Prova finale

Progetto reti logiche

Prof Gianluca Palermo – Anno 2022/2023

Sebastiano Piantanida (cod. persona: 10658432 matricola: 933994)

Indice:

Introduzione

Scopo del progetto

Lo scopo del progetto è implementare un componente hardware descritto in linguaggio VHDL, in grado di ricevere in input uno stream di bit da cui il componente estrae un indirizzo di memoria e l'indicazione del canale d'uscita su cui indirizzare i dati letti a tale indirizzo.

Specifiche generali

Il componente riceve in ingresso 2 bit rappresentanti l'indirizzo della porta di uscita (out_port), N bit (max 16) che corrispondono all'indirizzo di memoria a cui accedere (temp_addr), il componente trasmetterà l'indirizzo tramite o_mem_addr alla memoria RAM, leggerà i dati dalla ram tramite i_mem_data ed infine mostrerà tali dati sull'uscita corrispondente (o_z0, o_z1, o_z2, o_z3).

Interfaccia del componente

Il componente descritto ha la seguente interfaccia:

```
entity project_reti_logiche is
  Port (
        i_clk : in STD_LOGIC;
        i_rst : in STD_LOGIC;
        i_start : in STD_LOGIC;
        i_w : in STD_LOGIC;
        o_z0 : out STD_LOGIC_VECTOR (7 downto 0);
        o_z1 : out STD_LOGIC_VECTOR (7 downto 0);
        o_z2 : out STD_LOGIC_VECTOR (7 downto 0);
        o_z3 : out STD_LOGIC_VECTOR (7 downto 0);
        o_done : out std_logic;
        o_mem_addr : out std_logic_vector (15 downto 0);
        i_mem_data : in std_logic_vector (7 downto 0);
        o_mem_we : out std_logic ;
        o_mem_en : out std_logic
      );
end project_reti_logiche;
```

Nello specifico:

- i_clk è il segnale di CLOCK in ingresso dal test bench;
- i_rst è il segnale di RESET che inizializza la macchina e la porta in uno stato pronto a ricevere il segnale di START;
- i_start è il segnale di START generato dal test bench;

- i_w segnale (seriale) contenente 2 bit di selezione della porta d'uscita e n bit d'indirizzo di memoria;
- o_zn n-esima porta d'uscita che mostra i dati in uscita (vettore ad 8 bit);
- o_done segnale di uscita che comunica la fine dell'elaborazione e che i dati sono visibili sulle uscite;
- o_mem_addr segnale in uscita verso la memoria su cui viene trasmesso l'indirizzo del dato in memoria;
- i_mem_data segnale in ingresso dalla memoria dove viene letto il dato in memoria richiesto;
- o_mem_we segnale per abilitare la scrittura sulla memoria;
- o_mem_en segnale per abilitare la scrittura/lettura della memoria;

Descrizione segnali interni

```
signal state: state_type;
signal out_port: std_logic_vector(1 downto 0);
signal temp_addr: std_logic_vector(15 downto 0);
signal out_data: std_logic_vector(7 downto 0);
signal temp_z0: std_logic_vector(7 downto 0);
signal temp_z1: std_logic_vector(7 downto 0);
signal temp_z2: std_logic_vector(7 downto 0);
signal temp_z3: std_logic_vector(7 downto 0);
```

Dove:

- state inddica lo stato corrente nel quale si trova il componente
- out_port è un vettore a 2 bit su cui viene salvata la porta d'uscita selezionata
- temp_addr è un vettore a 16 bit che indica l'indirizzo di memoria che si vuole andare a leggere
- out_data è un vettore a 8 bit su cui viene salvato temporaneamente i dati letti dalla memoria
- temp_zn sono i vettori a 8 bit su cui vengono memorizzati valori delle porte di uscita

