Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Progetto Finale di Reti Logiche

Riccardo Paltrinieri
Matricola: 10626923
Professore: Gianluca Palermo

01/04/2020

Indice

1. Descrizione

La specifica del Progetto di Reti Logiche (Prova finale) 2019/2020 è ispirata al metodo di codifica a bassa dissipazione di potenza denominato "Working Zone"[1].
Il metodo di codifica Working Zone è un metodo pensato per il Bus Indirizzi che si usa per trasformare il valore di un indirizzo quando questo viene trasmesso, se appartiene a certi intervalli (detti appunto working-zone). In questo caso il componente progettato invia al Bus solo l'identificativo della working-zone a cui appartiene e il valore dell'offset codificato come one-hot.

1.1 Specifiche di progetto

Nella versione da implementare il numero di bit da considerare per l'indirizzo da codificare è 7. Il che definisce come indirizzi validi quelli da 0 a 127. Il numero di working-zone è 8 (Nwz=8) mentre la dimensione della working-zone è 4 indirizzi incluso quello base (Dwz=4).
Questo comporta che l'indirizzo codificato sarà composto da 8 bit: 1 bit per WZ_BIT + 7 bit per ADDR, oppure 1 bit per WZ_BIT, 3 bit per codificare in binario a quale tra le 8 working zone l'indirizzo appartiene, e 4 bit per codificare one hot il valore dell'offset di ADDR rispetto all'indirizzo base.
Il modulo da implementare leggerà l'indirizzo da codificare e gli 8 indirizzi base delle working-zone e dovrà produrre l'indirizzo opportunamente codificato.

1.2 Specifica dei componenti

1.2.1 Componente da progettare

Il componente descritto ha la seguente interfaccia:

```
entity project_reti_logiche is
    port (
        i_clk       : in std_logic;
        i_start     : in std_logic;
        i_rst       : in std_logic;
        i_data      : in std_logic_vector(7 downto 0);
        o_address   : out std_logic_vector(15 downto 0);
        o_done      : out std_logic;
        o_en        : out std_logic;
        o_we        : out std_logic;
        o_data      : out std_logic_vector (7 downto 0)
    );
end project_reti_logiche;
```

1.2.2 Memoria RAM

La memoria e il suo protocollo sono descritti nel seguente modo seguendo una specifica derivata dalla User Guide di Vivado:

