POLITECNICO
MILANO 1863

Progetto di Reti logiche Prof William Fornaciari - Anno 2022/2023

Alessandro Anziutti:

(Codice Persona: 10764167 - Matricola: 959863)

Francesco Andorlini:

(Codice Persona: 10807972 - Matricola: 984752)

Sommario

INTRODUZIONE

Abbiamo realizzato un componente HW attraverso il linguaggio VHDL che permetta, ricevendo segnali in ingresso, di modificare la memoria RAM in modo da aggiungere credibilità ad una serie di valori letti.

INTERFACCIA

Come interfaccia abbiamo diverse porte:

```
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
);
```

segnali in ingresso

a. **i_clk**: ingresso del clock, che temporizza l'esecuzione del componente
b. **i_rst**: segnale di reset, che comanda la reimpostazione del componente e di tutti i suoi segnali per permettere di partire con una nuova sequenza
c. **i_start**: segnale che permette di partire con l'esecuzione delle computazioni quando ricevuto ad alto
d. **i_add**: indirizzo di memoria RAM di partenza dove si trova il primo valore da controllare
e. **i_k**: numero di valori presenti nella sequenza e di conseguenza il doppio di k rappresenta la quantità di bit coperti in RAM dalla sequenza
f. **i_mem_data**: segnale di ritorno dalla RAM dopo che viene eseguita una lettura/scrittura; dopo una lettura viene ritornato il valore richiesto, dopo una scrittura il valore scritto nella memoria

segnali in uscita

g. **o_done**: notifica la fine della computazione e la possibilità di ripartire tramite un segnali di reset alto

h. **o_mem_addr**: segnale con cui possiamo passare l'indirizzo alla RAM in cui vogliamo eseguire una lettura/scrittura di quella cella

i. **o_mem_data**: segnale con cui passiamo il valore da scrivere in RAM nel caso in cui we sia alto, viene ignorato nel caso accedessimo in lettura

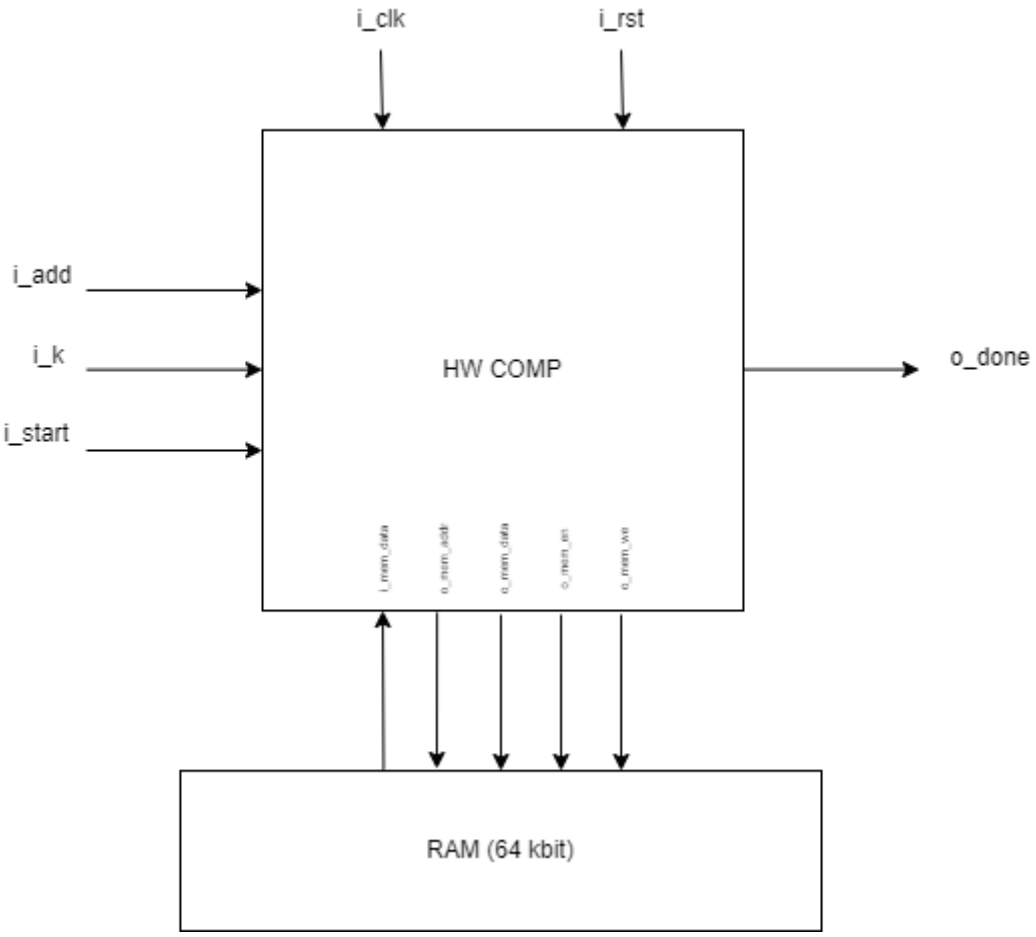
j. **o_mem_en**: segnale che indica se intendiamo accedere alla RAM

