POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

PROVA FINALE

Progetto reti logiche

Cognome nome:	Gumus Furkan	Mandara Vitaliano
Codice Persona:	10666579	10690633
Matricola:	965612	913717

Anno accademico 2023/2024

Professore:Fabio Salice

INDICE

1. INTRODUZIONE

1.1 Scopo

Lo scopo del progetto consiste nella implementazione di un modulo hardware descritto in VHDL che sia in grado di interfacciarsi con una memoria esterna.

Presi in ingresso l'indirizzo ADD e il valore K il modulo ad ogni segnale di ingresso start=1 elabora una sequenza di K parole in memoria a partire dall'indirizzo ADD.

Le sequenza è composta nelle posizioni pari (ADD+0,ADD+2...) da parole con valori compresi tra 0 e 255 e nelle posizioni dispari(ADD+1,ADD+3...) da un valore chiamato credibilità riferito alla parola nella posizione precedente.

Il modulo sostituisce nella memoria le parole che hanno valore zero con l'ultimo valore letto diverso da zero e modifica le credibilità secondo le seguenti regole:

-la credibilità vale 31 ogni qual volta che la parola letta della sequenza è diverso da zero

-La credibilità viene decrementata rispetto al valore precedente ogni volta che si incontra una parola zero nella sequenza. Una volta che il valore di credibilità raggiunge lo zero, non viene ulteriormente ridotto.

1.2 Interfaccia

Il componente da descrivere 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_add   : in std_logic_vector(15 downto 0);
        i_k     : in std_logic_vector(9 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_data : out std_logic_vector(7 downto 0);
        o_mem_we   : out std_logic;
        o_mem_en   : out std_logic
    );
end project_reti_logiche;
```

Il componente ha due ingressi a un bit (i_start,I_rst),uno da 16 bit(I_add) e uno da 10 bit(i_K).

Ha inoltre un'uscita da un bit(o_done) che comunica la fine dell'elaborazione e viene posto a 1 soltanto dopo che i valori di credibilità sono stati aggiornati opportunamente.

Il primo segnale di reset detto si sistema serve ad inizializzare la macchina senza il quale non può ricevere il primo segnale di start.I successivi reset servono per re-inizializzare il modulo

1.3 Memoria

La memoria che utilizzerà il componente è una Single-Port Block RAM Write-First Mode.

Tale memoria è sincronizzata con il fronte di salita del clock ed essendo Write-first qualunque dato scritto è disponibile dal ciclo di clock successivo alla scrittura.

La lettura avviene ponendo a 1 l'uscita o_mem_en, a 0 o_mem_we e l'indirizzo da cui voglio leggere a o_mem_addr, il dato sarà disponibile quindi al ciclo di clock successivo all'ingresso i_mem_data del componente. In totale quindi ci vogliono almeno 2 cicli di clock.

Per la scrittura invece si imposta 1 sia o_mem_en che o_mem_we. Pongo l'indirizzo a cui voglio scrive su o_mem_addr e il dato da scrivere su o_mem_data. Per scrivere basta 1 ciclo di clock.

```
-- Single-Port Block RAM Write-First Mode (recommended template)
--
-- File: rams_02.vhd
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rams_sp_wf is
port(
  clk  : in  std_logic;
  we   : in  std_logic;
  en   : in  std_logic;
  addr : in  std_logic_vector(15 downto 0);
  di   : in  std_logic_vector(7 downto 0);
  do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;