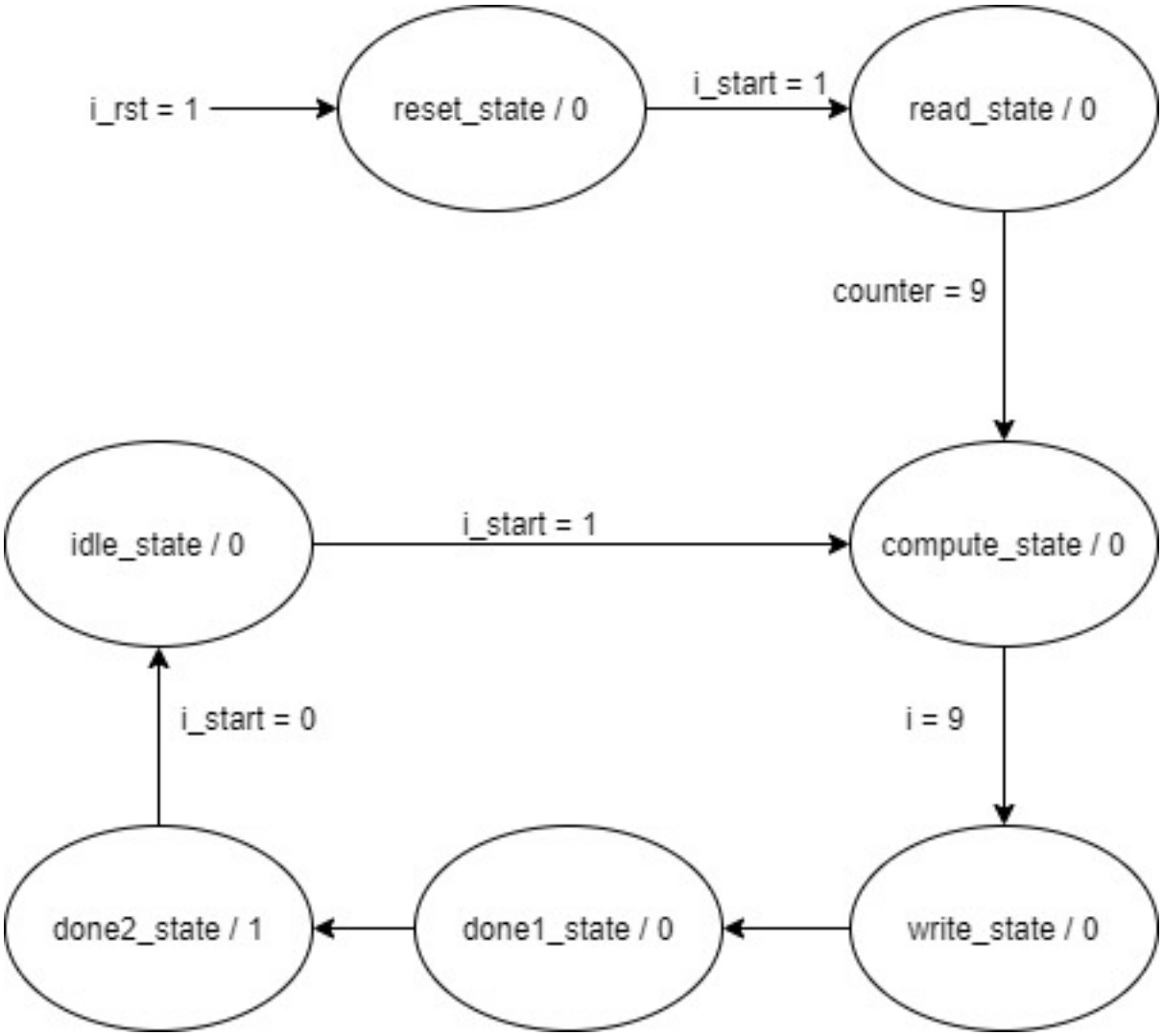
```
-- Single-Port Block RAM Write-First Mode (recommended template)
--
-- File: rams_02.vhd
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rams_sp_wf is
port(
  clk  : in  std_logic;
  we   : in  std_logic;
  en   : in  std_logic;
  addr : in  std_logic_vector(15 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;
        else
          do <= RAM(conv_integer(addr));
        end if;
      end if;
    end if;
  end process;
end syn;
```

2. Scelte Progettuali

2.1 Macchina a Stati Finiti

Per l'implementazione ho deciso di utilizzare una Macchina a Stati Finiti (FSM), in particolare una macchina di Moore con l'uscita che corrisponde al segnale o_done:



Lo stato "write" e il primo stato di "done" permangono per un solo ciclo di clock poi passano allo stato successivo senza alcuna condizione in quanto non eseguono alcuna operazione e non aspettano alcun segnale in input ma ne modificano solo alcuni interni o in uscita.

2.1.1 Reset state

Si tratta dello stato di partenza della macchina ma viene raggiunto ogni volta che il segnale i_rst viene posto a '1' (il controllo è implicito in ogni stato). Tutte le uscite vengono impostate al proprio valore di default, dopodichè la

macchina rimane in attesa del segnale di start.

2.1.2 Read state

In questo stato viene attivata la memoria del testbench impostando il valore di o_en=1 e viene attivato un ciclo attraverso un contatore a 4 bit e un case statement per leggere dalla memoria un valore per ogni periodo di clock. Questo valore viene salvato in una memoria temporanea e vengono incrementati il contatore e l'indirizzo in output.

2.1.3 Compute state

Una volta che sono stati letti tutti gli indirizzi dalla memoria parte un'altro ciclo che per ogni periodo di clock calcola la differenza tra l'indirizzo da codificare e l'indirizzo dell'i-esima working-zone e assegna il risultato alla variabile offset. Se l'indirizzo non fa parte di alcuna working-zone viene semplicemente trasmesso concatenato al wz_bit posto a '0'. Nel caso in cui la differenza sia < 4, invece, l'indirizzo viene codificato secondo la specifica, in particolare:

- Il bit wz_bit viene posto a '1'.
- Il numero (i) dell'i-esima working-zone viene convertito in 3 bit e assegnato a wz_num.
- L'offset precedentemente calcolato viene codificato one-hot in wz_offset attraverso un demultiplexer implementato attraverso un case-statement.

