Relazione prova finale Reti Logiche

A.A 2019-2020

Panzeri Matteo Matricola:890586

Indice

1 Specifica di progetto

1.1 Caratteristiche generali

L'obiettivo del progetto è quello di sintetizzare un componente hardware che permette di codificare un indirizzo secondo il metodo working zone. Questo metodo, usato per l'address bus, trasforma, prima di inviarli, alcuni indirizzi appartenenti ad aree specifiche della memoria dette working zone.
Una working zone è definita come un intervallo di indirizzi di dimensione fissa.
Nella memoria possono essere presenti più working zone, in particolare nella versione da implementare l'indirizzo da codificare ha 7 bit, da cui, gli indirizzi validi sono quelli da 0 a 127.
Sono presenti 8 working zone composte da 4 indirizzi ciascuna.

1.2 Funzionamento del componente

Il modulo comincerà l'elaborazione ricevuto un segnale di start (START=1). Questo verrà mantenuto alto per tutta la durata dell'elaborazione.
Terminata l'elaborazione del segnale in ingresso, cioè scritto in memoria il risultato, il segnale di termine verrà portato alto (DONE=1).
Il valore di questo segnale verrà mantenuto finché il segnale d'attivazione (START) non risulterà basso (START=0), solo in seguito il segnale DONE potrà essere abbassato.
Se il modulo riceverà un altro segnale di START=1 allora dovrà ricominciare l'elaborazione.

1.3 Elaborazione dell'indirizzo

Una volta ricevuto l'indirizzo in ingresso (ADDR) il componente si comporterà in 2 modi distinti.

Se l'indirizzo non appartiene a nessuna working zone l'indirizzo trasmesso sarà l'indirizzo ricevuto concatenato ad un bit 0, questo è detto WZ_BIT, cioè trasmetterò come risultato WZ_BIT&ADDR.

Se l'indirizzo appartiene alla working zone verrà trasmesso un segnale così composto:
WZ_BIT&WZ_NUM&WZ_OFFSET .
WZ_BIT sarà un solo bit 1.
WZ_NUM sarà composto da 3 bit e rappresenta in quale delle 8 working zone è l'indirizzo in ingresso.
WZ_OFFSET sarà composto da 4 bit e rappresenta quale dei 4 indirizzi componenti la working zone è il segnale in ingresso, questo viene codificato one hot.
Secondo la codifica one hot se lo sfasamento dell'indirizzo da elaborare rispetto alla base della WZ è 0 (è la base) WZ_OFFSET=0001,se lo sfasamento è 1 WZ_OFFSET=0010, se lo sfasamento è 2 WZ_OFFSET=0100,
infine se è 3 WZ_OFFSET=1000.
Si può notare come qualunque sia il risultato esso sarà un segnale di 8 bit.

1.4 I dati

I dati sono raccolti in una memoria (già fornita) con indirizzamento al Byte e parte dall' indirizzo 0.
Ogni indirizzo è composto da 8 bit.
Anche se l'indirizzo da elaborare è composto da 7 bit esso è salvato su 8(l'ottavo sarà sempre 0).
Gli spazi di memoria con indirizzo da 0 a 7 contengono l'indirizzo base

della working zone. Quello con indirizzo 8 conterrà il segnale da elaborare mentre quello di indirizzo 9 sarà l'indirizzo su cui scrivere il risultato dell'elaborazione del segnale d'ingresso.