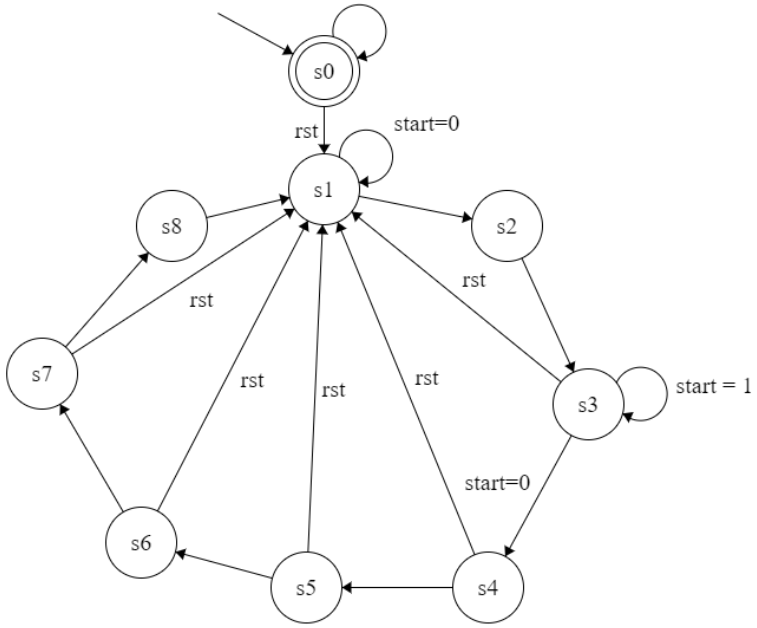
Design

Dopo aver ricevuto il primo segnale di Reset il componente si sposta dallo stato di idle(s0) allo stato s1 nel quale, dopo aver ricevuto il segnale START = 1, comincia l'elaborazione. Il componente legge quindi la porta d'uscita e l'indirizzo del dato cercato finché il segnale SATRT resta alto dopo di che si interfaccia con la memoria per produrre il risultato

Stati della macchina

- **S0:**
  Stato di idle nel quale si trova il sistema prima di ricevere il primo segnale di RESET che inizializzi il componente.
- **S1:**
  Stato iniziale della computazione, il componente attende in questo stato che il segnale i_start venga portato a 1, quando ciò avviene il componente memorizza il bit letto in out_port(1) e si sposta allo stato successivo S2. Ogni volta che viene portato a 1 il segnale i_rst il sistema viene riportato in questo stato.

- **S2:**
  In questo stato il componente legge il secondo bit su i_w che viene memorizzato in out_port(0) prima di passare allo stato successivo S3.
- **S3:**
  Stato in cui la macchina rimane finché il segnale i_start è alto. In questo stato viene letto bit a bit l'indirizzo di memoria dal segnale i_w e viene salvato su temp_addr il quale viene fatto scorrere verso sinistra ad ogni ciclo di clock per permettere di posizionare il nuovo bit letto da i_w in posizione 0.
- **S4:**
  In questo stato viene portato il segnale o_mem_en a 1 per permettere l'accesso alla memoria e viene scritto su o_mem_addr l'indirizzo letto in input e salvato su temp_addr.
- **S5:**
  Stato in cui viene riportato a 0 il segnale o_mem_en e vengono letti gli 8 bit in uscita dalla memoria da i_mem_data e salvati su out_data.
- **S6:**
  Stato in cui i dati salvati in out_data vengono salvati sulla memoria di uscita (temp_z0, temp_z1, temp_z2, temp_z3) corrispondente alla porta salvata in out_port.
- **S7:**
  Viene portato a 1 il segnale o_done per indicare il termine dell'elaborazione e vengono mostrati sulle uscite (o_z0, o_z1, o_z2, o_z3) i dati salvati sulle memorie di uscita (temp_z0, temp_z1, temp_z2, temp_z3)
- **S8:**
  Stato di reset in cui viene riportato a 0 il segnale o_done e vengono reinizializzare i segnali interni necessari all'elaborazione del successivo input. La macchina viene poi riportata allo stato S1 pronta a ricevere il prossimo input.

