SCHEMA DELLA RELAZIONE

PROVA FINALE DI RETI LOGICHE

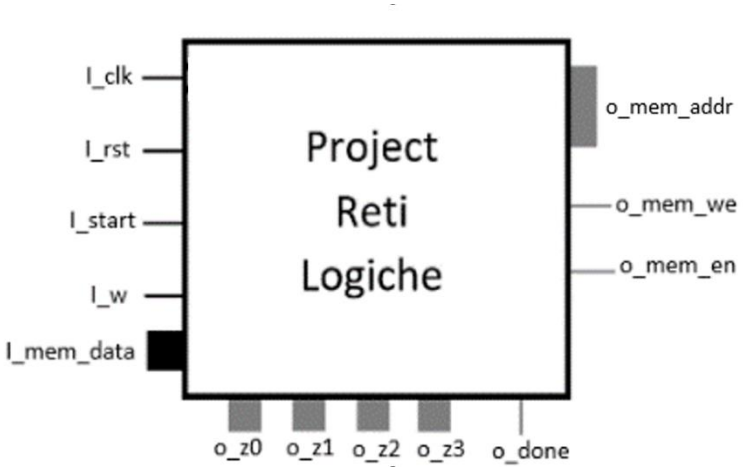
STUDENTE:	Lo Mastro Francesco Gregorio
MATRICOLA	956467
CODICE PERSONA	10709176
PROFESSORE DI PROGETTO	Salice Fabio
ANNO SCOLASTICO	2022-2023

Sommario

INTRODUZIONE

Il progetto di reti logiche dell'anno 2022/2023 richiede di progettare un componente hardware di supporto alla lettura di dati da una memoria.

Il componente può essere rappresentato come un dispositivo di questo aspetto:



In nero sono stati indicati i canali di ingresso, fra questi distinguiamo:

I_clk: canale di ingresso per il clock di sistema, il sistema ne avrà bisogno per coordinare le sue operazioni, specie sul fronte di salita. (1 bit)
I_rst: canale di ingresso per il reset del componente, se tale segnale è alto verranno resettati tutti i valori delle uscite, così come lo stato interno del componente. (1 bit)
I_start: canale di ingresso per il segnale di start. Con esso si avvia il componente e si da inizio alla fase di campionamento dell'ingresso "i_w". (1 bit)
I_w: canale di ingresso seriale per il componente. (1 bit)
I_mem_data: canale di ingresso utilizzato da una memoria esterna per mostrare il dato che le è stato richiesto. (8 bit)

In grigio invece sono mostrate i canali di uscita:

O_z0/1/2/3: Sono 4 canali utilizzabili dal componente per mostrare il risultato del proprio compito. (8 bit)
O_done: canale di uscita utilizzato come indicatore. Se alto indica la fine dell'utilizzo corrente. (1 bit)
O_mem_addr: canale di uscita collegato alla memoria esterna. Con questo canale il componente può comunicare alla memoria un indirizzo di memoria. (16 bit)
O_mem_en: canale di uscita collegato alla memoria esterna, lo si attiva se si intende abilitare la memoria per qualsiasi operazione (lettura o scrittura). (1 bit)
O_mem_we: canale di uscita collegato alla memoria esterna, lo si attiva se si intende scrivere in memoria. (1 bit)

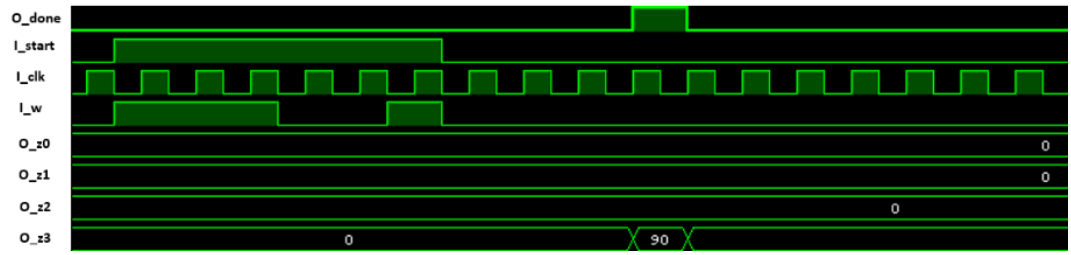
Il componente svolge un ruolo simile ad un dereferenziatore; come prima cosa riceve **2 bit di intestazione** dai quali determina l'uscita (O_zi) da utilizzare per l'output del risultato.
00 indica l'uscita O_z0, 01 invece fa riferimento a O_z1, 10 a O_z2 ed infine 11 indica O_z3.
Il componente riceve poi **un massimo di 16 bit** con cui ricostruisce un **indirizzo di memoria a 16 bit**.
Tale indirizzo viene inviato ad una memoria esterna (tramite o_mem_addr) la quale risponde con un dato da 8 bit (i_mem_data) che il componente dovrà mostrare nell'uscita identificata dai bit di intestazione raccolti.
Quando il risultato è pronto ad essere mostrato, il componente attiva il segnale "**o_done**" e simultaneamente tutte le uscite mostrano il valore piu recente che le ha (eventualmente) attraversate negli utilizzi precedenti.
Per tutto il resto del tempo, nella fase in cui "o_done" = 0, tutte le uscite mostrano costantemente il valore 0.

Vi sono alcuni di vincoli che i comandi che vengono impartiti ed i segnali generali devono rispettare:

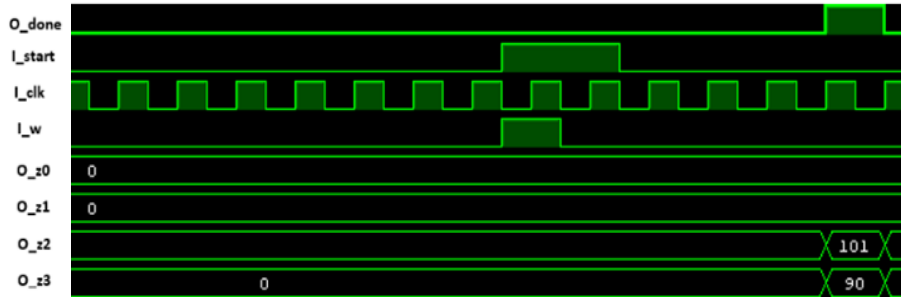
- O_done deve attivarsi per un solo ciclo di clock ad ogni utilizzo.
- Fra il fronte di discesa di ogni segnale “I_start” e il fronte di salita del successivo “o_done” possono passare al massimo 20 cicli di clock.
- Il segnale “i_w” deve essere campionato sul fronte di salita del clock e verrà acquisito solo se “I_start” è alto in quel momento.
- Se il numero di bit acquisiti dopo i 2 bit di intestazione è inferiore a 16, si aggiunge un padding di 0 sui bit piu significativi (pagging sinistro), fino a formare 16 bit.
- Ogni segnale di “I_start” è attivo per un massimo di 18 cicli e un minimo di 2 cicli.

