POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

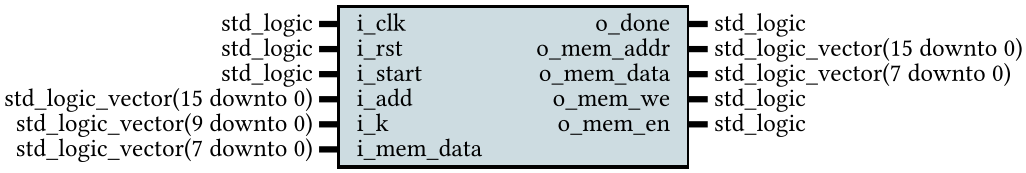
Prova finale: Reti Logiche 2023/2024

Angelo Prete
angelo2.prete@mail.polimi.it
10767149

1. Introduzione

Il componente realizzato si occupa di elaborare una sequenza in memoria andando a sostituire alle celle con valori pari a 0, che possono essere interpretati come valori mancanti, l'ultimo valore valido la cui credibilità viene decrementata man mano. Il componente potrebbe essere utilizzato, ad esempio, per correggere letture assenti di sensori, che spesso indicano valori assenti con 0.

Il componente presenta i seguenti ingressi e uscite



descritti con più dettaglio nei paragrafi successivi.

a. Collegamento a memoria RAM

Il componente deve essere collegato a una memoria RAM che rispetta la seguente interfaccia

```
entity ram 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 ram;
```

In particolare, deve essere collegata al componente realizzato come in tabella

Segnale	RAM	Componente	Dimensione

Enable	en	o_mem_en	1 bit
Write enable	we	o_mem_we	1 bit
Address	addr	o_mem_addr	16 bits
Data in	di	i_mem_data	8 bits
Data out	do	o_mem_data	8 bits

e RAM e componente stesso devono condividere il segnale di clock.

b. Descrizione funzionamento

Funzionamento modulo:

1. Vengono forniti in input l'indirizzo di partenza, il numero di coppie (divise in valore e credibilità) di celle da processare e un segnale di start (dopo l'eventuale segnale di reset)
2. Il modulo inizia a processare i dati in memoria, come descritto nel seguente pseudocodice

input: indirizzo di partenza a, numero di iterazioni k
$a_f \leftarrow a + 2 * k$ (indirizzo finale da processare più uno)
$d_l \leftarrow 0$ (ultimo dato letto diverso da 0)
$c_l \leftarrow 0$ (ultima credibilità)
RAM (memoria RAM rappresentata come un vettore)
while $a \neq a_f$ **do**
  $d \leftarrow \text{RAM}[a]$
  if $d \neq 0$ **then**
    $d_l \leftarrow d$
    $\text{RAM}[a + 1] \leftarrow 31$
    $c_l \leftarrow 31$
  else
    $\text{RAM}[a] \leftarrow d_l$
    $c_l \leftarrow \max((c_l - 1), 0)$
    $\text{RAM}[a + 1] \leftarrow c_l$
  end
  $a \leftarrow a + 2$
end

3. Finita l'operazione, il componente lo segnala ponendo **o_done** alto e aspetta che venga portato basso il segnale **o_start**.

c. Esempio funzionamento

2. Architettura

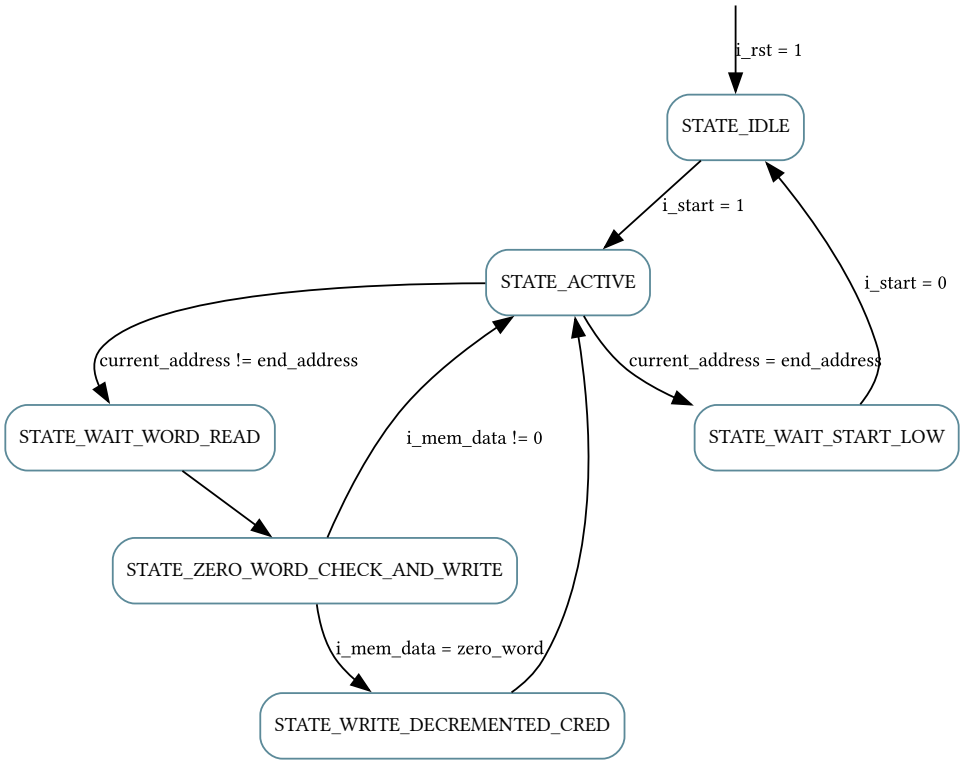
Data la semplicità del componente, non si è ritenuto necessario dividerlo in più entities. Il risultato finale è una singola entity, la cui architettura realizza una macchina a stati tramite due processi.

