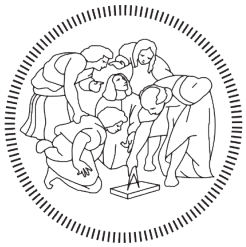


POLITECNICO
MILANO 1863

POLITECNICO DI MILANO

INGEGNERIA INFORMATICA

RELAZIONE PROVA FINALE RETI LOGICHE
ANNO SCOLASTICO: 2022/2023

Docente di Riferimento:
Prof. Gianluca Palermo

Progetto di:
Arimondo Scrivano
Codice Persona 10712429

Anna Sonzini
Codice Persona 10730238

Anno accademico:
2022/2023

Indice

Introduzione

```
// Intefaccia del componente
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

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;
```

Il progetto, descritto in VHDL tramite l'interfaccia riportata sopra, implementa un componente con i seguenti ingressi/uscite:

- i _w (bit di ingresso)
- i _start (bit di inizio trasmissione)
- i _rst (bit per il reset)
- i _mem _data (contenuto della memoria /8 bit)
- o_mem _addr (indirizzo di 16 bit da inviare alla memoria)
- le 4 uscite o_Z0, o_Z1, o_Z2, o_Z3
- il bit di fine trasmissione o _done
- i segnali per il controllo sulla memoria o_mem _we e o_mem _en

Il Comportamento del componente è di ricevere una serie di bit sul canale i _w. la trasmissione comincia quando il segnale i_start è uguale a '1' e si conclude quando questo torna a '0'. i primi due bit della trasmissione identificano il canale d'uscita (00->o _Z0, 01-> o _Z1, 10-> o _Z2, 11-> o _Z3), gli altri indicano l'indirizzo di memoria nel quale è salvato il contenuto di 8 bit che dev'essere mostrato sull'uscita corrispondente. Quando la trasmissione dell'indirizzo è inferiore ai 16 bit necessari, questo viene riempito di zeri a sinistra fino ad arrivare al numero esatto.

Finita la ricezione, quando il bit di done è asserito a '1', per un singolo ciclo di clock l'uscita modificata mostra il nuovo valore, invece le altre uscite non modificate mostrano i valori precedentemente salvati, per poi, esaurito il ciclo di clock, garantire o _done a zero, e le uscite a '0000 0000' .

Implementazioe del Componente

Il progetto contiene una FSM con 6 stati, 4 registri sincronizzati sul fronte di salita del clock e quattro processi per la gestione di diverse funzionalità: la ricezione, il completamento dell'indirizzo da inviare alla memoria, l'aggiornamento dei registri e il display delle uscite.

La FSM gestisce i diversi stati e la corretta esecuzione dei diversi processi tramite differenti segnali di enable

```
// primo processo

--PROCESSO + REGISTRO DI RICEZIONE INDIRIZZO USCITA
ADD_RECEIVER: process(i_rst,add_receiver_rst, i_clk)
begin
if add_receiver_rst='1' or i_rst='1' then
    raw_address<="0000000000000000";
    fill_address<= 18;
elsif i_clk'event and i_clk='1' then
if i_start='1' then
        raw_address<=raw_address(16 downto 0)& i_w;
        fill_address<= fill_address-1;
end if;
end if;
end process;
```

Il primo processo, sincronizzato con il clock, salva sul vettore raw _address, quando i _start ='1', il valore di i _w nella posizione fill _address, per poi incrementare questo contatore. la scelta dell'implementazione del contatore è per compiere la corretta traduzione nel processo successivo dell'indirizzo della memoria e dell'indirizzo di uscita.

