Progetto di reti logiche

Prof. Gianluca Palermo - Anno 2019/2020

Fabio Stecchi (numero matricola: 889223, codice persona: 10573273)

Manuel Tsironas (numero matricola: 889717, codice persona: 10581775)

Indice

1. Introduzione

1.1 Scopo del progetto

Si definisca una working zone come un intervallo di indirizzi di dimensione fissa che parte da un indirizzo base: lo scopo del progetto è di realizzare un componente HW in VHDL che, preso in ingresso un indirizzo, ne restituisca una codifica in base alla sua appartenenza o meno ad una working zone.

1.2 Specifiche generali

Nel caso in cui l'indirizzo non appartenga ad una working zone esso viene trasmesso uguale con un bit di indirizzamento a 0 davanti.

In caso contrario l'indirizzo viene codificato come segue:

- Il primo bit di indirizzamento a 1.
- I bit dal secondo al quarto indicano a quale delle 8 working zone l'indirizzo appartiene (in binario).
- I bit dal quinto all'ottavo indicano l'offset dell'indirizzo dall'indirizzo di base della working zone di appartenenza (in codifica one_hot).

Nel nostro caso le working zone sono 8 e ognuna di esse è costituita da 4 indirizzi.

1.3 Dati e descrizione della memoria

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al byte:

- Gli indirizzi tra 0 e 7 contengono gli indirizzi base delle working zone.
- L'indirizzo 8 contiene l'indirizzo da codificare.
- L'indirizzo 9 deve essere usato per scrivere l'indirizzo codificato.

Figura 1: rappresentazione indirizzi significativi della memoria

2. Architettura

2.1 Interfaccia del componente

Il componente da realizzare 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 TestBench;
- `i_rst` è il segnale di `RESET` che inizializza la macchina pronta per ricevere il primo segnale `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.

2.2 Scelta progettuale

Abbiamo scelto di implementare il componente con una FSM sfruttando due processi:

- Il primo sensibile al segnale di clock e a quello di reset che si occupa dell'avanzamento degli stati della FSM e di riportare la macchina nello stato iniziale in caso di reset.
- Il secondo definisce cosa succede in ogni singolo stato della FSM.

2.3 Stati della macchina

La FSM è costituita da 17 stati, i cui principali sono:

- s0: stato in cui si attende che il segnale di start passi a 1.
- read_code_state: si legge da memoria l'indirizzo da codificare.
- address_wz_state: si assegna all'o_address l'indirizzo di base della prima working zone.
- check_wz_number_state: attraverso un counter che tiene conto di quante wz sono state lette (e confrontate con l'indirizzo da codificare) si decide se continuare a leggere da memoria l'indirizzo della prossima wz (se il counter è minore di 7) o se codifcare l'indirizzo in quanto non appartiene a nessuna wz (se il counter è uguale a 7).
- read_wz_state: legge l'indirizzo della wz corrente.
- check_address_in_wz_state, check_address_in_wz_state_2,

`check_address_in_wz_state_3`, `check_address_in_wz_state_4`: controlla se l'indirizzo base, l'indirizzo base + 1, l'indirizzo base + 2 o l'indirizzo base +3 sono uguali all'indirizzo da codificare. In caso affermativo codifica l'indirizzo in quanto appartenente alla wz, in caso negativo continua a ciclare.

- `final_address_in_wz_state`: codifica l'indirizzo nel caso in cui quest' ultimo appartenga ad una working zone come descritto dalla specifica
- `final_addres_no_wz_state`: codifica l'indirizzo nel caso in cui quest'ultimo non appartenga a nessuna working zone (lasciandolo inalterato ma con un 0 davanti).
- `done_state`: setta `o_done` a 1.
- `final_state`: aspetta che `o_start` ritorni a 0 e poi torna allo stato iniziale.

2.4 Minimalità della FSM

La FSM realizzata non rappresenta la soluzione a numero di stati minimo in quanto è possibile ridurre ulteriormente il diagramma per ottenere una soluzione più compatta. Tuttavia abbiamo ritenuto che fosse la scelta migliore in termini di chiarezza e operabilità del codice.