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;
```

1.4 Esempi

L'esempio mostra il contenuto delle celle di memoria prima e dopo una elaborazione con una sequenza generica..

Indirizzo cella	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243
valore pre-elaborazione	128	0	64	0	0	0	100	0	2	0
valore post-elaborazione	128	31	64	31	64	30	100	31	2	31

In quest'altro esempio la sequenza inizia con degli zeri

Indirizzo cella	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243
valore pre-elaborazione	0	0	0	0	0	0	100	0	0	0
valore post-elaborazione	0	0	0	0	0	0	100	31	100	30

ARCHITETTURA

2.1 Funzionamento

Il modulo aspetta di ricevere il reset di sistema controllando l'ingresso i_rst. Una volta ricevuto il segnale imposta l'uscita DONE=0 e attende di ricevere il segnale i_start.

Al ricevimento dell'ingresso i_start=1 salva in reg3 i_add e legge la prima parola dalla memoria all'indirizzo salvato nel registro(se i_k>0 altrimenti termina).

La parola letta viene salvata sia nel registro reg2 che in reg4(guarda ottimizzazioni).

Se la prima parola letta è:

-diversa da zero(o_zero=0) incrementa l'indirizzo di memoria in reg3 e scrive 31 sia in memoria che nel registro reg1 che contiene l'ultima credibilità scritta in memoria.