2 Progetto

2.1 Interfaccia del componente

riporto l'interfaccia del componente data nella specifica ufficiale.

```
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;

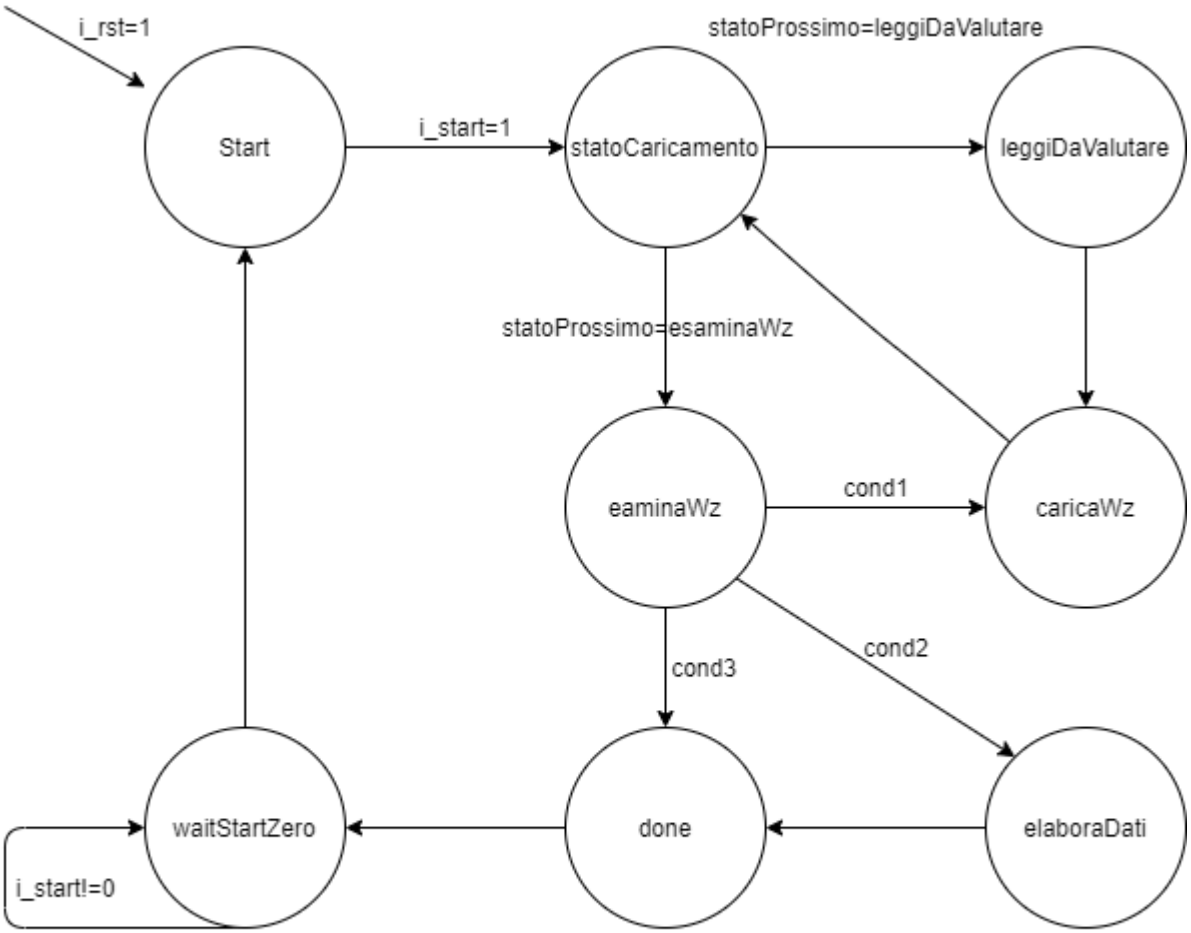
• o_address è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;

• 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.

2.2 Macchina a stati finiti



cond1:la working zone esaminata non contiene l'indirizzo da elaborare e ci sono altre Wz da esaminare
cond2:ho trovato la working zone che contiene l'indirizzo da elaborare
cond3:non c'è working zone che contiene l'indirizzo da elaborare

2.3 Analisi degli stati

-Start: è lo stato di partenza, in questo stato si inizializzano le variabili usate nell'elaborazione e si imposta l'indirizzo di lettura all'ottava cella di memoria

-Stato caricamento: è uno stato che dal punto di vista della specifica non fa niente se non permettere la lettura dei dati in assenza di questo il passaggio da uno stato all'altro avverrebbe senza aver caricato i dati e quindi si effettuerebbe un' elaborazione non corretta.

-Leggi da valutare: è lo stato in cui leggo il contenuto della cella 8 e lo conservo(in un apposita variabile) fino alla fine dell'elaborazione.

-carica wz :è lo stato che permette la lettura delle celle di memoria (dalla 0 alla 7).
-esamina wz: è lo stato in cui si determina se l'indirizzo estratto dalla cella 8 è contentuto o meno in una working zone. Ci sono 3 possibili casi:
1) l'indirizzo non è contenuto nella working zone appena estratta ma ci sono altre working zone da controllare (cond1 in figura) si torna in caricaWz che si occuperà di caricare la base della working zone successiva.
2) l'indirizzo non è contenuto nella working zone appena estratta ma non ci sono altre working zone da controllare, si provvede ad aggiungere il bit 0 in testa all'indirizzo e caricarlo in memoria e quindi passare allo stato done (cond3 in figura).
3) l'indirizzo è contenuto nella working zone appena estratta e si passa quindi allo stato che elabora il valore da scrivere in RAM(9).

-elaboraDati:stato che elabora l'indirizzo e lo scrive in memoria come da specifica.

-done :stato che si occupa di segnalare il termine dell'elaborazione

-waitStartZero:stato che attende che start venga abbassato per abbassare done e tornare nuovamente a start.

3 Sintesi

Dal documento di sintesi sono state estratte informazioni relativo all'uso di componenti base usati dal sintetizzatore per creare il componente.

Module project_reti_logiche
Detailed RTL Component Info :

+---Adders :
2 Input 32 Bit Adders := 1
2 Input 8 Bit Adders := 1
3 Input 8 Bit Adders := 1
2 Input 3 Bit Adders := 1

+---Registers :
32 Bit Registers := 1
16 Bit Registers := 2
8 Bit Registers := 1
7 Bit Registers := 2
3 Bit Registers := 2
1 Bit Registers := 3

+---Muxes :
8 Input 32 Bit Muxes := 1
8 Input 16 Bit Muxes := 2
8 Input 8 Bit Muxes := 1
2 Input 4 Bit Muxes := 1
4 Input 4 Bit Muxes := 1
10 Input 3 Bit Muxes := 1
8 Input 1 Bit Muxes := 13
10 Input 1 Bit Muxes := 1

4 Testbenches e simulazioni

Per quanto riguarda la simulazione del componente questa è stata fatta sia in pre-sintesi che post-sintesi nello specifico (Functional simulation). Sono stati eseguiti diversi testbench ricavati a partire dai 2 dati in fase di

specifica, in viene riportato uno di questi testbench il quale è il più caratteristico tra quelli eseguiti.

Testbench_resetInMezzoTrovatoInMezzo.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;

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( 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( 90 , 8)),
                         8 => std_logic_vector(to_unsigned( 47 , 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           => mem_address,
        o_done              => tb_done,
        o_en          => enable_wire,
        o_we                => mem_we,
        o_data        => mem_i_data
        );

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;
      --reset in mezzo all'esecuzione
      wait for 3500 ns;
      tb_rst <= '1';
      wait for c_CLOCK_PERIOD;
      tb_rst <= '0';
      tb_start <= '0';
       wait for 500 ns;
```