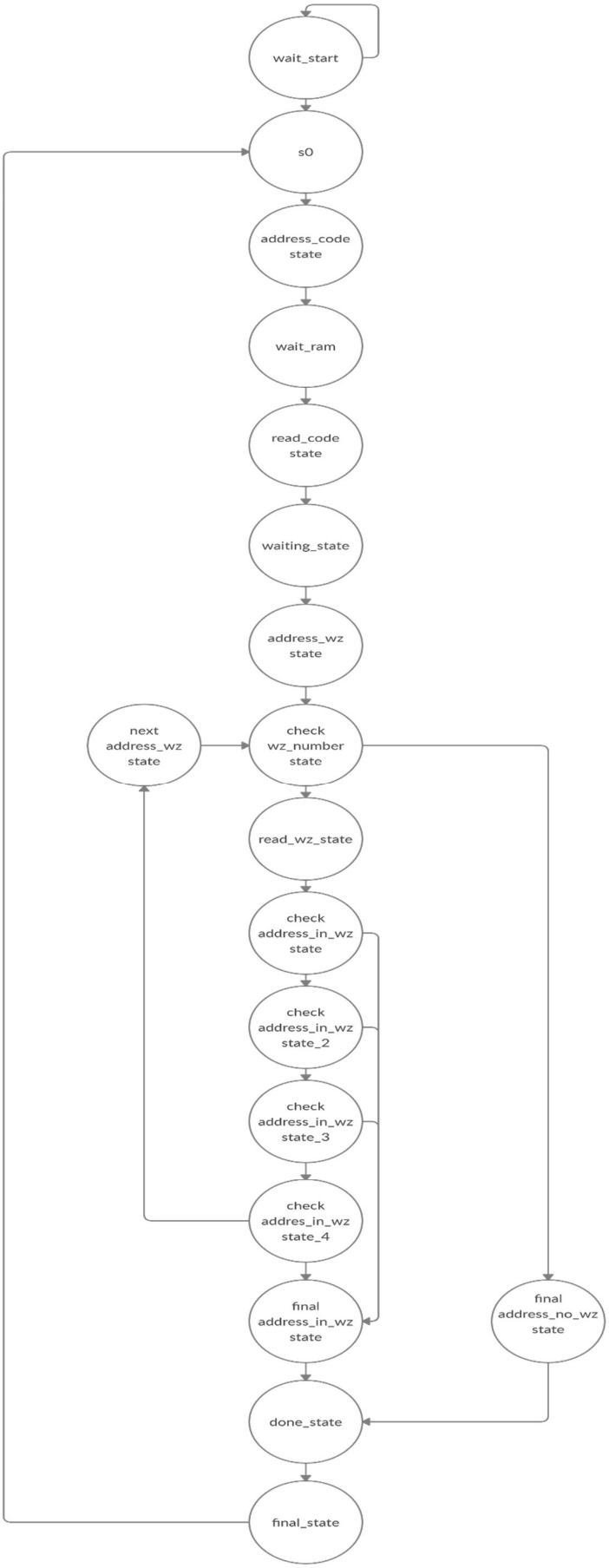

Figura 2: macchina a stati

3.Risultati dei test

Per verificare il corretto funzionamento del nostro componente l'abbiamo sottoposto a diverse simulazioni. Abbiamo fatto diversi test bench tra cui uno che generava casualmente i dati presenti in memoria coprendo così tutti i casi possibili. In seguito illustreremo solamente i test che riteniamo più significativi:

- **Test fornito dal docente** (indirizzo presente in una wz):

  indirizzo da codificare: 33

  valore codificate: 180 (10110100 – b4 HEX)