-zero(o_zero=1) incrementa di due l'indirizzo in reg3 cosicché da puntare la prossima parola in memoria. Esegue lo stesso procedimento finchè non legge una parola diversa da zero

Per le parole successiva alla prima se leggo:

-una parola diversa da zero incremento l'indirizzo in reg3, scrivo 31 in reg1 e in memoria all'indirizzo in reg3 e salvo la parola in reg2 e reg4

-zero scrivo l'ultima parola diversa da zero salvata in reg4 in memoria all'indirizzo reg3, incremento reg3, decremento la credibilità contenuta in reg1 e scrivo in memoria reg1 con indirizzo reg3

Il modulo da l'uscita DONE=1 per 1 ciclo di clock quando l'uscita del registro reg3 è uguale ad i_add+(i_k + i_k)

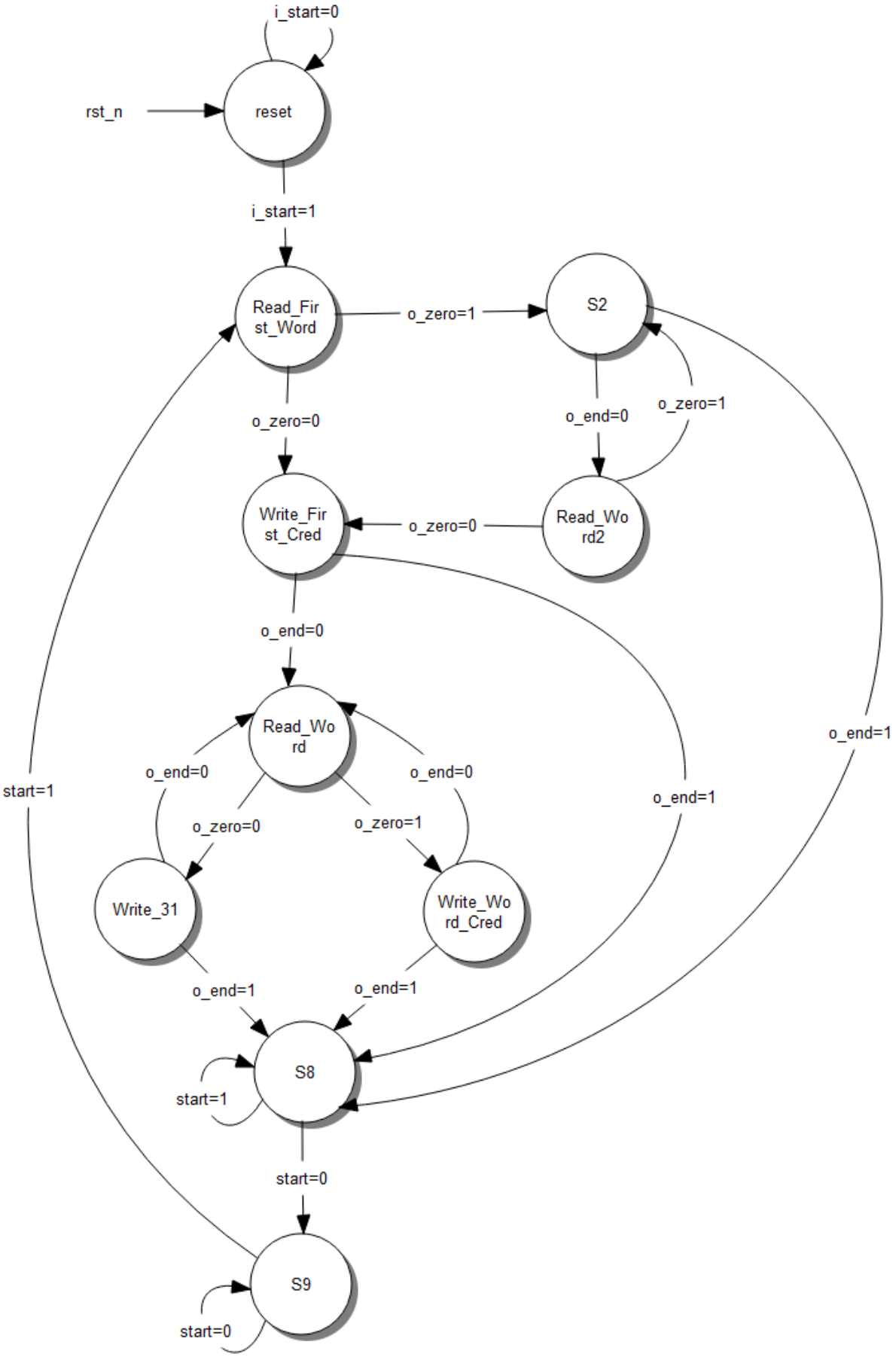
Se durante o alla fine di una esecuzione riceve un segnale di reset tutti i registri vengono resettati e aspetta di nuovo un nuovo ingresso start

Il componente è composto da un datapath e da una macchina a stati finiti che lo controlla.

2.2 ASF

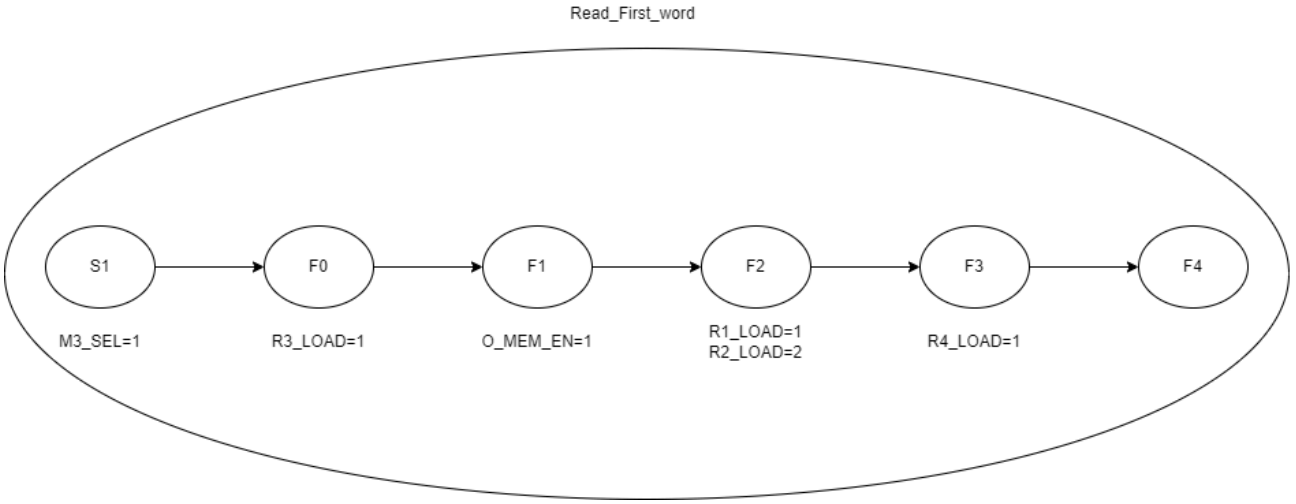
Per rendere più chiaro e non appesantire il disegno sono state adottate le seguenti considerazioni:

- Sono state omesse appositamente le frecce di reset che da qualsiasi stato nel caso venisse ricevuto un input i_rst=1 ritorna nello stato reset.
- Negli stati in cui non sono stati esplicitamente specificati i segnali di controllo vanno intesi come uguali a zero