k. **o_mem_we**: segnale che indica se accediamo alla RAM in lettura o in scrittura

combinazione per la lettura en = 1, we = 0

combinazione per la scrittura en = 1, we = 1

HOW IT WORKS



Schema di Input/Output

All’istante iniziale abbiamo che i_start è a 0.

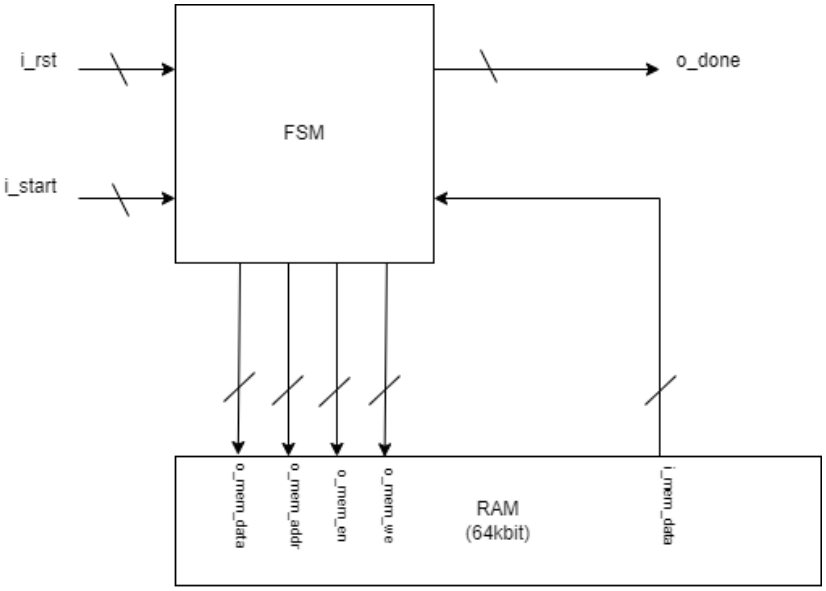
Appena i_rst sale a 1, o_done, o_mem_addr, o_mem_data, o_mem_we e o_mem_en vengono inizializzati a 0.

Una volta che o_done è a 0, i_rst si abbassa a 0 e i_start sale a 1, inizia l’esecuzione del componente che, attraversando diversi stati, interpella la RAM per la lettura e la scrittura.