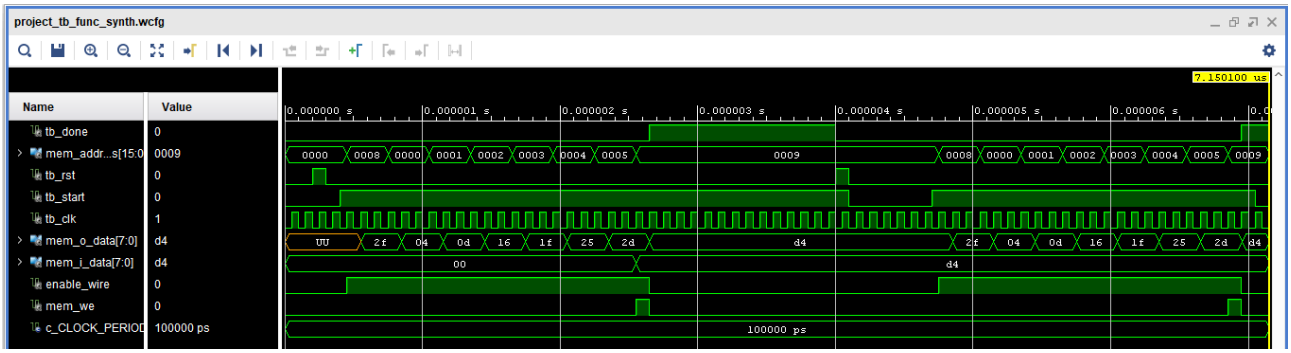
```
        wait for c_CLOCK_PERIOD;
        tb_start <= '1';
        wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;

    tb_start <= '0';
    wait until tb_done = '0';

    -- Maschera di output = 0 - 42
    assert RAM(9) = std_logic_vector(to_unsigned( 212 , 8)) report "TEST FALLITO. Expected  212  found " &
integer'image(to_integer(unsigned(RAM(9))))  severity failure;

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

Il testbench qui riportato ha la caratteristica di inviare il segnale di reset quando il componente trova la working zone a cui appartiene l'indirizzo dato da elaborare.
L'obbiettivo è quello di capire se il componente ricominci l'elaborazione nonostante possa concludere l'elaborazione.
Qui in seguito viene riportato il waveForm generato dalla simulazione in post sintesi del testbench sopra descritto.



5 Conclusioni e possibili miglioramenti

Uno dei possibili miglioramenti del componente è sicuramente quello di eliminare lo stato LeggiDaValutare dalla macchina a stati finiti e unirlo allo stato EsaminaWz in questo modo si semplifica la FSM. Dal punto di vista del tempo d'esecuzione il componente non subirà miglioramenti notevoli visto che, in ogni caso la cella di memoria RAM(8) dovrà essere letta ed eliminando lo stato LeggiDaValutare si cambia lo stato in cui questo viene

fatto.
Per migliorare il tempo d’esecuzione invece si potrebbe unire lo stato ElaboraDati allo stato EsaminaWz.
Bisogna notare però che anche questo permette di recuperare poco tempo, infatti questo stato è attraversato una volta (se l’indirizzo è in una working zone) oppure non viene attraversato e il tempo d’esecuzione del componente sarebbe uguale a quella del componente specificato in questa relazione.