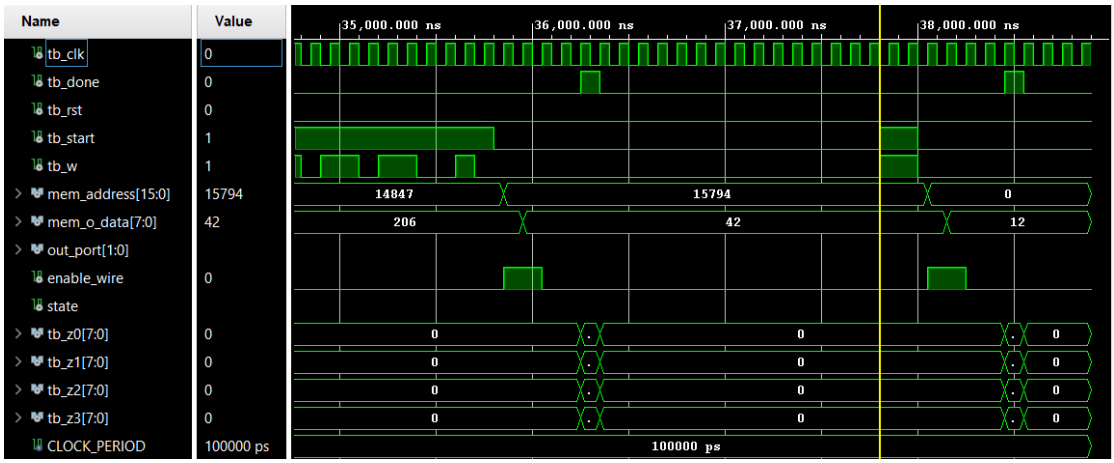


Risultati dei test

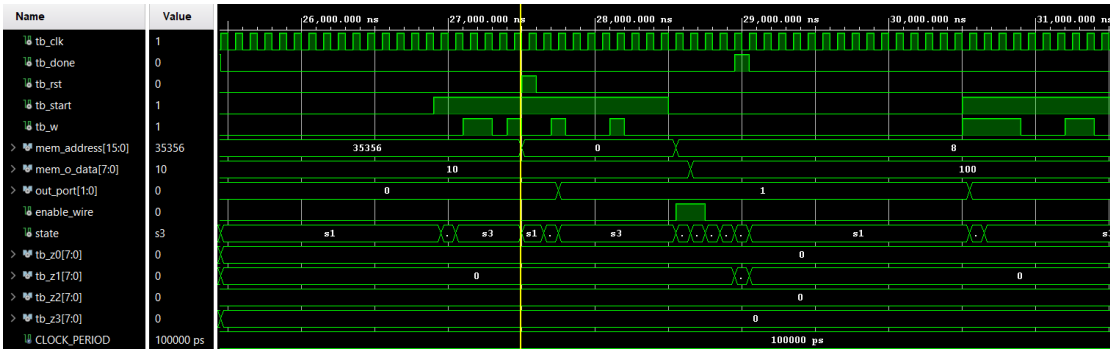
Al fine di verificare il comportamento del componente, dopo averlo testato coi test banch di esempio, ho realizzato altri test banch per ferificarne il corretto funzionamento anche durante i corner case e massimizare la copertura di tutti i possibili cammini dell'automa

I corner case analizzati sono:

1. Il segnale i_start rimane alto solo per due cicli di clock e quindi l'indirizzo di memoria di input è formato da 0 bit, in tal caso il componente utilizza l'indirizzo base 0.

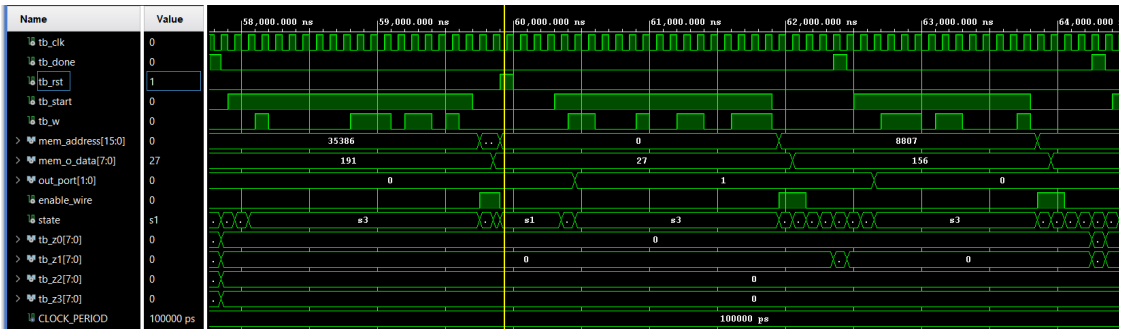


2. Ricezione del segnale di reset nel mezzo della lettura del segnale i_w, in questo caso il sistema torna correttamente allo stato iniziale e riprende la lettura del segnale i_w a partire dal ciclo di clock successivo a quello in cui ha ricevuto il segnale di reset.