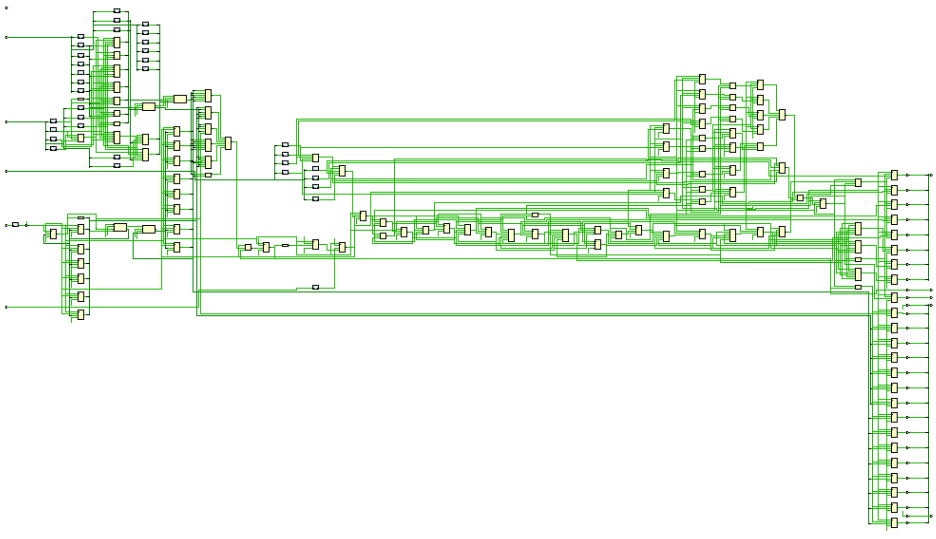
L’esecuzione continua finchè, attraverso un controllo interno che compara la mia posizione attuale con la posizione del k-esimo valore letto, o_done verrà posto a 1 e terminerà, con il conseguente abbassamento di o_done e di i_start a 0.

Da qui il componente rimane silente finchè non verrà di nuovo posto i_rst a 1 e ritornerà in stato di “pronto a cominciare”.

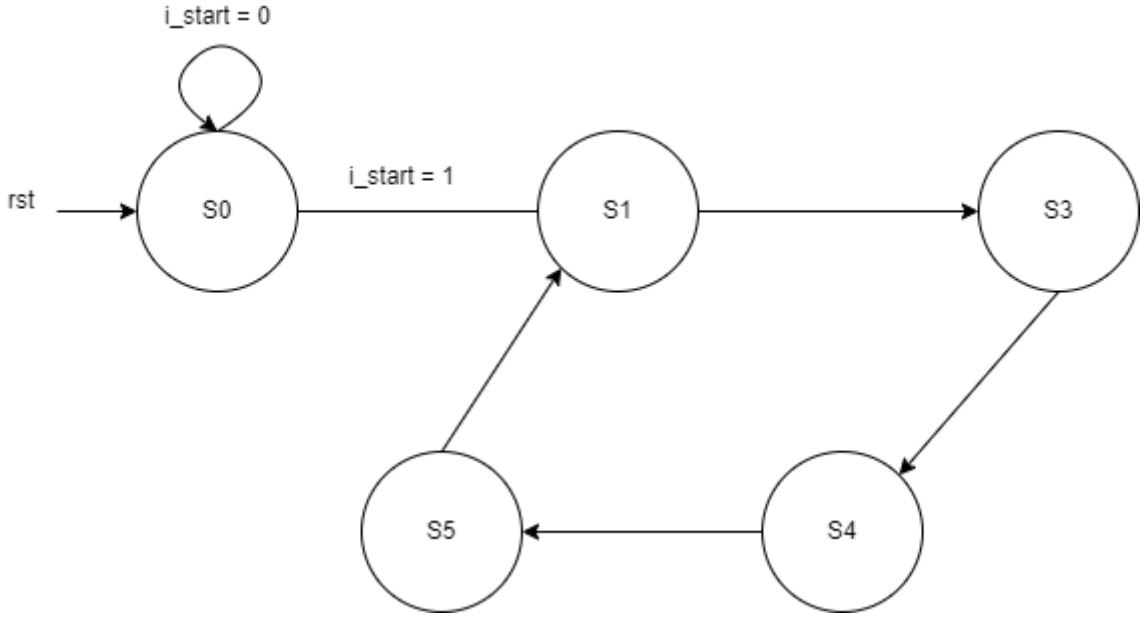
ARCHITECTURE



Questa è l’architettura del nostro HW, formata solo da un FSM



FSM



Gli stati si suddividono in:

- **S0**: stato di reset, ha comportamenti diversi in base al valore logico del segnale i_rst

o_mem_addr	**o_mem_data**	**o_mem_en**	**o_mem_we**	**o_done**	**temp_done**
U/00	U/00	U/0	U/0	U/0/1	U/0/1

in questo stato inoltre inizializzo tutti i segnali interni a 0.