Questo è l'unico processo indipendente dalla FSM. Si è scelta questa implementazione per via del ritardo di un eventuale segnale di enable che, posto a '1' dalla FSM, non sarebbe stato visibile dal processo durante la trasmissione del primo bit valido ma solo al ciclo di clock successivo. Questo passo si colloca nello Stato S1 della FSM.

```
// secondo processo

CONV_ADD: process(i_rst, i_clk,conv_add_rst)
begin
if i_rst='1' or conv_add_rst='1' then
```

```
o_mem_addr<="0000000000000000";
add_out<="00";
else
if rising_edge(i_clk) then
if conv_add_start<= '1' then

case fill_address is
when 16=>
    o_mem_addr<="0000000000000000";
    add_out<= raw_address(1 downto 0);
 when 15=>
    o_mem_addr<="000000000000000"& raw_address(0);
    add_out<= raw_address(2 downto 1);
  when 14=>
    o_mem_addr<="00000000000000" & raw_address(1 downto 0);
    add_out<= raw_address(3 downto 2);
   when 13=>
    o_mem_addr<="0000000000000" & raw_address(2 downto 0);
    add_out<= raw_address(4 downto 3);
    when 12=>
    o_mem_addr<="000000000000" & raw_address(3 downto 0);
    add_out<= raw_address(5 downto 4);
     when 11=>
    o_mem_addr<="00000000000" & raw_address(4 downto 0);
    add_out<= raw_address(6 downto 5);
      when 10=>
    o_mem_addr<="0000000000" & raw_address(5 downto 0);
    add_out<= raw_address(7 downto 6);
       when 9=>
    o_mem_addr<="000000000" & raw_address(6 downto 0);
    add_out<= raw_address(8 downto 7);
        when 8=>
    o_mem_addr<="00000000" & raw_address(7 downto 0);
    add_out<= raw_address(9 downto 8);
         when 7=>
    o_mem_addr<="0000000" & raw_address(8 downto 0);
    add_out<= raw_address(10 downto 9);
          when 6=>
    o_mem_addr<="000000" & raw_address(9 downto 0);
    add_out<= raw_address(11 downto 10);
           when 5=>
    o_mem_addr<="00000" & raw_address(10 downto 0);
    add_out<= raw_address(12 downto 11);
            when 4=>
    o_mem_addr<="0000" & raw_address(11 downto 0);
```

```
    add_out<= raw_address(13 downto 12);
             when 3=>
    o_mem_addr<="000" & raw_address(12 downto 0);
    add_out<= raw_address(14 downto 13);
              when 2=>
    o_mem_addr<="00" & raw_address(13 downto 0);
    add_out<= raw_address(15 downto 14);
               when 1=>
    o_mem_addr<="0" & raw_address(14 downto 0);
    add_out<= raw_address(16 downto 15);
                when 0=>
    o_mem_addr<= raw_address(15 downto 0);
    add_out<= raw_address(17 downto 16);
                 when others=>
    o_mem_addr<="0000000000000000";
    add_out<="00";
end case;
end if;
end if;
end if;
end process;
```

il secondo processo, sincronizzato con il clock, gestisce l'assegnamento dell'indirizzo di uscita e il rimpimento dell'indirizzo di 16 bit della memoria. Si attiva tramite il segnale di enable "conv _add _start = 1" dato dalla FSM durante il secondo Stato.

Il contatore del processo precedente permette, tramite switch case, di conoscere con esattezza quanti bit sono stati ricevuti e da questo determinare il riempimento del corretto numero di '0'.

```
// terzo processo

--PROCESSO + DECODER PER LA GESTIONE DEGLI ENABLE SUI REGISTRI

DECODER_ENABLE: process(start_decoding,i_rst)
begin
if i_rst='1' then
    enable_reg<="0000";
elsif start_decoding='1' then
        case add_out is
            when "00"=> enable_reg<="0001";
            when "01"=> enable_reg<="0010";
            when "10"=> enable_reg<="0100";
            when "11"=> enable_reg<="1000";
```

```
            when others=> enable_reg<= "0000";
        end case;
    else
        enable_reg<="0000";
    end if;
end process;
```

Il processo, asincrono, attivato dal segnale di enable "start _decoding" dalla FSM allo Stato S4, decide, dato il vettore add _out modificato al processo sopracitato, quali registri modificare.