a. Macchina a stati finiti (entity project_reti_logiche)

La macchina a stati finiti dell'architecture del componente è una macchina di Mealy. Internamente, le transizioni della FSM sono sul fronte di salita del clock.

È composta da 6 stati, sono quindi necessari 3 flip flop per memorizzare lo stato corrente. Ogni stato ha uno specifico compito:

- **STATE_IDLE**: Quando la FSM è in attesa di iniziare una nuova computazione, si trova in questo stato. È possibile arrivare qui a seguito del reset asincrono o della fine di una computazione.
- **STATE_ACTIVE**: In questo stato la FSM decide se processare una nuova coppia di indirizzi di memoria, a seguito di un controllo sull'indirizzo da processare, oppure teminare la computazione. Se l'indirizzo da processare non è l'ultimo, si preparano i segnali di memoria per leggere il dato all'indirizzo corrente.
- **STATE_WAIT_START_LOW**: Arriviamo in questo stato quando gli indirizzi da processare sono finiti, la macchina segnala questo ponendo il segnale o_done alto e aspetta che i_start venga abbassato a 0, evento seguito dal ritorno nello stato di idle.
- **STATE_WAIT_WORD_READ**: Questo stato serve per permettere alla memoria di fornire il dato richiesto negli stati precedenti; infatti, da specifica, la memoria ha un delay di 2 nanosecondi solo al termine dei quali può fornire il dato richiesto.
- **STATE_ZERO_WORD_CHECK_AND_WRITE**: Il dato è finalmente disponibile: se è uguale a 0 bisogna sovrascriverlo (comunicandolo opportunamente alla RAM) con l'ultimo dato diverso da 0 e spostarsi nello stato di scrittura della credibilità decrementata, altrimenti, scriviamo nell'indirizzo successivo in RAM il massimo valore di credibilità (31) e torniamo nello stato active.
- **STATE_WRITE_DECREMENTED_CRED**: Siamo in questo stato se abbiamo letto un valore pari a 0 in memoria. Scriviamo in memoria quindi un valore di credibilità decrementato rispetto al precedente (o 0 se l'ultima credibilità era già pari a 0 stesso).



i. Processo 1: Clock e reset asincrono

ii. Processo 2: Scelta stati e scritture in memoria

b. Modulo 2

3. Risultati sperimentali

a. Sintesi

A seguito del processo di sintesi (con target **xa7a12tcpg238-2I**), otteniamo i seguenti dati:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	78	0	134600	0.06
LUT as Logic	78	0	134600	0.06
LUT as Memory	0	0	46200	0.00
Slice Registers	51	0	269200	0.02
Register as Flip Flop	51	0	269200	0.02
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Notiamo che il componente usa:

- **51 flip flop** (0.02%), tutti e soli i previsti
- **78 look-up tables** (0.06%)
- **0 latches**, risultato ottenuto grazie ad opportune scelte progettuali

La percentuale di occupazione degli elementi disponibili è molto bassa: la logica implementata è molto semplice e non necessita di ampi spazi di memoria o complesse operazioni.

b. Simulazioni

Il componente è stato sottoposto sia testbeches scritti a mano per verificare il suo comportamento in edge-cases, sia a testbeches generati casualmente per verificare il corretto funzionamento su vari range di memoria.

i. Testbench ufficiale

Il primo testbench ad essere stato provato è quello presente nei materiali per il progetto, funziona correttamente e rispetta i vincoli di clock.

ii. Start multipli

Questo testbench è stato scritto per verificare il corretto funzionamento del componente a seguito di più esecuzioni senza reset intermedi.

iii. Reset durante l'esecuzione

Grazie a questo test si è dimostrato il funzionamento del componente quando il segnale di reset asincrono viene portato a 1 durante l'esecuzione.

4. Conclusioni

Il componente, oltre a rispettare la specifica, è stato implementato in modo efficiente. È stata posta infatti particolare attenzione ad utilizzare un nm