ESEMPIO

Di seguito riportiamo un esempio di come il componente lavora e del flusso di alcuni dei segnali presenti:
Sono mostrati 2 utilizzi in successione del componente.



Nel primo utilizzo “I_start” è alto per 6 cicli di clock e consente di campionare i bit di intestazione “11_2” da “i_w”, quindi l’uscita scelta è “O_z3”.
Successivamente “I_start” consente di campionare i bit “1001_2” che vengono estesi a “0000 0000 0000 1001_2”.
Nella memoria esterna l’indirizzo “9_{10}” contiene il dato “90_{10}”. Tale dato viene comunicato al componente dalla memoria esterna e, quando sarà pronto, “o_done” si alzerà mostrandolo nell’uscita “O_z3”.



Nel secondo utilizzo “I_start” è alto per soli 2 cicli di clock e consente di campionare i bit di intestazione “10_2” (O_z2) ma nessun bit di indirizzo.
L’indirizzo di memoria dunque è interamente composto da padding “0000 0000 0000 0000_2”.
Il dato in questo indirizzo di memoria è “101_{10}” e verrà mostrato in “O_z2” quando sarà pronto.

Notare come, al secondo “O_done”, 90 e 101 si vedano entrambi nelle rispettive uscite O_z3 e O_z2, questo perché sono i valori piu recenti che tali uscite hanno mostrato. Le altre 2 uscite restano a 0 perché non sono state usate recentemente.
Inoltre, quando “O_done ” è basso tutte le uscite mostrano sempre 0.

ARCHITETTURA

Il componente è composto da 7 moduli

I 2 moduli REG_2 e REG_16

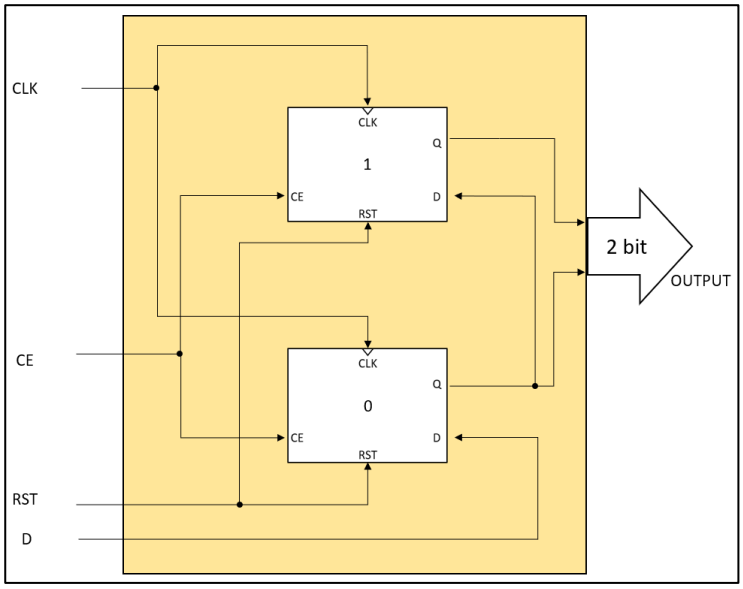
Il componente riceve gli input in modo seriale, un bit alla volta. È perciò indispensabile disporre di dispositivi che consentano la memorizzazione seriale ma allo stesso tempo in grado di poter mostrare tutti i bit memorizzati in parallelo.
In realtà si distinguono 2 input ben precisi quando "i_start" = 1:

- input destinato alla scelta dell'uscita del componente -> **Primi 2 bit ricevuti**
- input che identifica l'indirizzo di memoria da ispezionare -> **Prossimi 16 bit al massimo (eventuale padding)**

I 2 input verrengono memorizzati in due registri a scorrimento, chiamati rispettivamente **REG_2 e REG_16**.
I registri a scorrimento risultano molto comodi per questo utilizzo, infatti lo scorrimento consente di memorizzare i bit nello stesso ordine con cui sono arrivati e inoltre REG_16 non necessiterà di ulteriore tempo per aggiungere il padding nelle cifre piu significative.
Per assicurare questa funzionalità ogni registro è opportunamente azzerato mediante **RST** e prevede un Clock Enable (**CE**) per abilitarne e coordinarne la scrittura.
I registri ottengono i dati dall'igresso D, il quale è collegato alla cella di memoria 0, rappresentante il bit meno significativo del registro.

La struttura di REG_2 e REG_16 è molto simile, la differenza è data dal numero di celle di memoria (Flip Flop D) disponibili in ciascuno di essi, rispettivamente 2 e 16.
Questi Flip Flop sono sensibili al fronte di salita del clock (**CLK**), tuttavia saranno abilitati alla memorizzazione solo se il segnale **CE** (Clock Enable) è attivo in quell'istante.

Per semplicità mostriamo la struttura del solo REG_2 (REG_16 è analogo):



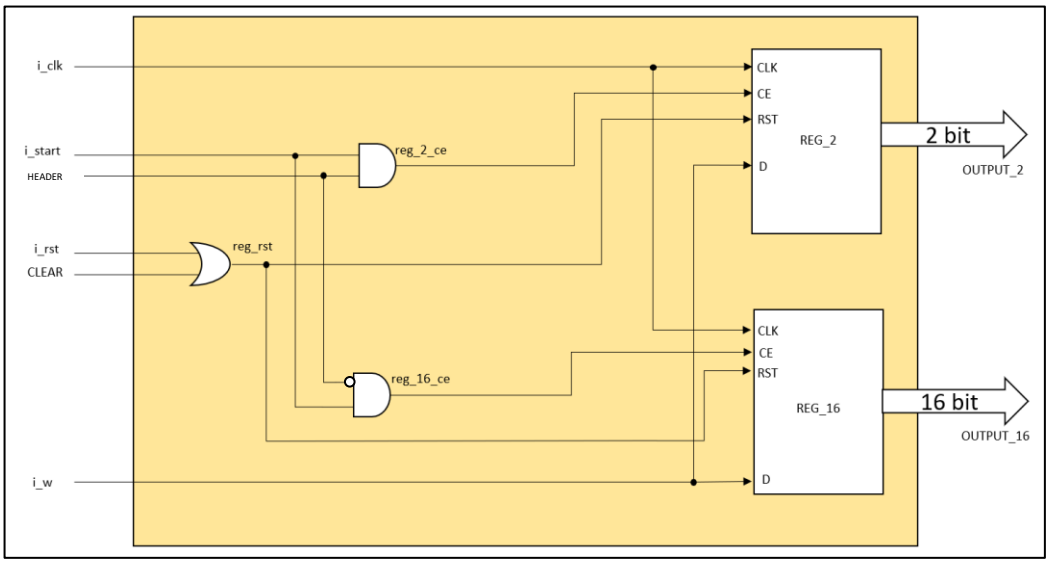

```
----------------------------------------------------------------------
--                  REG_2
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg_2 is
    Port(
            CLK: in std_logic;
            RST: in std_logic;
            CE: in std_logic;
            D: in std_logic;
            OUTPUT: out std_logic_vector(1 downto 0)
        );
end reg_2;

architecture Behavioral of reg_2 is
signal internal_output: std_logic_vector(1 downto 0);
begin

process(CLK,RST)
begin
    if(RST='1') then
        internal_output<="00";
    elsif (CLK'event and CLK='1') then
         if (CE = '1') then
               internal_output<=internal_output(0) & D;
         end if;
    end if;
end process;

OUTPUT<=internal_output;
end Behavioral;
```