L'implementazione dei Registri, sincronizzati con il segnale di clock, con Enable, è dovuta alla necessità di mantenere uno storico sulle uscite precedentemente modificate. Questo passo si colloca nello Stato S4 della FSM.

```
//quarto processo

--PROCESSO SELETTORE USCITA
SEL_USCITA: process(i_rst,s_display)
begin
if i_rst='1' then
    o_z0<="00000000";
    o_z1<="00000000";
    o_z2<="00000000";
    o_z3<="00000000";
    elsif s_display='1' then
        o_z0<= r_0;
        o_z1<= r_1;
        o_z2<= r_2;
        o_z3<= r_3;
    else
        o_z0<="00000000";
        o_z1<="00000000";
        o_z2<="00000000";
        o_z3<="00000000";
    end if;
end process;
```

Il processo, asincrono, seleziona, dato il segnale "s_display" modificato dalla FSM nello Stato S5, quali uscite far vedere, in modo tale da mostrare i valori dei registri solo al segnale alto di o _done e "0000 0000" negli altri casi.

```
//processo della gestione dei segnali di "enable" della FSM
--PROCESSO PER LA GESTIONE DEI SEGNALI DI AVANZAMENTO STATO  DELLA FSM
```

```
STATE_MANAGER: process(current_state)
begin
 o_done<='0';
 add_receiver_rst<='0';
 conv_add_rst<='0';
 conv_add_start<='0';
 s_display<='0';
 start_decoding<='0';
case current_state is
WHEN S0=>
--stato di reset--
WHEN S1=>
-- stiamo ricevendo l'indirizzo--

WHEN S2=>
conv_add_start<='1';
  --stiamo convertendo l'indirizzo--
WHEN S3=>


WHEN S4=>
  start_decoding<='1';
WHEN S5=>
  s_display<='1';
  o_done<='1';
  add_receiver_rst<='1';
  conv_add_rst<='1';
end case;
end process;
```

Il processo riportato sopra descrive l'assegnamento dei diversi enable dati i diversi Stati.

Lo Stato S0 è inserito come stato di Reset e come stato di riavvio per un'eventuale trasmissione futura.

Lo Stato S1, a cui si arriva quando istart assume il valore '1', è stato inserito per il primo processo.

Lo Stato S2, a cui si arriva quando i start scende a '0', è stato inserito per determinare quando iniziare la traduzione dell'indirizzo corretto.

Lo Stato S3 è stato inserito come stato di attesa. Il motivo di questa scelta è dettato dall'assegnamento a '1' del segnale "conv add start". Questo, essendo soggetto a ritardo, non è visibile al secondo processo durante il risingedge del clock e per evitare malfunzionamenti, si è voluto inserirlo.

Lo Stato S4, successivo di un ciclo di clock rispetto a S3, è inserito per assegnare a '1' il segnale di modifica dei registri e, infine, lo stato S5, successivo anch'esso di un ciclo di clock rispetto al precedente, è introdotto per poter far partire l'ultimo processo e per asserire a '1' il bit di uscita odone. Dopo questo, la macchina si riporta nello stato di reset S0;

Casi di test

Per verificare il corretto funzionamento del componente, questo è stato testato con i testbench forniti. In seguito, viene presentata una descrizione della risposta della macchina sintetizzata ai vari test e ai diversi casi critici.