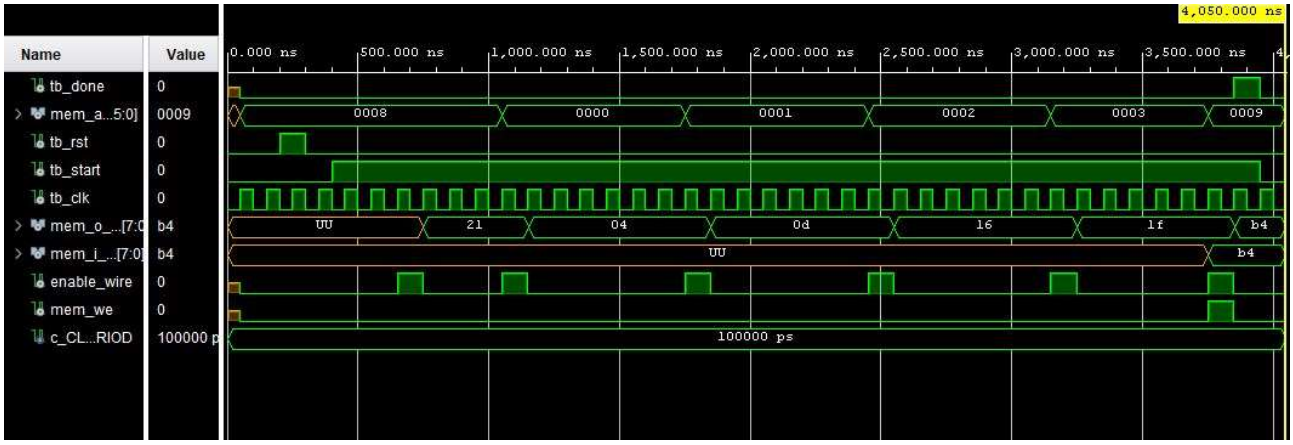

Figura 3: wave simulazione indirizzo presente in una wz

- **Test fornito dal docente** (indirizzo non presente in nessuna wz):

  indirizzo da codificare: 42

  valore codificato: 42 (00101010 – 2a HEX)

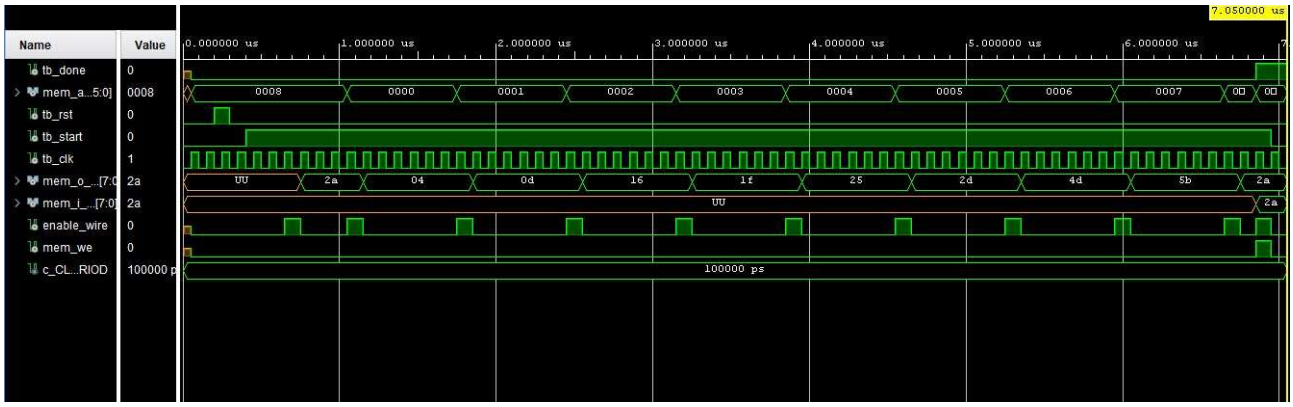

Figura 4: wave simulazione indirizzo non presente in nessuna wz

Abbiamo poi sottoposto il componente a dei test che verificassero il corretto funzionamento dei segnali:

- **Multi start**: il test verifica la corretta sincronizzazione dei segnali `i_start, i_rst, o_done` andando ad eseguire due simulazioni di fila.