***NOTA** Le celle dei registri sono numerate, il numero più alto identifica il bit piu significativo, percio lo scorrimento è uno **Shift Sinistro**.*
***NOTA** I registri **non sono circolari**, perciò l'informazione presente nella cella con numero piu alto verrà persa e sostituita.*
****NOTA** Il segnale interno "**internal_ouput**" corrisponde all'uscita OUTPUT del registro in questione. È necessario in quanto Vivado non consente l'utilizzo di un segnale di uscita nella espresione di shift.*

I 2 moduli per il controllo dei registri

Si è deciso di separare la logica di controllo dei 2 registri in una parte Datapath ed in una parte FSM.

DATAPATH_REGISTRI



Questo datapath mostra come sono stati incapsulati i moduli REG_2 e REG_16, inoltre viene mostrato come vengono generati i segnali Clock Enable (CE) e RST di ciascun registro:

- "reg_rst" è una **OR** tra i segnali "i_rst" ed il segnale "CLEAR".
- "reg_2_ce" è una **AND** tra i segnali "i_start" ed il segnale "PRIMIDUE".
- "reg_16_ce" è una **AND** tra i segnali "i_start" ed il segnale "PRIMIDUE" **negato**.

I segnali "i_rst", "i_start", "i_clk", "i_w" sono i segnali originali di cui abbiamo parlato nell'introduzione.
CLEAR e **HEADER** sono segnali creati ed imposti dalla FSM che illustreremo nel prossimo modulo.
Il datapath raccoglie i 2 contenuti dei registri a scorrimento e li trasmette all'esterno tramite i segnali **OUPUT_2** e **OUTPUT_16**.

```
----------------------------------------------------------
--              DATAPATH_REGISTRI
----------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity datapath_registri is
    Port(
           i_clk: in std_logic;
           i_rst: in std_logic;
           i_start: in std_logic;
           i_w: in std_logic;
           CLEAR: in std_logic;
           HEADER: in std_logic;
           OUTPUT_2: out std_logic_vector(1 downto 0);
           OUTPUT_16: out std_logic_vector(15 downto 0)
       );
end datapath_registri;
```

```
architecture Behavioral of datapath_registri is
signal reg_2_ce: std_logic;
signal reg_16_ce: std_logic;
signal rst_reg: std_logic;

component reg_2 is
    Port(
           CLK: in std_logic;
           RST: in std_logic;
           CE: in std_logic;
           D: in std_logic;
           OUTPUT: out std_logic_vector(1 downto 0)
       );
end component;

component reg_16 is
   Port(
           CLK: in std_logic;
           RST: in std_logic;
           CE: in std_logic;
           D: in std_logic;
           OUTPUT: out std_logic_vector(15 downto 0)
       );
end component;
```

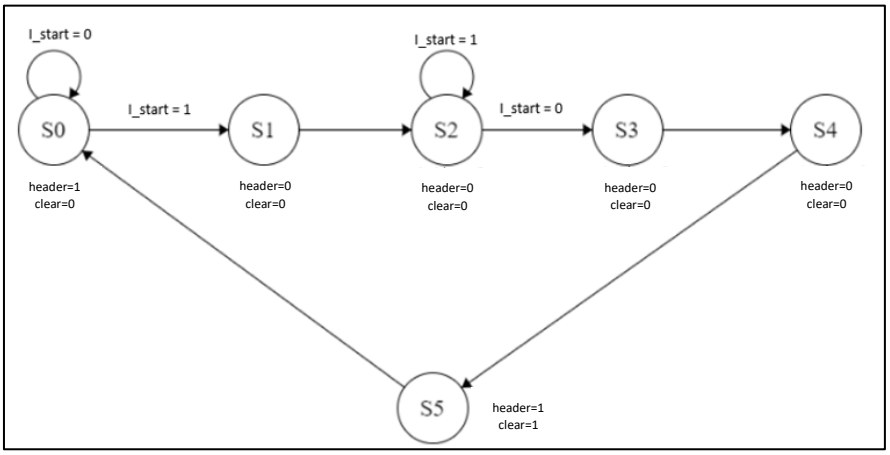
```
begin
reg_2_ce <= i_start and HEADER;
reg_16_ce <= i_start and not HEADER;
rst_reg <= i_rst or CLEAR;

REG2: reg_2 port map
(
   i_clk,
   rst_reg,
   reg_2_ce,
   i_w,
   OUTPUT_2
);
REG16: reg_16 port map
(
   i_clk,
   rst_reg,
   reg_16_ce,
   i_w,
   OUTPUT_16
);

end Behavioral;
```

FSM_REGISTRI

La macchina a stati finiti che getisce il datapath_registri è progettata sulla base di una macchina di Moore.
Di norma essa cambia stato **sul fronte di salita del clock** tuttavia è sensibile al segnale **"i_rst" in modo asincrono**, ciò significa che, in caso di "i_rst" = 1, la macchina si resetta immediatamente senza badare al clock.
La macchina a stati controlla il datapath_registri fornendo i 2 segnali "clear" e "header" generati con la seguente logica:

- **"header"**: è un segnale che, se alto, identifica che si stanno memorizzando (o si dovranno memorizzare) i primi due bit di intestazione che andranno in REG_2. Se invece tale segnale fosse basso, i bit ricevuti andrebbero in REG_16 in quanto non farnno parte dell'intestazione.
- **"clear"**: è un segnale che ha le stesse funzionalità del segnale "i_rst" in quanto, se alto, resetta e azzera il contenuto dei registri.
  "I_rst" è un segnale controllato dall'esterno del componente, non può quindi essere imposto dallo stesso componente per la preparazione dei registri fra un utilizzo ed un altro.
  Inoltre "I_rst" ha un impatto su tutti i moduli del componente (anche quelli di cui dobbiamo ancora parlare), "clear" invece è scelto dalla FSM ed avrà un impatto solo sul datapath dei registri e sui registri REG_2 e REG_16.