In entrambi i casi si ottiene un indirizzo codificato a 8 bit che viene trasmesso allo stato successivo.

2.1.4 Write state

Il componente scrive nell'indirizzo "9" della memoria l'indirizzo codificato dallo stato precedente ed esegue il reset delle variabili utilizzate in esso per la prossima computazione.

2.1.5 First done state

Se la scrittura in memoria è avvenuta correttamente l'uscita o_done viene posta a '1' notificando l'avvenuta elaborazione al testbench.

2.1.6 Second done state

In questo stato il componente attende che il testbench, una volta ricevuto il segnale o_done, risponda abbassando il segnale di i_start. Dopo aver abbassato nuovamente il segnale o_done si passa allo stato di inattività

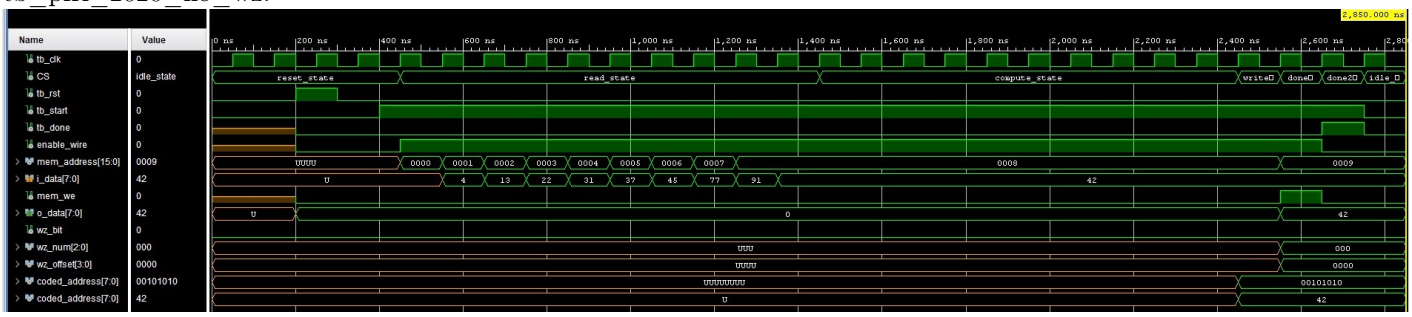
2.1.7 Idle state

Nello stato di inattività il componente progettato esegue l'unica operazione di controllare ad ogni ciclo di clock se il segnale di i_start è stato posto a '1'. In tal caso la macchina passerà allo stato di codifica dell'indirizzo, come richiesto da specifica.

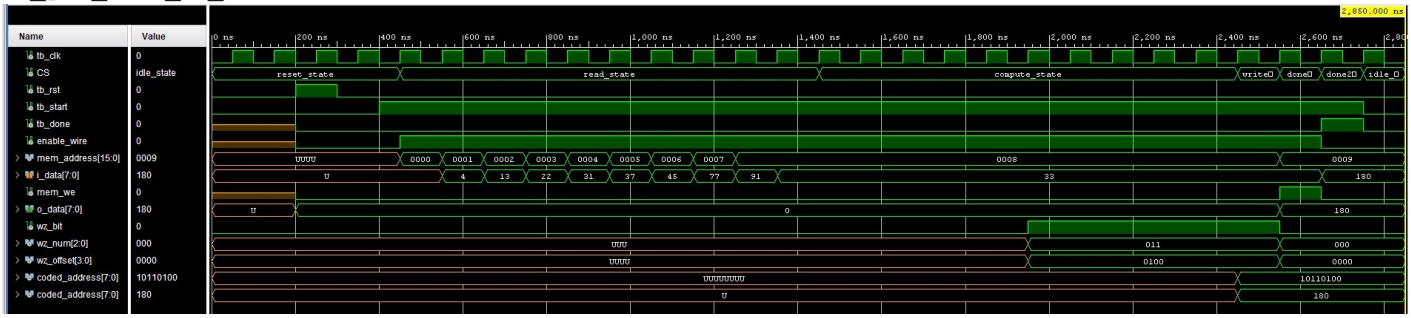
3. Risultati dei Test

3.1 Testbench forniti da specifica

tb_pfrl_2020_no_wz:



tb_pfrl_2020_in_wz:



3.2 Altri test

Per testare il componente ho realizzato i seguenti testbench sulla base di quelli dati da specifica:

3.2.1 Indirizzi casuali

Attraverso un generatore di numeri casuali è stato possibile testare il codice con molteplici combinazioni di numeri diversi, consentendo di sistemare diversi bug e assicurare la correttezza in condizioni normali.

3.2.2 Doppio start

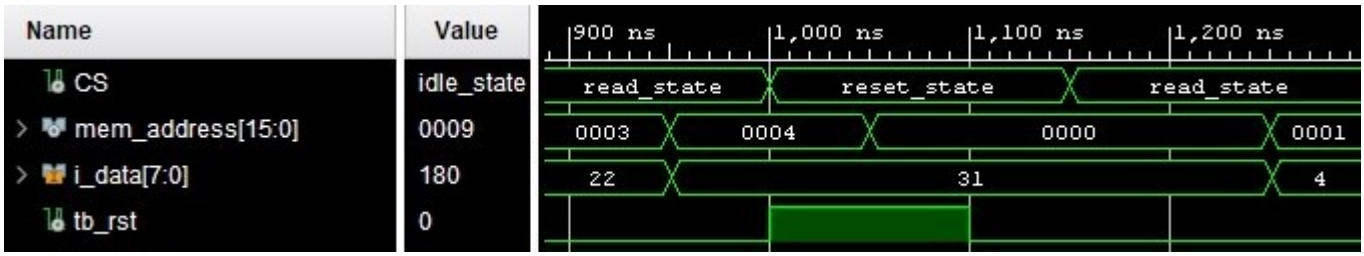
Testbench che verifica la funzionalità richiesta nello stato "idle" di tornare alla codifica dell'indirizzo nel caso in cui il segnale di start venga posto a 1 alla fine dell'elaborazione.

3.2.3 Doppia lettura con reset

Testbench che verifica il corretto funzionamento del componente nel caso in cui, finita l'elaborazione dell'indirizzo codificato, venga fatto il reset della macchina e vengono caricati in memoria dei nuovi indirizzi delle working-zone.

3.2.4 Reset dopo 'n' cicli di clock

Il testbench, subito dopo aver alzato il segnale di start, aspetta 5, 15, 20, 21, 22, 23 cicli di clock e alza nuovamente il segnale di reset per poi proseguire normalmente.
Il numero n di cicli è stato scelto in modo che il segnale di reset si alzasse durante ogni stato della macchina simulando ogni caso possibile di reset improvviso.

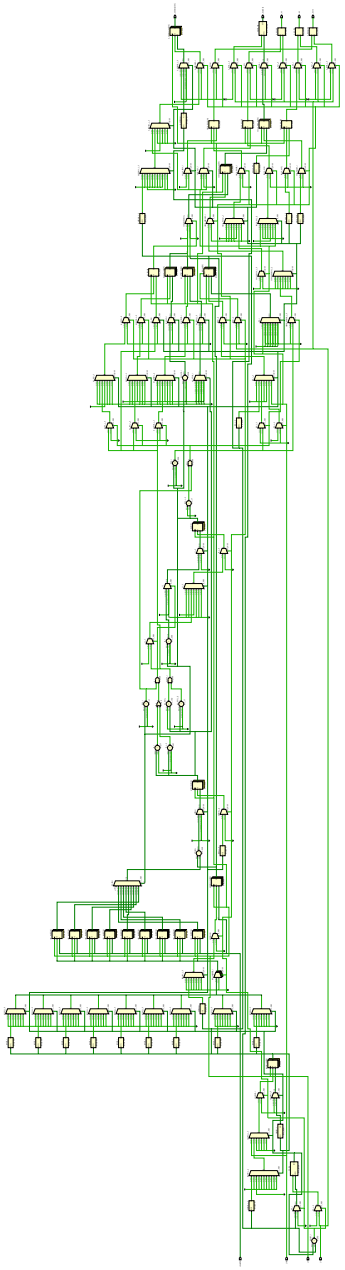


In ogni caso il componente ha passato i test utilizzati con i risultati aspettati.