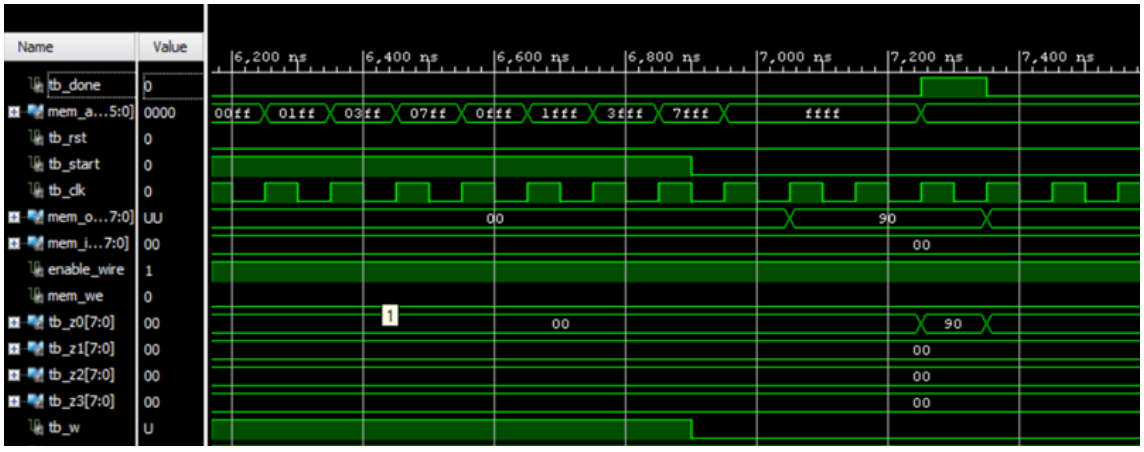
Simulazioni:

In generale, in ogni test viene verificata la capacità della macchina di mostrare correttamente sul canale d'uscita sollecitato il contenuto presente all'indirizzo corretto e garantisce, che ad ogni cambiamento del segnale di DONE, le uscite modificate in precedenza ripresentino il contenuto passato.

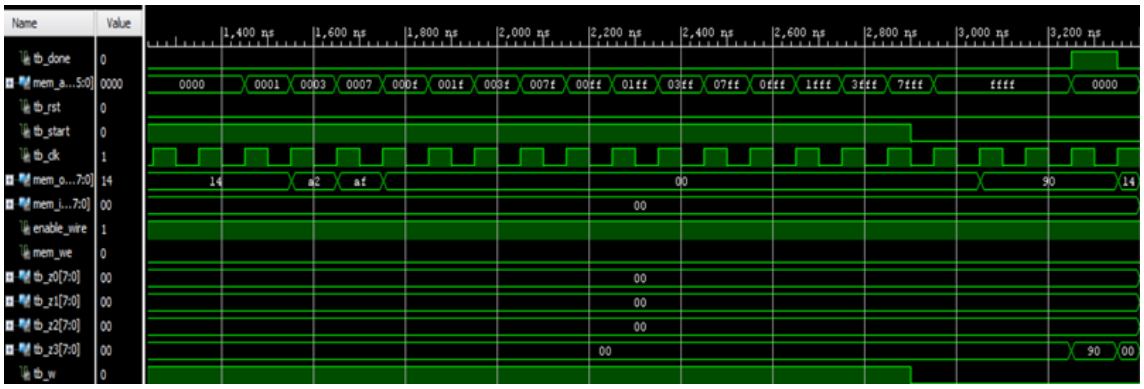
In particolare: nel quarto e nel quinto test viene evidenziato il corretto funzionamento del componente all'arrivo asincrono del segnale di RESET, in corrispondenza del quale la macchina viene, da specifica, sempre re-inizializzata. Nel sesto viene messo in evidenza che in corrispondenza della durata massima del segnale START=1 (18 cicli di clock in cui w presenta il valore '1'), il componente indirizza su z3 il contenuto corretto.

In tutti gli altri, ogni uscita viene sempre sollecitata con contenuti diversi e si può vedere come il contenuto memorizzato in precedenza viene ripresentato al cambio del DONE, quando non si è verificato il RESET.