Nel caso in cui temp_done sia uguale a 1, e quindi ho impostato precedentemente il valore di o_done a 1, viene settato o_done a 0 di modo da essere pronto per una nuova esecuzione.

- **S1**: stato di lettura del valore della parola

o_mem_addr	**o_mem_data**	**o_mem_en**	**o_mem_we**	**o_done**
curr_addr	00	1	0	0

curr_addr è un segnale interno al componente che permette di mantenere un puntatore all'indirizzo di memoria del valore che dobbiamo leggere, inizialmente inizializzato in S0 al valore di i_add

- **S2**: stato di scelta di percorso da intraprendere, scelto in base al valore letto dalla memoria nello stato prima e ricevuto tramite i_mem_data

o_mem_addr	**o_mem_data**	**o_mem_en**	**o_mem_we**	**o_done**	**path**	**curr_val**
curr_addr	00	0	0	0	01/10/11	i_mem_addr

dove path è appunto il segnale che indica il nome del percorso:

- 01: quando il primo valore letto è uno 0, allora dovremo andare a scrivere uno 0 in credibilità, e così finchè non si arriverà a leggere un valore diverso da 0

f.e.:

input: 0 0 0 0 0 0 8 0 ...

output: 0 0 0 0 0 0 8 31 ...

- 10: quando leggo uno 0 ma ho precedentemente letto un valore diverso da 0, allora dovrò scrivere al posto dello 0 letto il valore precedente e come credibilità la credibilità precedente diminuita di 1
- 11: quando leggo un valore diverso da 0

quindi il path 01 sarà percorribile solo nel caso in cui non ho ancora letto un valore diverso da 0, appena se ne leggerà uno, potremo intraprendere solo i path 10 e 11.

curr_val è un segnale che permette di mantenere il valore letto nel caso in cui il successivo valore letto sia uno 0.

- **S3**: stato di scrittura della credibilità

o_mem_addr	**o_mem_data**	**o_mem_en**	**o_mem_we**
curr_addr + 1	00/prec_cred-1/1f	1	1
o_done	**path**	**prec_cred**	
0	01/10/11	00/prec_cred-1/1f	

in base al path intrapreso andremo a passare alla RAM come o_mem_data uno tra i seguenti valori

- 01 → 00
- 10 → prec_cred - 1
- 11 → 1f (31 in decimale)

dove prec_cred è un segnale che mantiene il valore di credibilità precedente e viene usato nel caso in cui si verifichi il path 10.

- **S4**: stato di scrittura del valore

o_mem_addr	**o_mem_data**	**o_mem_en**	**o_mem_we**	**o_done**	**path**
curr_addr	prec_val	1	1	0	11

in questo stato, che nel caso di path uguale a 01 o 10 sarà bypassato, andremo a passare alla RAM il valore con cui rimpiazzare lo 0.

prec_val è un segnale che mantiene il valore precedente della parola nel caso in cui si legga uno 0 dopo aver letto un valore diverso da 0.

- **S5**: stato di update e control dei segnali

o_mem_addr	**o_mem_data**	**o_mem_en**	**o_mem_we**
curr_addr + 2	prec_val	0	0
o_done	**path**	**prec_val**	**temp_done**
0/1	01/10/11	00/curr_val/prec_val	0/1

In questo stato vado a:

- aggiornare il valore di prec_val per l'esecuzione successiva in base al path calcolato nello stato S2
- controllare se il curr_addr sia la posizione dell'ultimo valore ponendo successivamente, in caso affermativo, o_done e temp_done a 1. (temp_done è un segnale utilizzato nel controllo in S0, dato che non posso conoscere a posteriori i valori delle uscite)
- aggiornare il curr_addr sommandogli 2. (che in questo caso equivale a sommare due valori esadecimali e quindi navigare in avanti di 2 byte cioè 16 bit)

SEGNALI INTERNI AL FSM

```
type S is (S0, S1, S2, S3, S4, S5);
signal curr_state, next_state : S;

signal done_con: std_logic;
signal subbed: std_logic;
signal temp_done: std_logic;
signal curr_addr : std_logic_vector(15 downto 0);
signal path : std_logic_vector(1 downto 0);
signal curr_val : std_logic_vector(7 downto 0);
signal curr_cred : std_logic_vector(7 downto 0);
signal prec_val : std_logic_vector(7 downto 0);
signal prec_cred : std_logic_vector(7 downto 0);
```