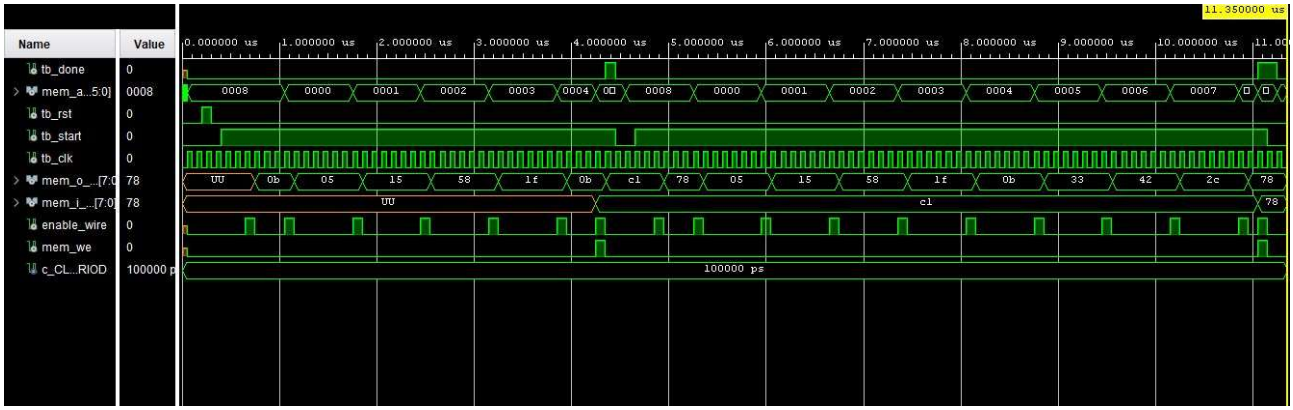

Figura 5: wave multi start

- **Reset asincrono**: il test verifica che il segnale di reset non comprometta la computazione e che riinizi facendo tornare la macchina nello stato iniziale.

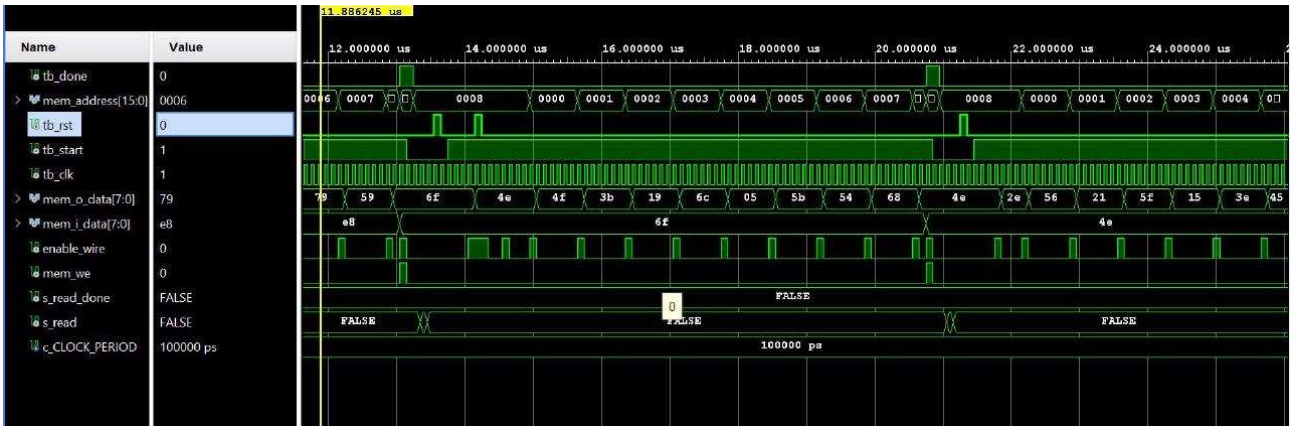

Figura 6: wave reset asincrono

4.Conclusioni

Il componente sintetizzato supera correttamente tutti i test a cui lo abbiamo sottoposto nelle 3 simulazioni: `Behavioral, Post-Synthesis Functional` e `Post-Synthesis Timing`.

Ulteriori informazioni quali il numero di LUT e FF e lo schematic, ricavabili dalla sintesi del componente, potranno essere ricavate dalle seguenti immagini:

Name	Constraints	Status	...	...	...	...	...	...	...	LUT	FF
✓ synth_1	constrs_1	synth_design Complete!								76	73
▷ impl_1	constrs_1	Not started									

Figura 7 e 8: synthesis information