FSM_registri:



S0: Stato iniziale in cui la macchina permane fintanto che "i_start" rimane basso. È anche lo stato di reset, in cui la macchina viene portata ogni volta che "i_rst" = 1 ("clear" non viene interpretato come reset della FSM).
"header" = 1 in quanto, già da adesso "i_start" potrebbe essere alto, in tal caso il sistema deve essere pronto ad acquisire il primo bit di intestazione e prepararsi ad entrare in S1 al prossimo fronte.

S1: Stato in cui il secondo bit di intestazione viene acquisito.
"header" = 0 viene eseguito appena dopo il fronte di salita del clock, quindi il modulo acquisisce il secondo bit di intestazione e poi abbassa questo segnale.

S2: Stato in cui si acquisiscono i possibili bit di indirizzo, fintanto che "i_start" = 1.
"header" e "clear" vengono mantenuti a 0 fino ad S5.

S3/S4: Stati di attesa.
Rimandiamo a dopo la spiegazione di questi 2 stati di attesa.

S5: Stato di preparazione al prossimo utilizzo.
"clear" = 1 pulisce il contenuto dei registri per l'utilizzo successivo.
"header" =1 prepara già il prossimo utilizzo.

***NOTA:** La macchina a stati in questione prevede alcuni stati di attesa, tali stati sono progettati per lavorare in sintonia con la "FSM_uscite" di cui parleremo piu avanti.*
Infatti, se togliessimo tali stati, S5 pulirebbe i registri prima che possano essere utilizzati correttamente dalla macchina a stati delle uscite e si perderebbe l'informazione acquisita.
***NOTA:** Per semplicità di rappresentazione non sono stati rappresentati gli archi che prevedono il reset della macchina, essi sono presenti in ogni stato e portano in S0.*
****NOTA:** Nel codice VHDL "state" prende il posto dell'usuale "next_state", si è deciso di non usare alcun "curr_state".*

I 2 moduli per la gestione delle uscite

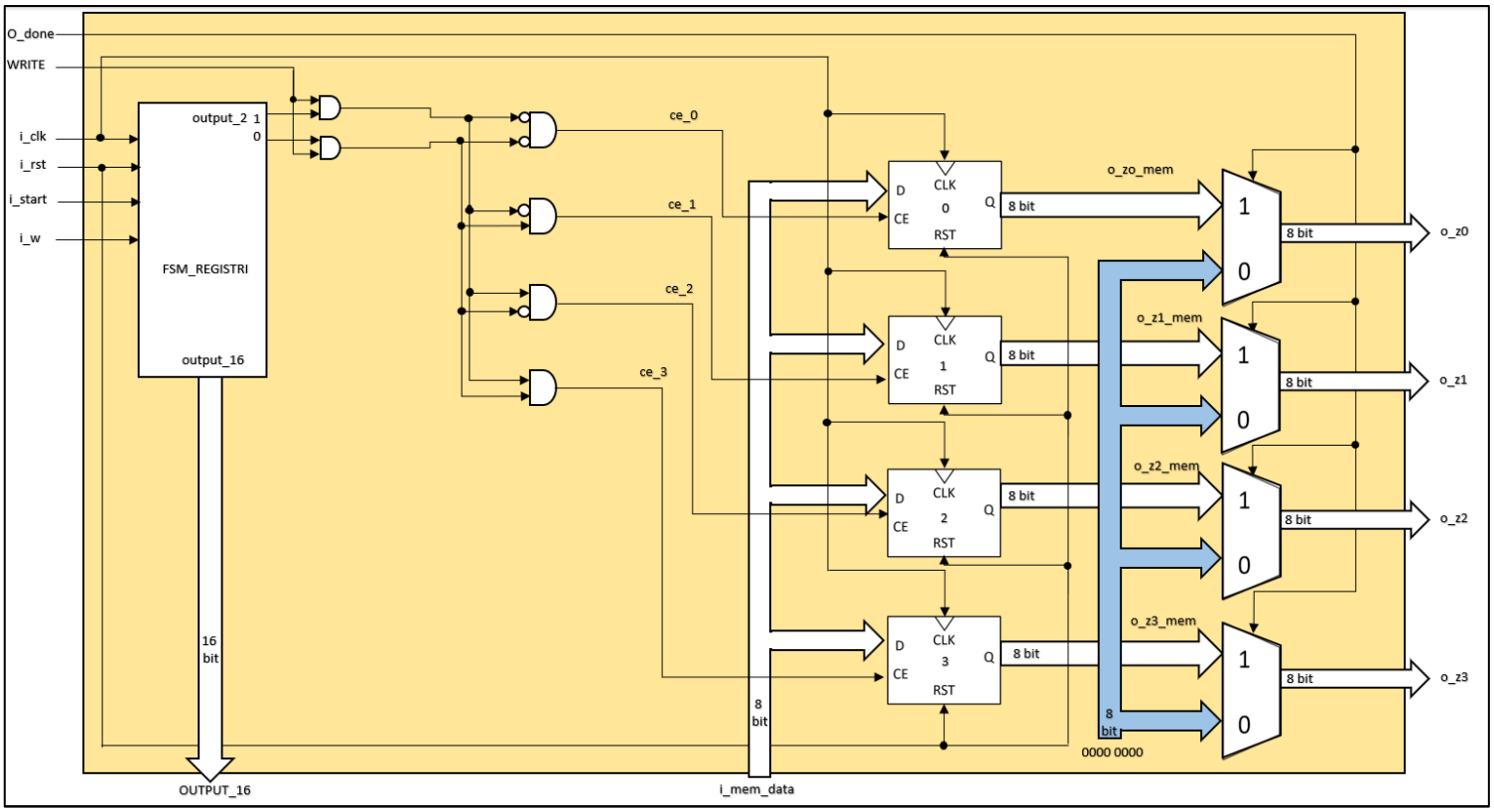
Il componente è ora in grado di memorizzare correttamente gli input fornitogli separandoli in intestazione ed indirizzo di memoria.
Di seguito viene illustrato un componente in grado di inviare l'indirizzo di memoria memorizzato alla memoria, ottenere il dato di 8 bit da essa e comunicare quest'ultimo all'esterno, nell'uscita corretta.
Anche tale componente è diviso in Datapath e FSM.