- Siccome sono presenti 24 stati il disegno illustrato raggruppa più stati in macrostati. Ogni macrostato viene poi descritto più specificamente


i_start=0
rst_n
reset
i_start=1
Read_First_Word
o_zero=1
S2
o_zero=0
o_end=0
o_zero=1
Write_First_Cred
o_zero=0
Read_Word2
o_end=0
Read_Word
o_end=0
o_end=0
o_zero=0
o_zero=1
Write_31
Write_Word_Cred
o_end=1
o_end=1
start=1
o_end=1
o_end=1
S8
start=1
start=0
S9
start=0

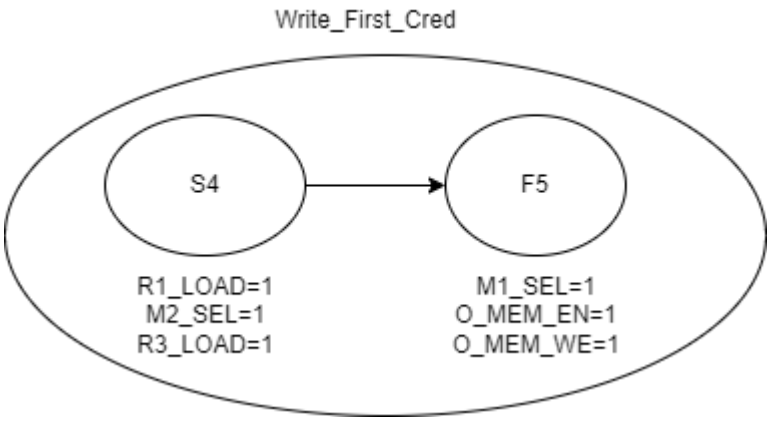
Read_First_Word

In questo macrostato leggo la prima parola all'indirizzo i_add preso in input



Write_First_Cred

Macrostato che si occupa di scrivere in memoria il valore della credibilità della prima parola diversa da zero



S2

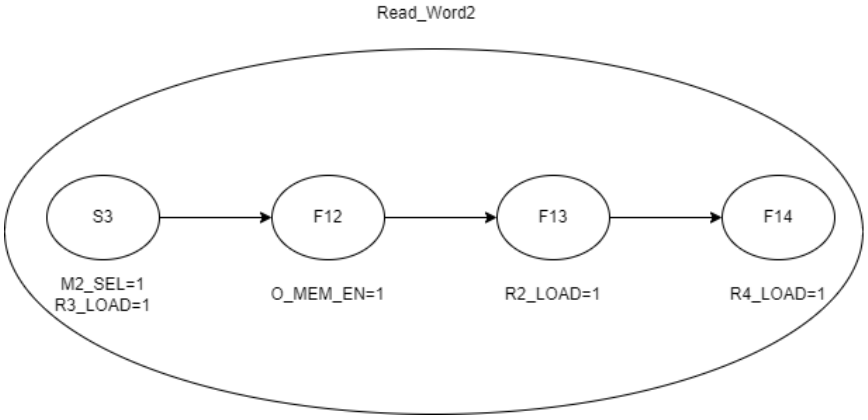
Incrementa l'indirizzo di memoria salvato in reg3 senza scrivere la credibilità(La credibilità in memoria quindi rimane 0)

M2_SEL=1

R3_load=1

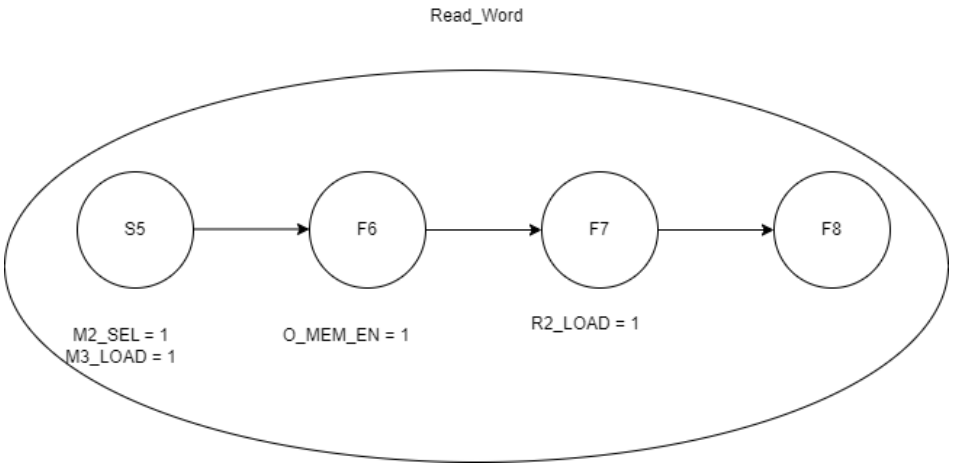
Read_Word2

Legge le parole successive alla prima, se sono zero torno in S2 e non ho bisogno di scrivere la credibilità in memoria ma mi basta incrementare reg3, se sono diverse da zero vado in S4



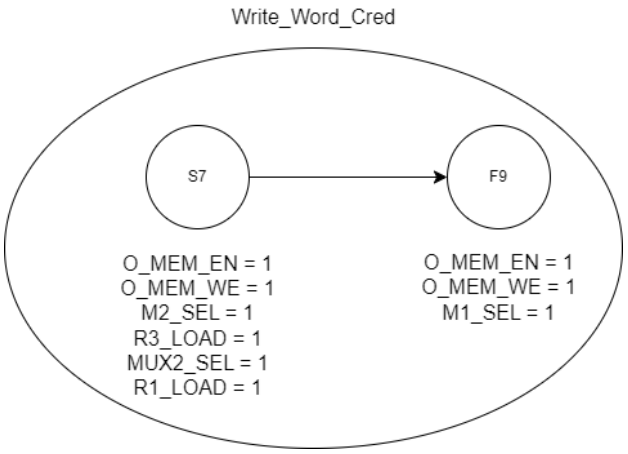
Read_Word

Legge parola dalla memoria con indirizzo salvato in reg3