Dove:

- **S**: è l'enumerazione degli stati
- **curr_state**: indica qual è lo stato attuale del FSM
- **next_state**: indica quale sarà lo stato in cui passeremo alla fine dell'esecuzione dello stato corrente
- **done_con, subbed**: sono segnali "sentinella" che partono abbassati rispettivamente nel controllo per alzare l'o_done e nel caso vada a sommare/sottrarre un segnale e vengono alzati una volta compiute le azioni; servono ad evitare un possibile doppio controllo/somma se uno stato passa per due fronti di salita del clock
- **temp_done**: serve a controllare il valore dell'o_done, dato che quest'ultimo essendo un uscita, non se ne può conoscere il valore a posteriori
- **curr_addr**: segnale che viene inizializzati a i_add appena i_start sale a 1 e mantiene il valore dell'indirizzo progressivo in cui andiamo a leggere/scrivere i valori; viene interpellato anche nel controllo dell'o_done essendo comparato all'indirizzo del k-esimo valore
- **path**: segnale che indica il "percorso" da intraprendere, come spiegato precedentemente (qui)
- **curr_val**: segnale che mantiene il valore letto nello stato S1, essendo un ingresso manipolabile nel tempo da altri stati, e viene utilizzato nel controllo in S2 per scegliere il valore di path
- **curr_cred**: segnale che mantiene il valore della credibilità e che viene aggiornato in base al path
- **prec_cred**: segnale che contiene il valore precedente della credibilità, viene usato per la scrittura in caso di lettura di 0 dopo una parola diversa da 0; viene aggiornato in base al path
- **prec_val**: segnale che mantiene il valore della parola precedente e serve nella scrittura del valore se la parola letta è 0 dopo aver letto un valore diverso da 0; viene aggiornato in base al path

RISULTATI EMPIRICI DELL'HW

Site Type	Used	Fixed	Prohibited	Available	Util%
CLB LUTs*	45	0	0	70560	0.06
LUT as Logic	45	0	0	70560	0.06
LUT as Memory	0	0	0	28800	0.00
CLB Registers	68	0	0	141120	0.05
Register as Flip Flop	68	0	0	141120	0.05
Register as Latch	0	0	0	141120	0.00
CARRY8	4	0	0	8820	0.05
F7 Muxes	0	0	0	35280	0.00
F8 Muxes	0	0	0	17640	0.00
F9 Muxes	0	0	0	8820	0.00

```
------------------------------------------------------------------------------------
Slack (MET) :             18.399ns  (required time - arrival time)
  Source:                 prec_cred_reg[1]/C
                            (rising edge-triggered cell FDRE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Destination:            o_mem_data[0]/R
                            (rising edge-triggered cell FDRE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns - clock rise@0.000ns)
  Data Path Delay:        1.413ns  (logic 0.499ns (35.318%)  route 0.914ns (64.682%))
  Logic Levels:           3  (LUT4=2 LUT5=1)
  Clock Path Skew:        -0.034ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    1.497ns = ( 21.497 - 20.000 )
    Source Clock Delay      (SCD):    1.965ns
    Clock Pessimism Removal (CPR):    0.433ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
  Clock Net Delay (Source):      1.006ns (routing 0.001ns, distribution 1.005ns)
  Clock Net Delay (Destination): 0.897ns (routing 0.001ns, distribution 0.896ns)
```

Possiamo notare che il componente non genera latch e rientra perfettamente nei limiti di tempo.

TESTBENCH

Tramite l'utilizzo di testbench abbiamo coperto le seguenti situazioni:

- numero elevato di parole da leggere
- multi-start (multiple letture consecutive)
- reset durante lettura/scrittura
- start alto durante il reset
- k = 0, sequenza vuota
- sequenza di soli 0
- sequenza di lettura di > 31 zeri che hanno portato il successivo azzeramento della credibilità nelle letture successive alla 31esima

Siamo riusciti ad avere una copertura totale di tutte le possibili combinazioni che ci hanno portato a superare tutti i testbenches sia in modalità behavioral, sia in modalità post-synthesis.