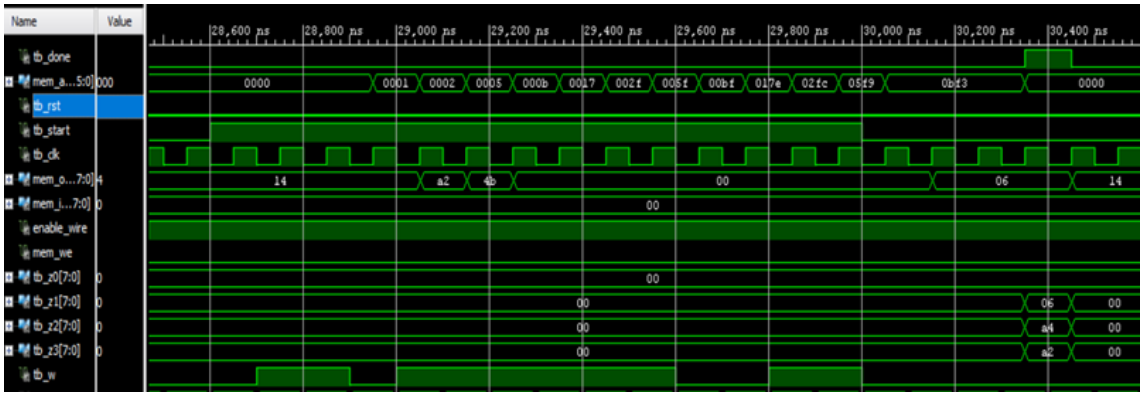
Di seguito sono riportate delle funzioni dei segnali post-sintesi più significative.



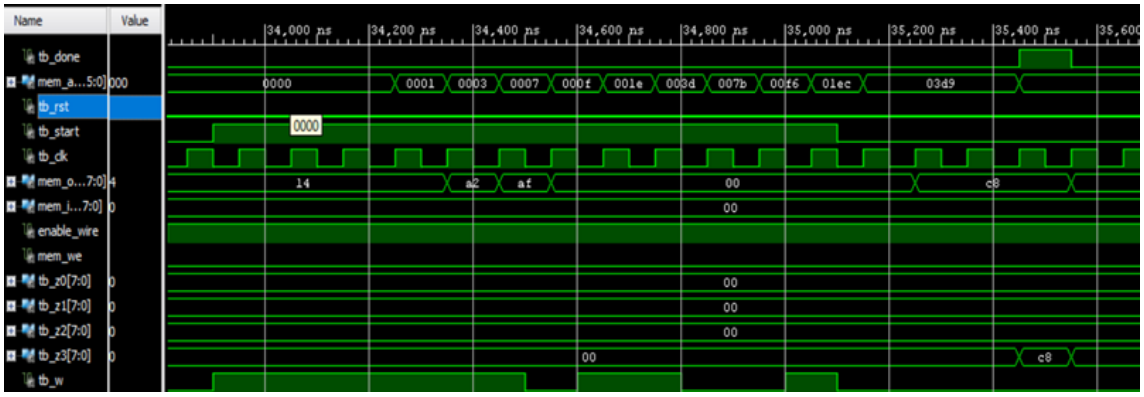
textTest bench 2: il segnale d'ingresso descrive il caso critico di indirizzo di memoria di 16 bit assegnati a '1'.



textTest bench 6: ricevuta la lunghezza massima in ingresso di 18 bit.



textTest bench 5 prima del reset: quando DONE passa a 1, si evidenziano le uscite con i valori correttamente ripresentati poiché memorizzati in precedenza.



textTest bench 5 dopo il reset: quando arriva il reset, il componente viene correttamente re-inizializzato, tutte le uscite vengono poste a 0 e al DONE successivo viene presentato come variato solo il nuovo valore letto sull'uscita corrispondente (in questo caso specifico, z3).

Risultati Post Sintesi

Il componente, eseguita la sintesi, continua a superare i testbench precedentemente eseguiti in simulazione passando i casi critici.

La macchina così progettata, inoltre, come evidenzia l'immagine sottoriportata, non ha elementi di tipo Latch ed esegue quanto voluto dalla specifica in circa 4 cicli di clock con periodo di 100 ns, rientrando nei requisiti di progetto.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	115	0	134600	0.09
LUT as Logic	115	0	134600	0.09
LUT as Memory	0	0	46200	0.00
Slice Registers	103	0	269200	0.04
Register as Flip Flop	103	0	269200	0.04
Register as Latch	0	0	269200	0.00
F7 Muxes	4	0	67300	<0.01
F8 Muxes	2	0	33650	<0.01