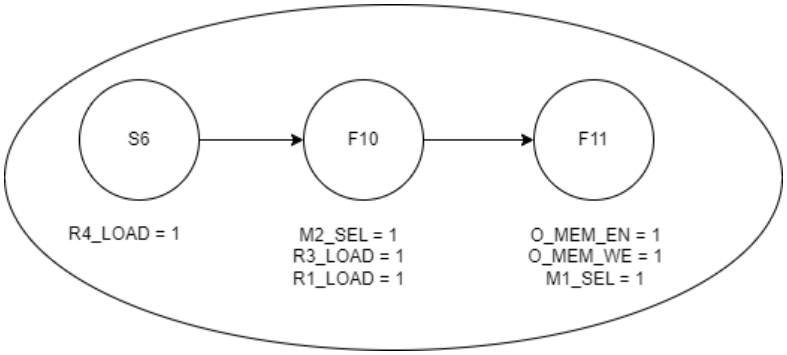
Write_Word_Cred

Se la parola letta è uno zero scrivo nell'indirizzo di memoria corrente l'ultima parola salvata in reg4,incremento indirizzo in reg4,Decremento Credibilità salvata in reg1, scrivo in memoria la credibilità



Write_31

Se la parola letta in Read_Word è diversa da 0 incremento l' indirizzo in reg3, scrivo 31 in reg1, scrivo la credibilità in reg1 in memoria all'indirizzo contenuto in reg3



S8

Lo stato S8 serve per aspettare che la sequenza sia finita e portare O_DONE uguale a 1 sino a un ciclo di clock dopo aver ricevuto i_start=0

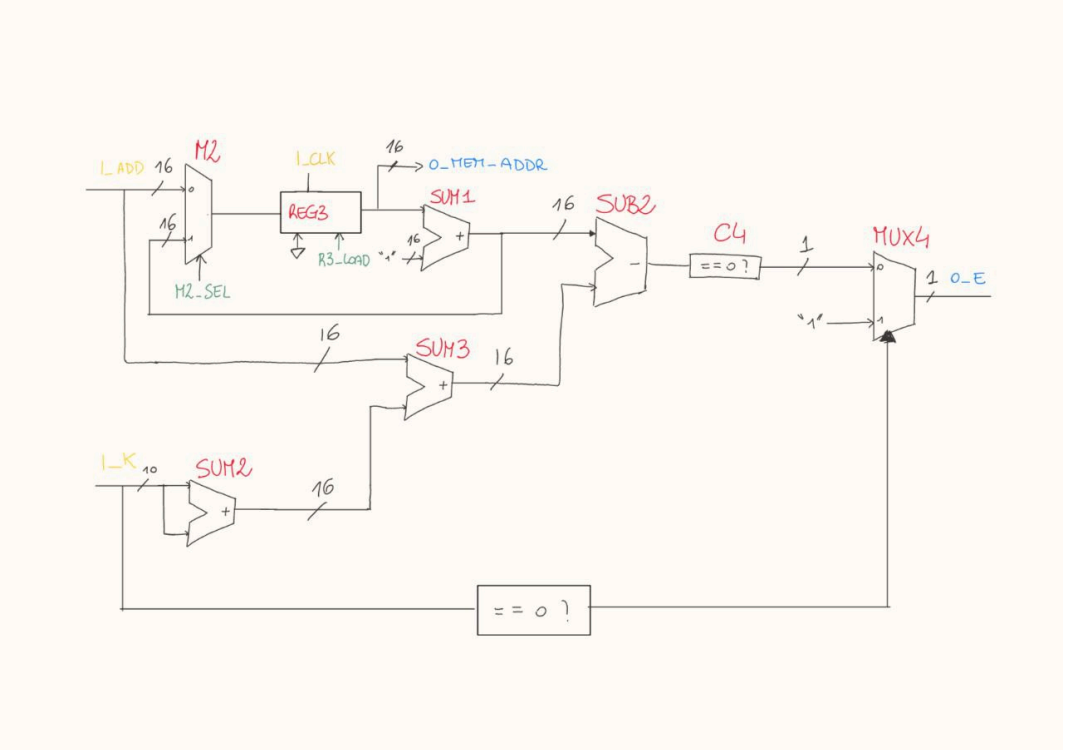
S9

Lo stato S9 serve per aspettare che il segnale i_start sia portato a 0 prima di ricevere una nuova sequenza e quindi un nuovo segnale i_start uguale a 1

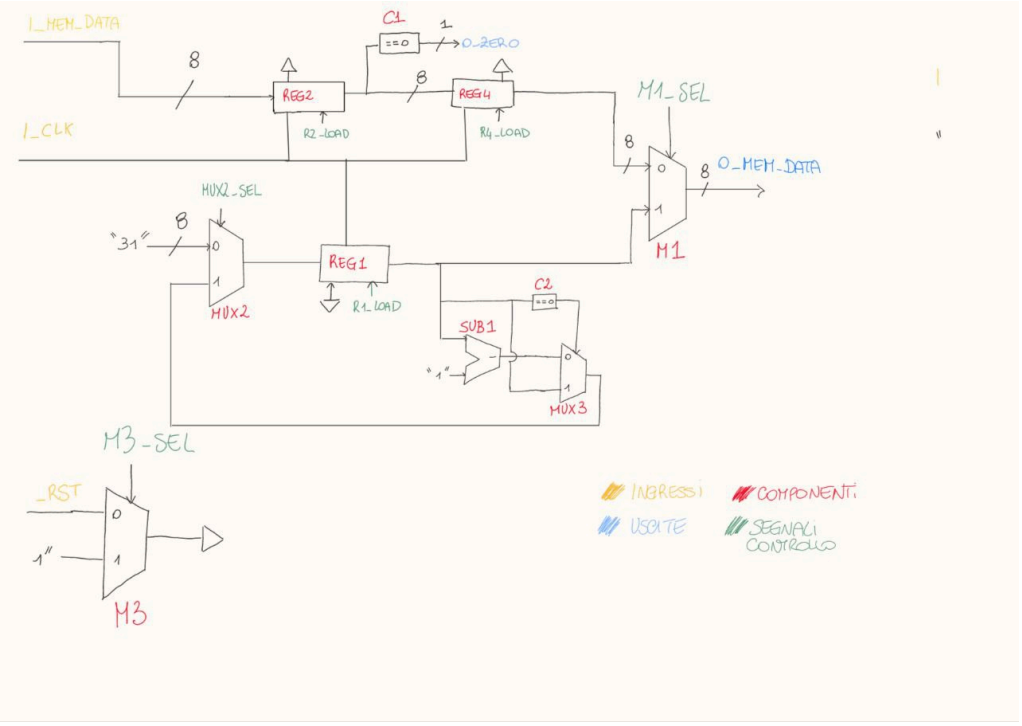
2.3 Datapath

Il datapath è composto da 2 sottomoduli:

-modulo calcolo indirizzo di lettura e scrittura in memoria



-modulo salvataggio parole lette e calcolo/salvataggio delle credibilità



C1,C2,C4 sono comparatori le cui uscite valgono 1 se il dato in ingresso è zero

reg1,reg2,reg3,reg4 sono registri a n bit composti da flip flop che commutano sul fronte di salita del clock

2.4 segnali

l'asf invia al datapath i seguenti segnali:

1. o_mem_en per abilitare la lettura dalla memoria
2. o_mem_we per abilitare la scrittura in memoria
3. r*_load per salvare nel reg* il dato passato
4. m1_sel per far scegliere al multiplexer m1 se il dato da propagare è la parola o la credibilità
5. m2_sel per far scegliere al multiplexer m2 se il dato da propagare è l' indirizzo iniziale i_add o quello calcolato
6. m3_sel per suddividere i casi di quando inizio a leggere una nuova sequenza (m3_sel = 1), un reset manuale (i_rst = 1 e m3_sel = 0) oppure sono in uno stato qualsiasi e non ho nessun reset (i_rst =0, m3_sel = 0)
7. mux2_sel per far scegliere al multiplexer mux2 se la credibilità da propagare è 31 oppure quella calcolata in base al numero di valori non validi letti

Se o_end = 1(ricevuto dal datapath) invia il segnale di uscita o_done=1 che indica la fine dell'elaborazione

Il datapath ha i seguenti segnali:

1. o_zero per indicare quando ha letto un valore che è uguale a zero
2. o_end per indicare quando ha finito una sequenza

2.5 Ottimizzazioni

La prima ottimizzazione a cui abbiamo pensato è stata quella di salvare (nel Reg4) l'ultimo valore diverso da zero letto. In questo modo quando si legge una parola non valida non c'è bisogno di rileggere l'ultima parola valida diversa da zero.

3 Risultati sperimentali

3.1 Sintesi