DATAPATH_USCITE

Il datapath di questo componente contiene il modulo FSM_REGISTRI, dal quale estrae i segnali "output_16" e "output_2", corrispondenti al contenuto dei registri REG_2 e REG_16 di cui abbiamo gia parlato.
"output_16" viene semplicemente propagato all'esterno (alla FSM del componente) mentre "output_2" è utilizzato per produrre i dei segnali di CE per 4 nuovi registri.
Il componente infatti dispone di 4 canali di uscita chiamati "o_zi", questi canali provengono da 4 registri da 8 bit chiamati "o_zi_mem".
Ogni registro è in grado di acquisire il segnale "i_mem_data" solo se "CLK" è sul fronte di salita ed il relativo "CE" risulta abilitato ed è resettabile tramite "i_rst"
I Clock Enable sono delle combinazioni fra i segnali "WRITE" (ricevuto dalla FSM esterna) e "output_2":

- ce_0: (**NOT** output_2[1]) **AND** (**NOT** output_2[0]) **AND** WRITE
- ce_1: (**NOT** output_2[1]) **AND** output_2[0] **AND** WRITE
- ce_2: output_2[1] **AND** (**NOT** output_2[0]) **AND** WRITE
- ce_3: output_2[1] **AND** output_2[0] **AND WRITE**

Il sistema può essere inteso nel seguente modo:
<u><< il segnale "WRITE" indica quando scrivere, il segnale "output_2" indica dove scrivere >></u>
I quattro canali sono propagati all'esterno utilizzando 4 multiplexer, questi consentono di mostrare il contenuto dei registri oppure "0000 0000_2" se il segnale selettore "o_done" è 0 oppure 1.
Il segnale "o_done" è prodotto dalla FSM del datapath e corrisponde al segnale richiesto da specifica.



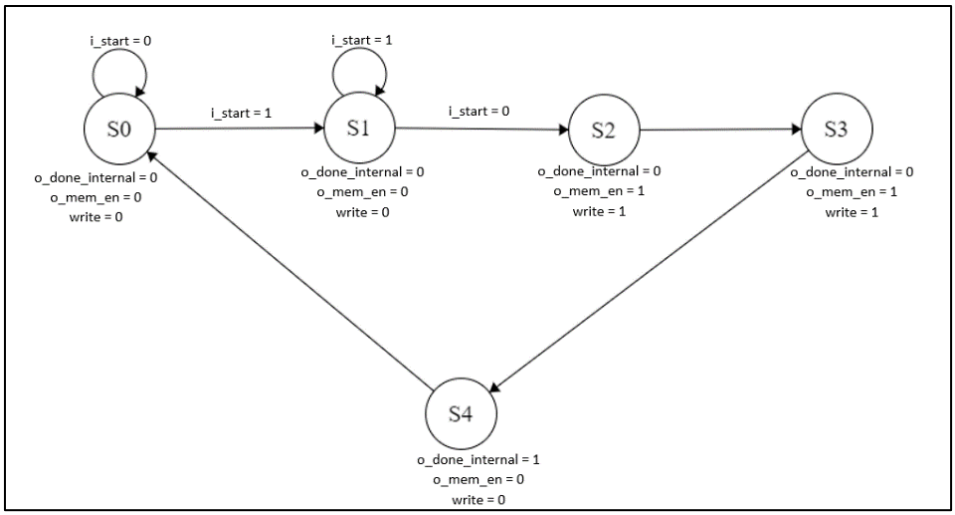
FSM_USCITE

La macchina a stati finiti che gestisce il datapath_uscite ha una struttura molto simile alla FSM_registri e funziona praticamente allo stesso modo, è sincrona al clock, resettabile tramite "i_rst" in modo asincrono ed è parallela a FSM_registri.

Questa macchina produce i segnali "o_done", "write" e gestisce "o_mem_en":

- "**o_done**" è il segnale richiesto da specifica, che viene propagato all'esterno e anche introdotto all'interno del datapath_uscite, questo serve scegliere se mostrare lo 0 oppure il contenuto dei 4 registri di cui abbiamo parlato. Il segnale è prodotto con il nome di "**o_done_internal**".
- "**write**" è un segnale che abilita la scrittura in uno dei 4 registri di uscita e viene alzato quando la memoria risponde alla richiesta di un dato in memoria.
- "**o_mem_en**" è un segnale che abilita la lettura da memoria.

FSM_uscite:



S0: Analogo ad S0 nella FSM_registri, è lo stato iniziale in cui la macchina permane fintanto che "i_start" = 0.

S1: Stato in cui la macchina permane fintanto che "i_start" = 1. Nel frattempo FSM_registri attraversa S1 ed S2

S2: Stato di richiesta alla memoria esterna.

I registri sono pronti appena "i_start" = 0. In questo stato si fa richiesta alla mamoria attivando i segnale "o_mem_en" e si prepara la scrittura dei registri con "write" = 1.

S3: Stato di attesa.

Dato che la memoria presenta un ritardo variabile (1ns nelle specifiche, 2 ns nel test banch), si decide di aspettare un altro ciclo di clock in questo stato, prolungando i segnali precedentemente utilizzati.

S4: Stato di "o_done" = 1

A questo punto il dato proveniente dalla memoria è stato scritto, si azzerano "o_mem_en", "write" e si attiva "o_done_internal" per un ciclo di clock (Al ciclo dopo verrà spento da S0)

La memoria esterna forza la FSM_uscite ad aspettare un totale di 2 cicli di clock prima di mostrare "o_done"=1 (una volta trovato "i_start" = 0), questi stati di attesa sono equivalenti agli stati S3 ed S4 nella FSM_registri e ne motivano l'utilizzo (Le macchine devono andare il parallelo).

***NOTA:** Anche qui, per semplicità di rappresentazione, si è evitato di mostrare gli archi di reset che condurrebbero a S0.*

***NOTA:** La macchina a stati non tratta il segnale "**o_mem_we**" il motivo è che il componente non scrive mai in memoria, quindi tale segnale è costantemente lasciato a 0.*

***NOTA**: La macchina a stati collega "o_mem_addr" direttamente a REG_16. Su tale canale saranno quindi visibili tutti i cambiamenti di REG_16. Ricordiamo però che "o_mem_addr" viene letto solo se "o_mem_en" = 1*

PROJECT_RETI_LOGICHE

Una semplice “scatola” che cambia il nome del componente e lo rende compatibile con le direttive date da specifica (stesso nome delle porte nella entity).

RISULTATI SPERIENTALI

Report Timing:

Il comando “report timing” restituisce uno **Slack (MET) : 7.516ns (required time - arrival time)**

Report utilization:

Il comando “report_utilization” mostra un utilizzo totale **di 78 Flip Flop**, **0 Latch** e **30 LUT**.
18 Flip Flop sono dovuti a REG2 e REG16, 8*4 = **32 Flip Flop** sono dovuti ai quattro registri da 8 bit denominati “o_zi_mem”.