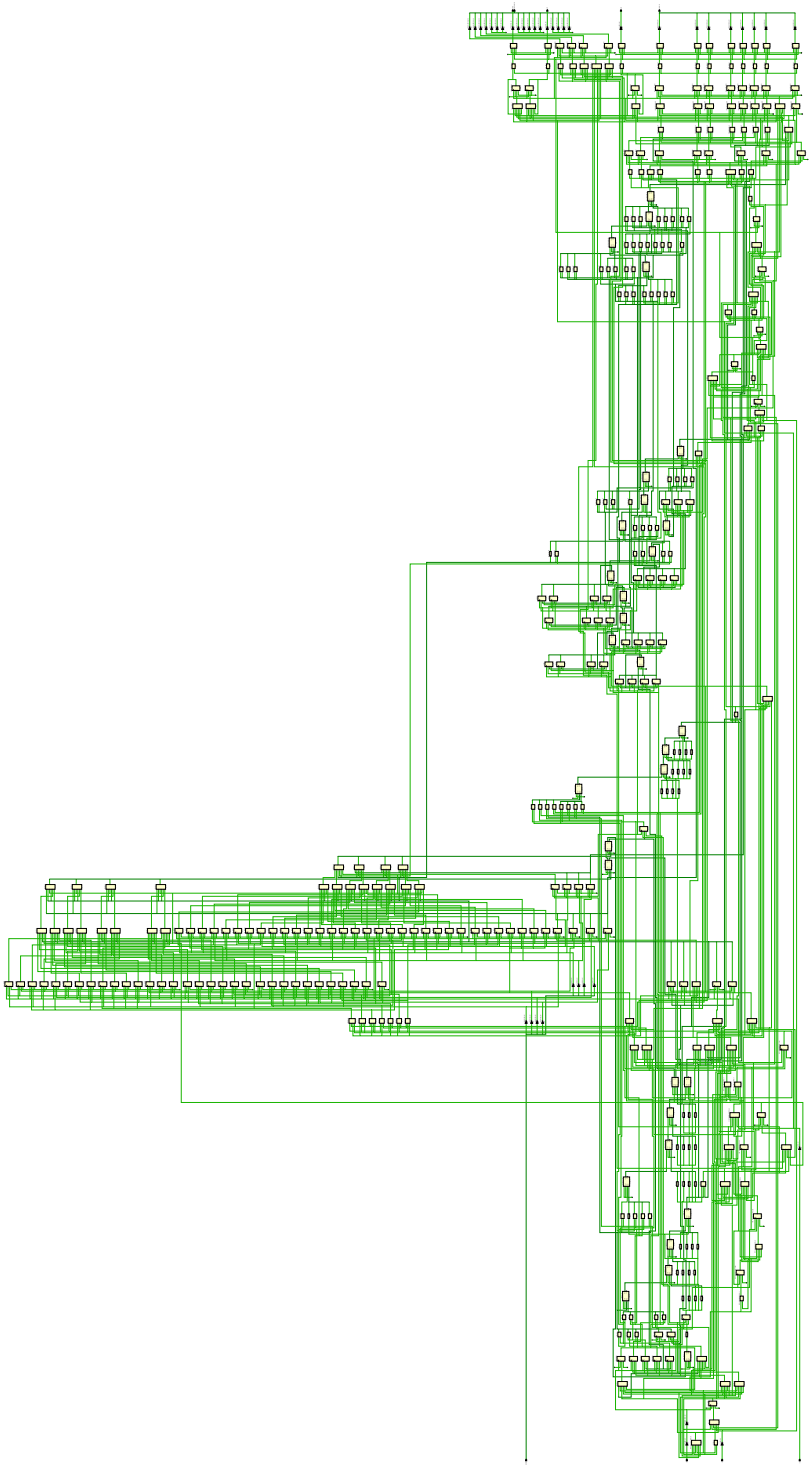
4. Risultati della Sintesi

4.1 Schematic

Elaborated design schematic: *(Orizzontal view)*

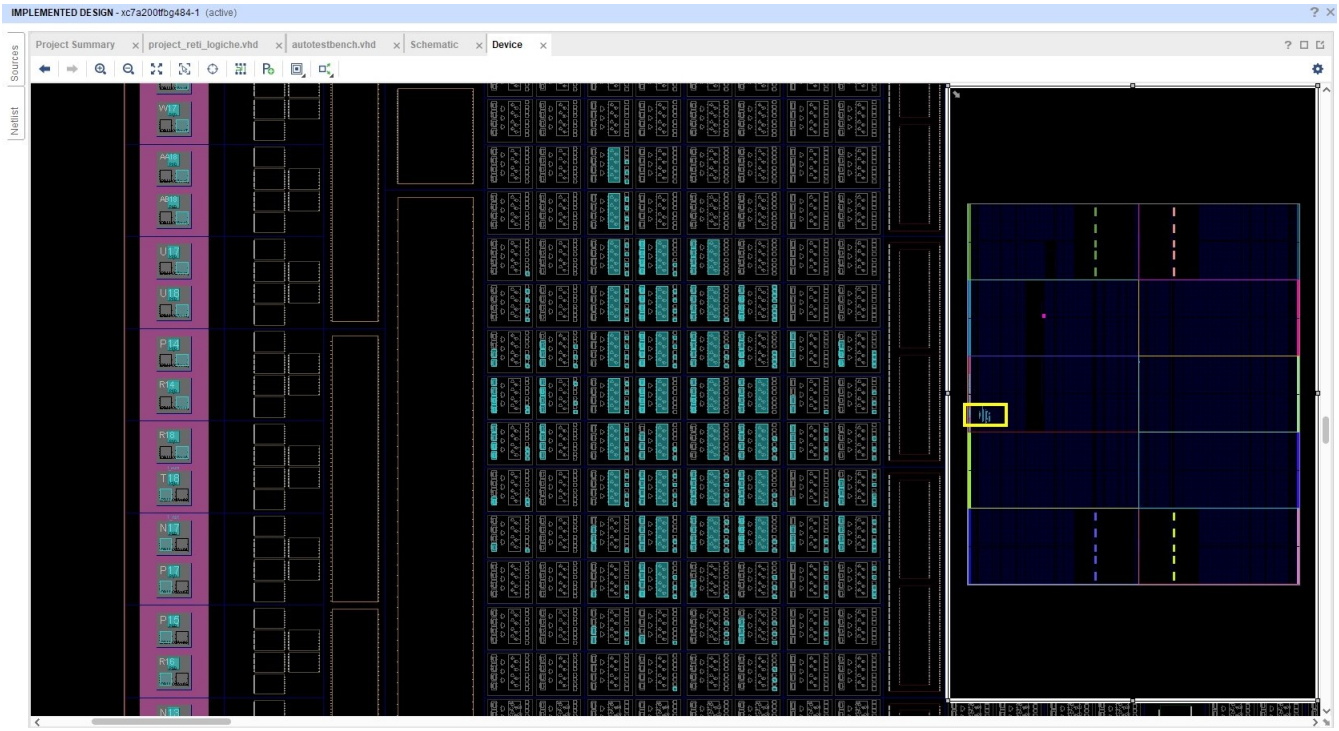


Synthesised design schematic: *(Orizzontal view)*



4.2 Implemented Design

(Non richiesto ma aggiunto per completezza)



5. Possibili Ottimizzazioni

- Nel caso in cui il componente, dopo essere entrato nell'"idle_state", avesse avuto come unica strada possibile il ritorno allo stato di reset (senza dover codificare nuovamente l'indirizzo nel caso in cui il segnale i_start sia impostato nuovamente a '1') il calcolo dell'offset poteva essere compiuto direttamente nel "read_state".
  Leggendo prima l'indirizzo da codificare e poi gli altri nelle 8 posizioni precedenti si potrebbe risparmiare il tempo dovuto alla rilettura di tutti gli indirizzi e i registri necessari a mantenerli in memoria.

Bibliography

[1] T. Lang E. Musoll and J. Cortadella. Working-zone encoding for reducing the energy in microprocessor address buses. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(5):568–572, 1998.