3.1.1 Report utilization

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	79	0	0	134600	0.06
LUT as Logic	79	0	0	134600	0.06
LUT as Memory	0	0	0	46200	0.00
Slice Registers	45	0	0	269200	0.02
Register as Flip Flop	45	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	4	0	0	67300	<0.01
F8 Muxes	0	0	0	33650	0.00

I componenti utilizzati dal tool di sintesi sono gli stessi che avevamo previsto nel datapath e fsm infatti:

3 registri da 8 bit(reg1,reg2,reg4) : 24 +

1 registro da 16 bit(reg3): 16 +

log2(25) : 5

=45 flip flops

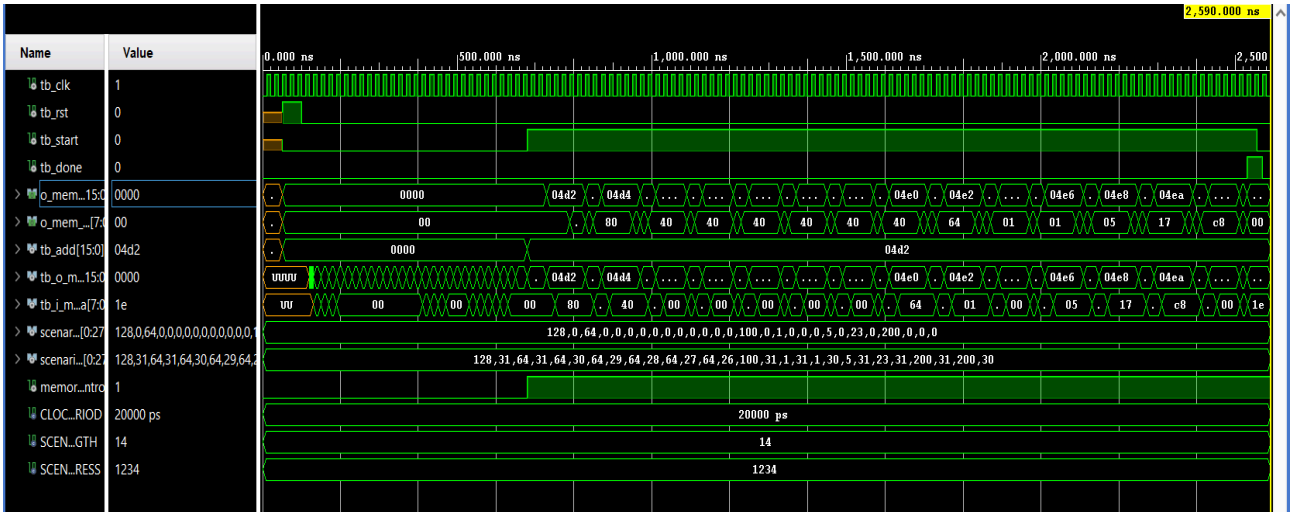
3.1.2 Report Timing

Il componente ha uno Slack Time(MET) di 13.63ns e un Data Path Delay di 6.213ns quindi rispetta ampiamente i requisiti di tempo previsti da specifica.

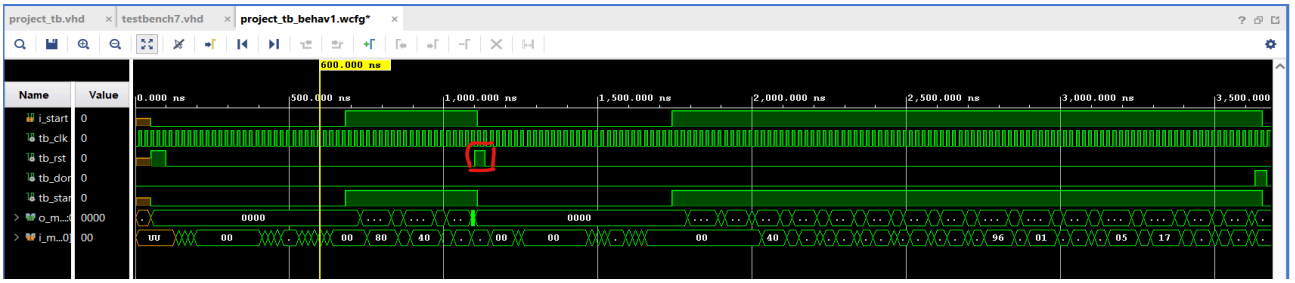
3.2 simulazioni

Per testare l'effettiva correttezza del componente sintetizzato con Vivado abbiamo creato e usato vari testbench. Di seguito elenchiamo quelli più significativi:

1. Testbench di esempio fornito dai docenti. La stringa elaborata è identica a quella fornita dal testbench. I segnali di start e di done si comportano come da specifica.

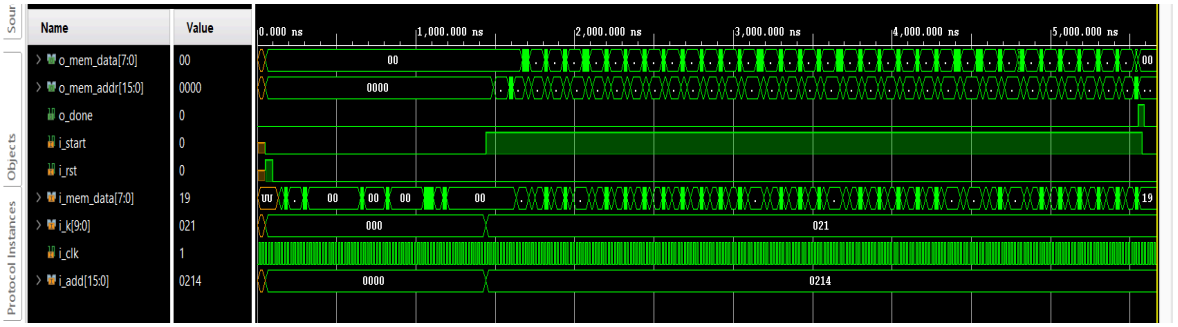