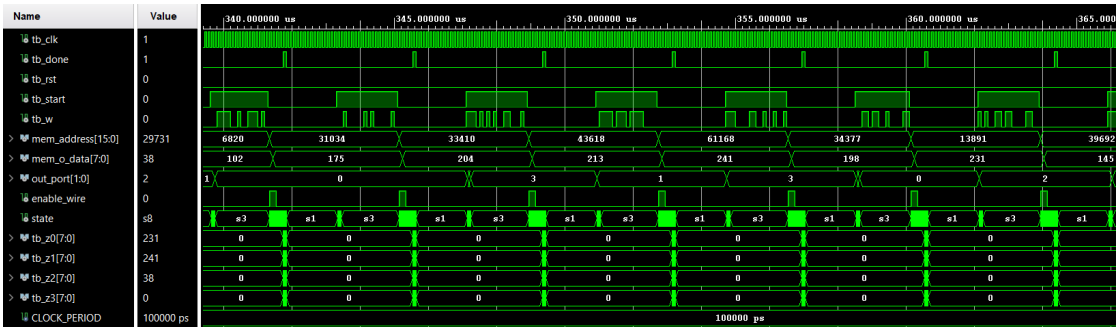


Test bench per la verifica del corretto funzionamento:

1. Reset Asincrono: viene verificato che la ricezione di un segnale di reset inaspettato mentre i_start è a 0 non comprometta il funzionamento del componente e che questo venga correttamente riportato allo stato iniziale indipendente dallo stato corrente



2. 1000 indirizzi di memoria di lunghezza variabile: il test verifica il corretto funzionamento dell'automa quando sottoposto ad una serie di input.



Questo test è stato realizzato grazie ad uno script Python in grado di generare un numero arbitrario di casi di test e di salvarli in un file .vhd utilizzabile come test bench in Vivado.

Conclusioni

Risultato della sintesi

Il componente risulta sintetizzato correttamente e supera i test precedentemente indicati nelle simulazioni: Behavioral, Post-Synthesis Functional e Post-Synthesis Timing.
Di seguito confronto dei tempi di esecuzione di due test bench di 1000 input uno con indirizzi di minima dimensione (0 bit) e l'altro con indirizzi tutti di dimensione massima (16bit), con numero minimo di cicli tra un input e l'altro (6 cicli di clock).

Min size

Failure: Simulation Ended! TEST PASSATO
Time: 800500 ns Iteration: 0 Process: /project_tb/testRoutine File:
C:/Users/Seba/Desktop/RTL_project/project_reti_loghiche_2.0/project_reti_loghiche_2.0.srcs/sources_1/new/tb_1000eMinSize.vhd
$finish called at time: 800500 ns: File
"C:/Users/Seba/Desktop/RTL_project/project_reti_loghiche_2.0/project_reti_loghiche_2.0.srcs/sources_1/new/tb_1000eMinSize.vhd" Line 14139
run: Time (s): cpu = 00:00:09; elapsed = 00:00:08. Memory (MB): peak = 1052.844; gain = 0.000

Max size

Failure: Simulation Ended! TEST PASSATO
Time: 2400500 ns Iteration: 0 Process: /project_tb/testRoutine File:
C:/Users/Seba/Desktop/RTL_project/project_reti_loghiche_2.0/project_reti_loghiche_2.0.srcs/sources_1/new/tb_1000eMaxSize.vhd
$finish called at time: 2400500 ns: File
"C:/Users/Seba/Desktop/RTL_project/project_reti_loghiche_2.0/project_reti_loghiche_2.0.srcs/sources_1/new/tb_1000eMaxSize.vhd" Line 15138
run: Time (s): cpu = 00:01:09; elapsed = 00:01:12. Memory (MB): peak = 1417.016; gain = 0.000

Ottimizzazioni

Ho principalmente attuato ottimizzazioni volte alla riduzione del numero di stati inizialmente ogni bit di indirizzo veniva letto in uno stato differente.
Inoltra la prima versione del componente non era strutturato come una FSM ma bensì era composto da sottocomponenti ognuno dei quali era adibito ad una funzione:

- lettura della porta di input e lettura dell'indirizzo di memoria
- scambio dati con la memoria
- scrittura dei dati sulle porte di uscita

Questo approccio è stato abbandonato per problemi di sincronizzazione tra i vari componenti che mi hanno portato ad optare per una FSM.