Simulazioni:

Test Bench Pubblico

Il componente passa il test bench pubblico fornito dai docenti in *Behavoral Simulation:*

```
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)

Time: 3800 ns  Iteration: 0  Process: /project_tb/testRoutine  File:
C:/Users/Francesco/Desktop/project_final/project_final.srcs/sim_1/new/test_banch_pubblico.vhd

$finish called at time : 3800 ns : File
"C:/Users/Francesco/Desktop/project_final/project_final.srcs/sim_1/new/test_banch_pubblico.vhd" Line 153
```

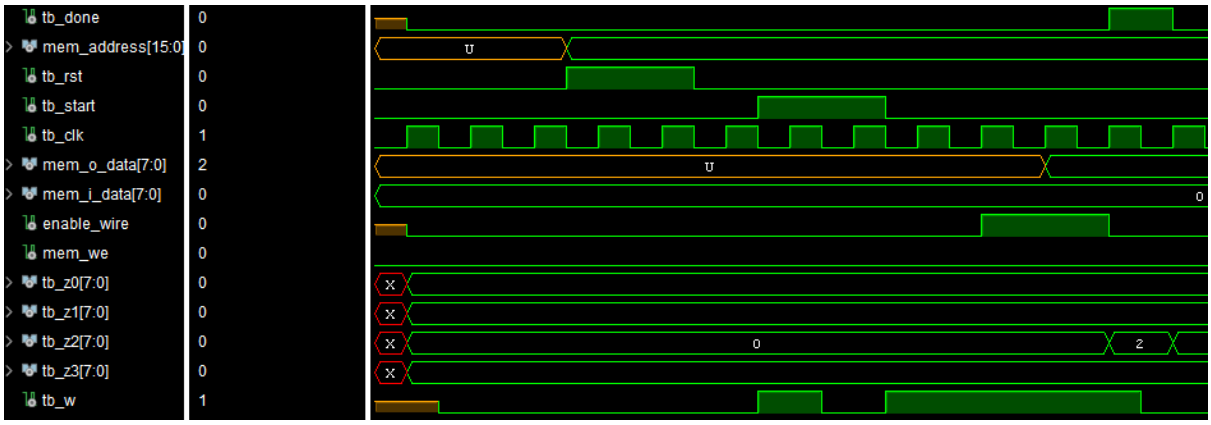
E anche il *Post-Synthesis Simulation:*

```
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)

Time: 3800100 ps  Iteration: 0  Process: /project_tb/testRoutine  File:
C:/Users/Francesco/Desktop/project_final/project_final.srcs/sim_1/new/test_banch_pubblico.vhd

$finish called at time : 3800100 ps : File
"C:/Users/Francesco/Desktop/project_final/project_final.srcs/sim_1/new/test_banch_pubblico.vhd" Line 153
```

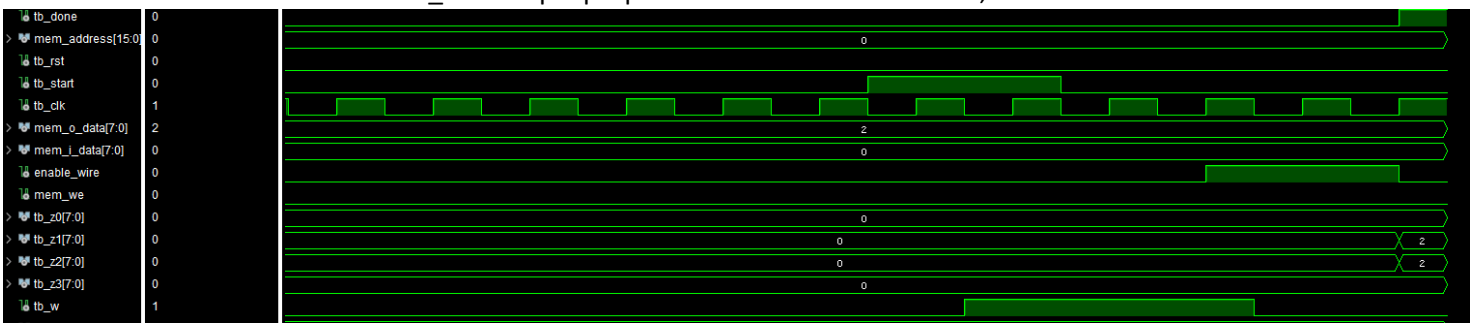
La sintesi sembra ritardare tutti i segnali creati dalle FSM di 100 ps, ciò non influisce sulla logica del componente, tuttavia porta il segnale “o_done” a non essere contemporaneo al fronte alto del clock.

Test Bench 1

Il test bech verifica il funzionamento del componente nel caso “i_start” = 1 per 2 cicli di clock. L’indirizzo di memoria che dovrebbe essere interpretato è quindi “0000 0000 0000 0000”, contenente il dato 2.



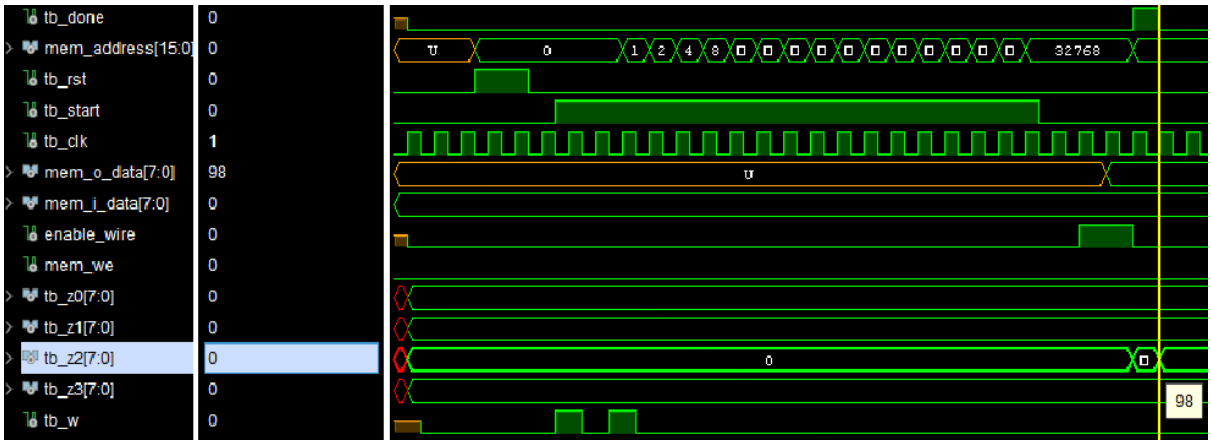
Un secondo utilizzo di “i_start” è poi proposto su una uscita differente, stesso indirizzo di memoria.



