoli di funzionamento e vengono riportate di seguito.

- **Test a bordo RAM**: 4 casi di test che valutano la corretta codifica di ADDR_TARGET quando questo coincide con i bordi della RAM ossia ADDR_TARGET = “00000000" e ADDR_TARGET = “01111111”, sia quando questi appartengono ad una WZ sia quando non appartengono a nessuna WZ.

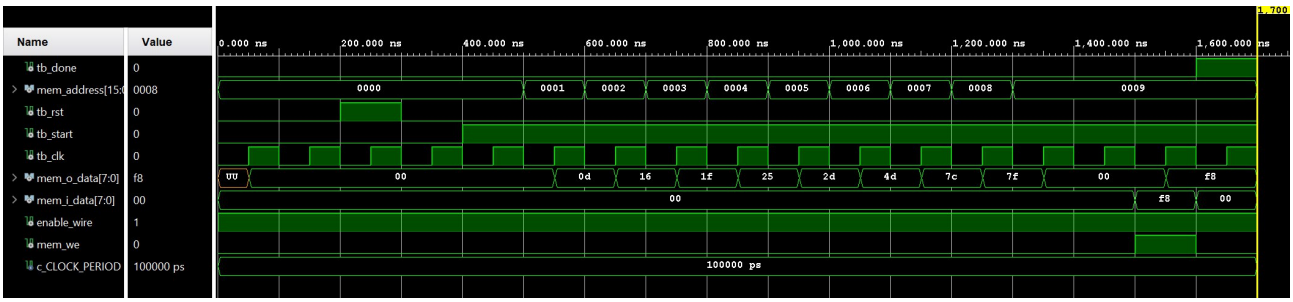

Figura 3.1: Simulazione Post-Syntesis del bordo RAM inferiore dentro la WZ con indirizzo base 124

- **Test a bordo WZ**: test che valuta la corretta codifica quando ADDR_TARGET coincide con il bordo superiore o con il bordo inferiore di una WZ.

- **Test WZ adiacenti**: test che valuta la corretta codifica quando ADDR_TARGET coincide con un indirizzo a cavallo tra due WZ adiacenti tra loro.

- **Test con più segnali di i_start**: test che valuta il comportamento in presenza di più segnali alti i_start e con cambiamento del valore di ADDR_TARGET.

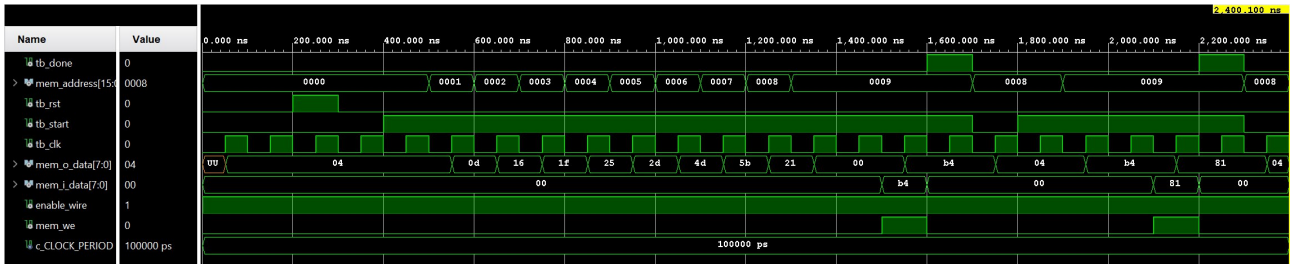

Figura 3.2: Simulazione Post-Syntesis con due segnali i_start e cambio di ADDR_TARGET

- **Test con più segnali di i_rst**: test che valuta il comportamento in presenza di più segnali alti i_rst, che forzano la rilettura delle WZ.

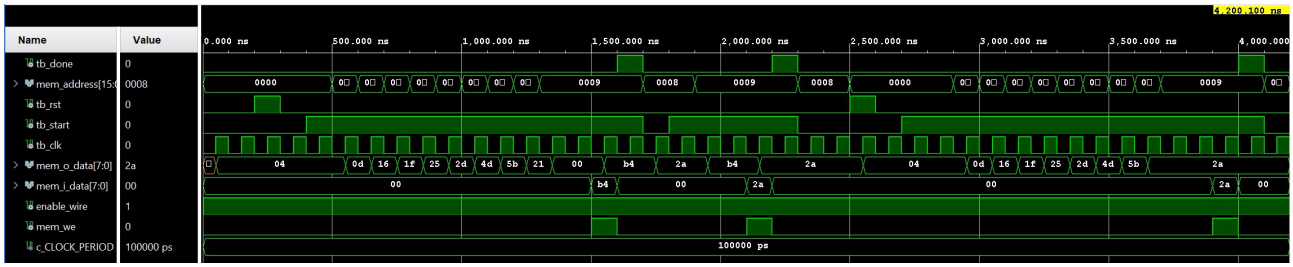

Figura 3.3: Simulazione Post-Syntesis con tre segnali di start e due segnali di reset

- **Test generici**: il componente è stato inoltre testato con numerosi casi non notevoli per accertare il comportamento voluto su una grossa casistica.

Tutti i test sono stati eseguiti sia in simulazione *Behavioural* sia in simulazione *Post-Syntesis Functional* e in entrambi i casi non sono stati riscontrati errori di funzionamento.

4. Conclusioni

Ricapitolando quanto esposto sopra il componente sviluppato risulta:

- Progettato prediligendo l'efficienza su numerose esecuzioni consecutiva.
- Progettato e testato con la frequenza di clock data nella specifica e pari a 10 MHz (periodo di 100ns).
- Testato anche a frequenza di clock più alte, passa tutti i test pubblici e numerosi test randomici con periodo di clock <= 20ns
- Correttamente sintetizzabile sul dispositivo consigliato nelle specifiche, ossia FPGA xc7a200tfbg484-1.
- Testato e funzionante in pre e post-sintesi.
- Correttamente implementabile senza errori.