2. Reset durante una elaborazione.Se arriva un segnale di reset durante una elaborazione il componente viene reinizializzato e aspetta un nuovo start per partire con una nuova elaborazione. Il segnale di done si alza solo al termine della seconda elaborazione poiché avvenuta correttamente.

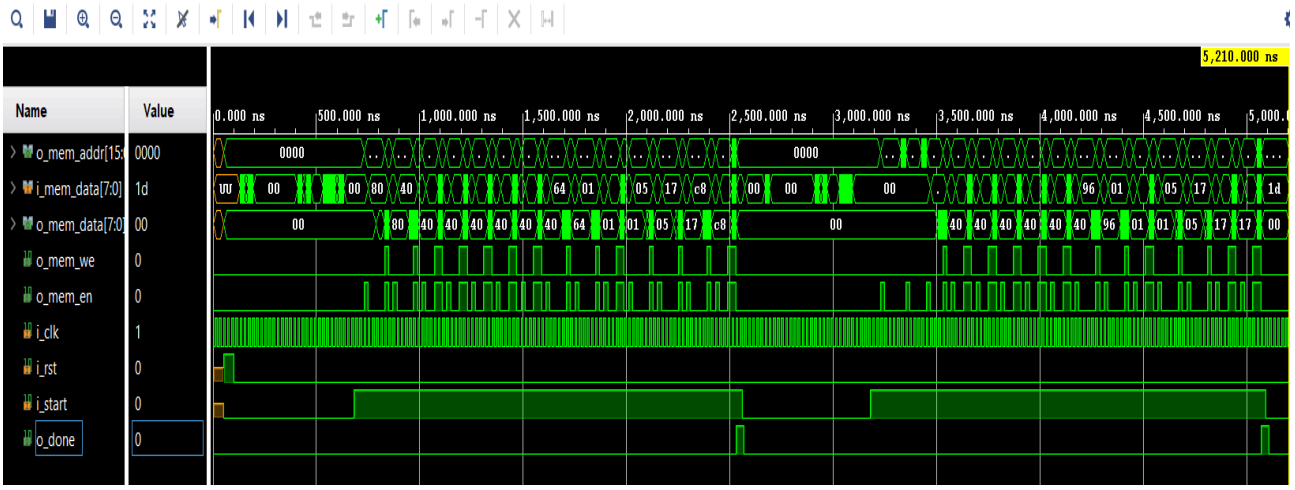


Vanno sicuramente menzionati anche gli altri casi particolari che abbiamo testato per verificare la correttezza del componente:

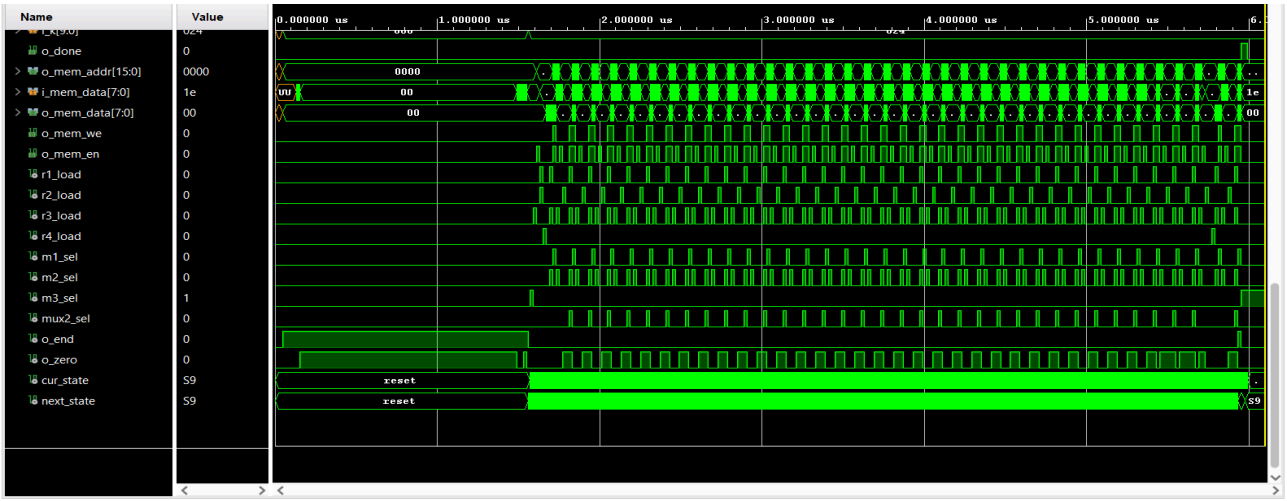
3. Verifica caso in cui inizi con 0, in figura mostriamo solo che il nostro componente si comporta correttamente in questo caso iniziale. Per completezza però diciamo che anche i valori che seguono sono inseriti correttamente.



4. Verifica se il componente elabora correttamente più sequenze consecutive



5. verifica se dopo aver letto più di trentuno volte zero continua a stampare zero



4 CONCLUSIONI

La macchina a stati finiti e il datapath sono stati progettati tenendo in considerazione l'efficienza temporale infatti ha uno slack time di 13 ns(inferiore quindi al requisito richiesto da specifica).

Il componente supera correttamente, in tutti i casi generici e corner case creati da noi, la behavioral simulation e la post-synthesis senza la presenza di latch o di errori.

Riteniamo quindi di avere creato un componente hardware in grado di soddisfare tutte le specifiche richieste.