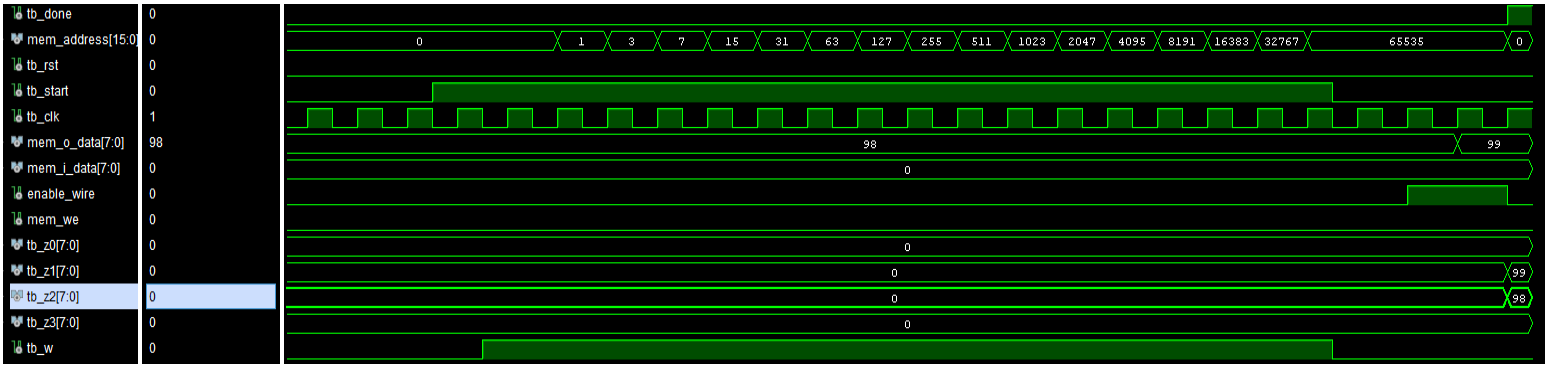
Test superato sia in *Behavoral Simulation* che in *Post-Synthesis Simulation.*

Test Bench 2

Il test bech verifica il funzionamento del componente nel caso “i_start” = 1 per 18 cicli di clock. L’indirizzo di memoria che dovrebbe essere interpretato è quindi “1000 0000 0000 0000”, contenente il dato 98.



C’è un secondo input in cui start è ancora alto per 18 cicli di clock ma questa volta “i_w” è sempre 1. Il secondo indirizzo di memoria che dovrebbe essere interpretato è quindi "1111 1111 1111 1111”, contenente il dato 99.



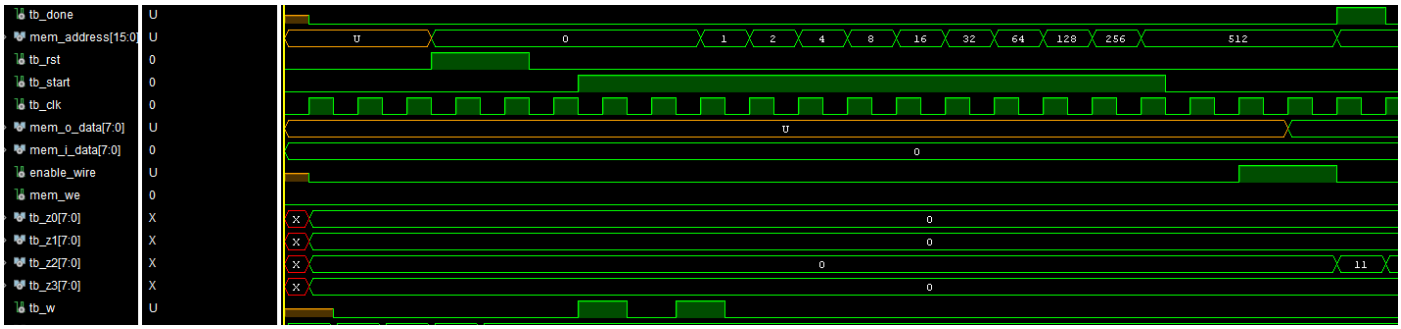
Test superato sia in *Behavoral Simulation* che in *Post-Synthesis Simulation.*

Test Bench 3

Il test bech verifica il funzionamento del componente nel caso “i_start” = 1 per un numero compreso tra 2 e 18 cicli, scelgo 12 cicli. Questo test ha lo scopo di verificare il padding di 0 a sinistra dell’indirizzo raccolto.

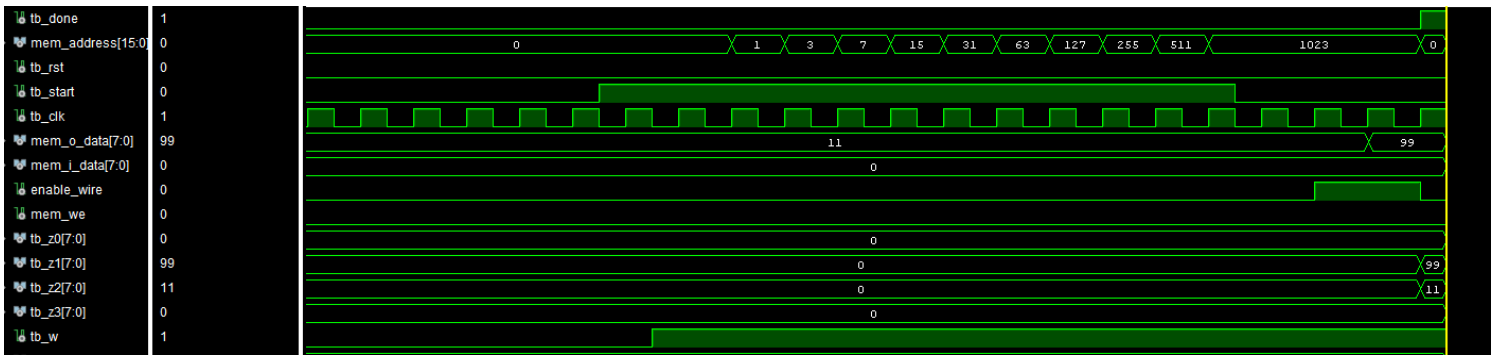
Il primo indirizzo di memoria che dovrebbe essere interpretato è “0000 0010 0000 0000”, dato che “i_start” = 1 per 12 cicli di clock, si aggiungono 4 bit di padding a sinistra.

L’indirizzo 512 contiene il valore 11.



Il secondo indirizzo di memoria che dovrebbe essere interpretato è “0000 0011 1111 1111”, per lo stesso motivo di prima.

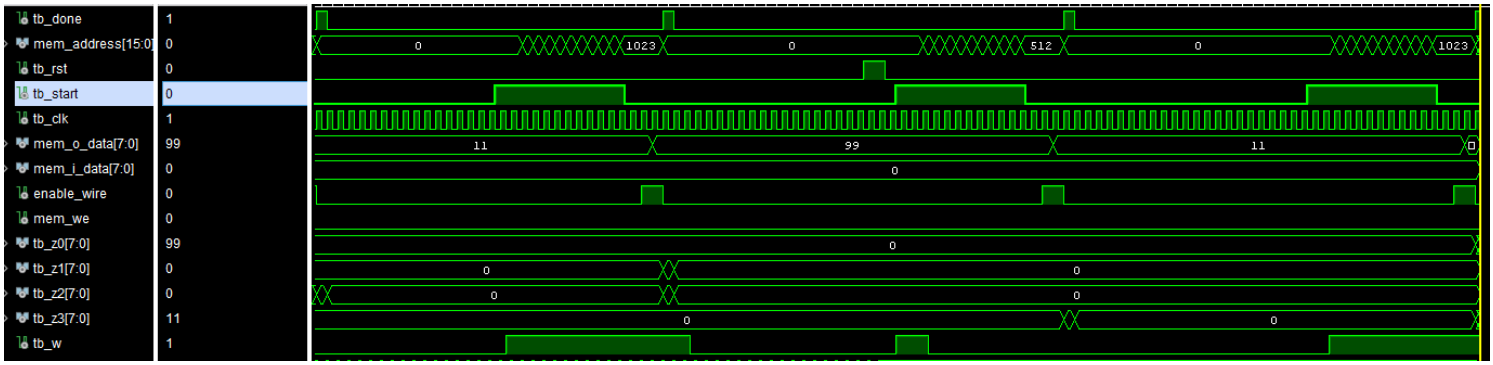
L’indirizzo 1023 contiene il valore 99.



Test superato sia in *Behavoral Simulation* che in *Post-Synthesis Simulation.*

Test Bench 4

Il test bech verifica il funzionamento del reset dei registri. Si utilizza un test simile al test 3, in cui viene però alzato il segnale di reset dopo 2 utilizzi di “i_start”, si riprende poi a ripetere un test di funzionamento con 2 ulteriori utilizzi di “i_start”.



Dal grafico si nota come le uscite perdano correttamente i valori memorizzati prima del restart.

Test superato sia in *Behavoral Simulation* che in *Post-Synthesis Simulation.*

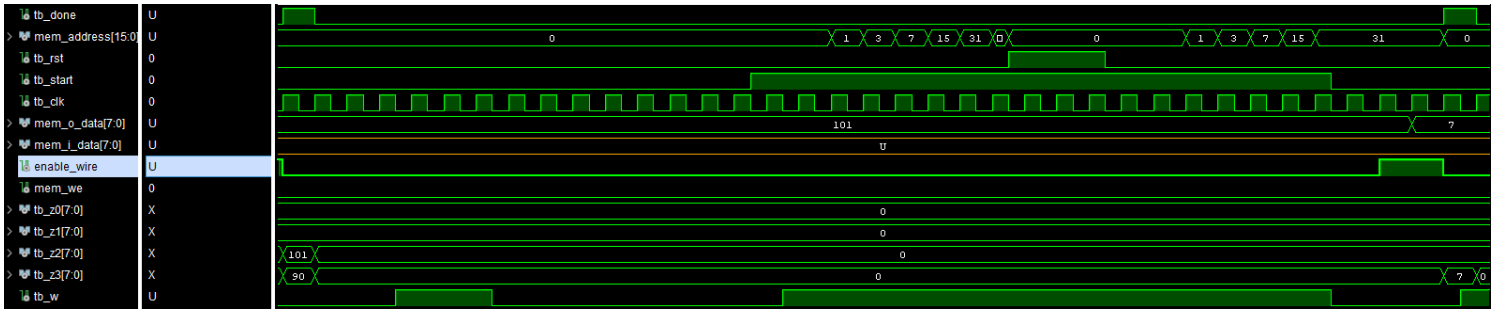
Test Bench 5

Questo test è simile al 4 ma più lungo (scenario di 222 bit). La differenza è che testa il funzionamento del reset e di tutti e 4 i canali di uscita.
(Grafico troppo ampio per essere riportato sul PDF).
Test superato sia in *Behavoral Simulation* che in *Post-Synthesis Simulation.*

Test Bench 6

Variante del test 5, qui “i_rst” si alza nel mezzo di un utilizzo di “i_start”. Il sistema dovrebbe annullare le operazioni e ricominciare non appena “i_rst” si abbassa (perché “i_start” è ancora alto).

(Riportiamo il grafico di tale sezione)



Dopo “i_rst”, “i_start” dovrebbe riuscire a cambionare un indirizzo “0000 0000 0001 1111”, contenente il valore 7. I precedenti valori delle uscite devono essere cancellati.
Test superato sia in *Behavoral Simulation* che in *Post-Synthesis Simulation.*

Conclusioni:

Tutti I test bench hanno rivelato che il progetto rispetta la specifica dei massimo 20 cicli di clock da quando i_start si azzera fino a quando o_done = 1.
Infatti il progetto è stato studiato per non impiegare alcun tipo di tempo aggiuntivo per l'apposizione del padding dunque, quando "i_start" si abbassa, il componente ha i registri pronti a comunicare l'indirizzo di memoria in cui prendere il dato. Bisogna solo aspettare che tale dato pervenga.
Dal momento in cui "i_start" si abbassa (includendo il tempo con cui la macchina si accorge del cambiamento al prossimo fronte di salita) fino al momento in cui "o_done" si alza possono intercorrere massimo 4 cicli di clock.
Tale tempistica potrebbe essere ottimizzabile ma ciò richiederebbe di modificare le FSM in modo da renderle sensibili ai cambiamenti di "i_start". Per evitare di mischiare macchine di MELEE e MOORE si è deciso di non implementare tale ottimizzazione.

La memoria esterna riportata nella specifica fornita, così come nel test_bench_pubblico, è stata interpretata come una memoria asincrona.
Essa sarebbe pronta a rispondere esattamente al fronte alto del clock successivo alla richiesta, ma le è stato aggiunto un ritardo variabile di 1 o 2 ns.

```
architecture syn of rams_sp_wf is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM : ram_type;
begin
  process(clk)
    begin
    if clk'event and clk = '1' then
      if en = '1' then
        if we = '1' then
          RAM(conv_integer(addr)) <= di;
          do                      <= di after 2 ns;
        else
          do <= RAM(conv_integer(addr)) after 2 ns;
        end if;
      end if;
    end if;
  end process;
end syn;
```

Le parti evidenziate mostrano che il process reagisce al fronte alto del clock, ma mostra il risultato dopo 2ns (Perciò nelle FSM abbiamo 2 stati di attesa).
Immagino che tale ritardo dipenda dalla memoria che si intende utilizzare, perciò in generale si riceverà risposta in modo asincrono rispetto al clock.

Infine, osservando la funzione Open Elaborated Design di Vivado, sembra che il programma abbia inferito correttamente la struttura dei circuiti del componente che sono stati rappresentati nella sezione Architettura.