Progetto finale di Reti Logiche

Prof. Fornaciari, Prof. Palermo e Prof. Salice

(AGGIORNATO AL 7 Novembre 2019)

Descrizione generale

La specifica della Prova finale (Progetto di Reti Logiche) 2019 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). Una working-zone è definita come un intervallo di indirizzi di dimensione fissa (Dwz) che parte da un indirizzo base. All’interno dello schema di codifica possono esistere multiple working-zone (Nwz).

Lo schema modificato di codifica da implementare è il seguente:

- se l’indirizzo da trasmettere (ADDR) non appartiene a nessuna Working Zone, esso viene trasmesso così come è, e un bit addizionale rispetto ai bit di indirizzamento (WZ_BIT) viene messo a 0. In pratica dato ADDR, verrà trasmesso WZ_BIT=0 concatenato ad ADDR (WZ_BIT & ADDR, dove & è il simbolo di concatenazione);
- se l’indirizzo da trasmettere (ADDR) appartiene ad una Working Zone, il bit addizionale WZ_BIT è posto a 1, mentre i bit di indirizzo vengono divisi in 2 sotto campi rappresentanti:
    - Il numero della working-zone al quale l’indirizzo appartiene WZ_NUM, che sarà codificato in binario
    - L’offset rispetto all’indirizzo di base della working zone WZ_OFFSET, codificato come one-hot (cioè il valore da rappresentare è equivalente all’unico bit a 1 della codifica).

  In pratica dato ADDR, verrà trasmesso WZ_BIT=1 concatenato ad WZ_NUM e WZ_OFFSET (WZ_BIT & WZ_NUM & WZ_OFFSET, dove & è il simbolo di concatenazione)

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] Per chi fosse interessato l’articolo originale è il seguente: E. Musoll, T. Lang and J. Cortadella, "Working-zone encoding for reducing the energy in microprocessor address buses," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 4, pp. 568-572, Dec. 1998.

Dati

I dati ciascuno di dimensione 8 bit sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0. Anche l'indirizzo che è da specifica di 7 bit viene memorizzato su 8 bit. Il valore dell'ottavo bit sarà sempre zero.

- Le posizioni in memoria da 0 a 7 sono usati per memorizzare gli otto indirizzi base delle working-zone:
    - 0 - Indirizzo Base WZ 0
    - 1 - Indirizzo Base WZ 1
    - ...
    - 7 - Indirizzo Base WZ 7
- La posizione in memoria 8 avrà al suo interno il valore (indirizzo) da codificare (ADDR);
- La posizione in memoria 9 è quella che deve essere usata per scrivere, alla fine, il valore codificato secondo le regole precedenti.

Note ulteriori sulla specifica

1. Nella codifica 1 hot si consideri il bit 0 come il meno significativo. In pratica:
    - WZ_OFFSET = 0 è codificato one hot come 0001;
    - WZ_OFFSET = 1 è codificato one hot come 0010;
    - WZ_OFFSET = 2 è codificato one hot come 0100;
    - WZ_OFFSET = 3 è codificato one hot come 1000;
2. Riprendendo ancora la specifica, il valore codificato sarà così composto:
    - Bit 7: valore del singolo bit di WZ_BIT;
    - Bit 6-4: valore codificato binario di WZ_NUM;
    - Bit 3-0: valore codificato one-hot di WZ_OFFSET
3. Se serve, si consideri che gli indirizzi base delle working-zone non cambieranno mai all'interno della stessa esecuzione;
4. Il modulo partirà nella elaborazione quando un segnale START in ingresso verrà portato a 1. Il segnale di START rimarrà alto fino a che il segnale di DONE non verrà portato alto; Al termine della computazione (e una volta scritto il risultato in memoria), il modulo da progettare deve alzare (portare a 1) il segnale DONE che notifica la fine dell'elaborazione. Il segnale DONE deve rimanere alto fino a che il segnale di START non è riportato a 0. Un nuovo segnale start non può essere dato fin tanto che DONE non è stato riportato a zero. Se a questo punto viene rialzato il segnale di START, il modulo dovrà ripartire con la fase di codifica.

ESEMPIO:
La seguente sequenza di numeri mostra un esempio del contenuto della memoria al termine di una elaborazione. I valori che qui sono rappresentati in decimale, sono memorizzati in memoria con l'equivalente codifica binaria su 8 bit senza segno.

CASO 1 CON VALORE NON PRESENTE IN NESSUNA WORKING-ZONE

Indirizzo Memoria	**Valore**	**Commento**
0	4	// Indirizzo Base WZ 0
1	13	// Indirizzo Base WZ 1
2	22	// Indirizzo Base WZ 2
3	31	// Indirizzo Base WZ 3
4	37	// Indirizzo Base WZ 4
5	45	// Indirizzo Base WZ 5
6	77	// Indirizzo Base WZ 6
7	91	// Indirizzo Base WZ 7
8	42	// ADDR da codificare
9	42	// Valore codificato con in OUTPUT

CASO 2 CON VALORE PRESENTE IN UNA WORKING-ZONE

Indirizzo Memoria	**Valore**	**Commento**
0	4	// Indirizzo Base WZ 0
1	13	// Indirizzo Base WZ 1
2	22	// Indirizzo Base WZ 2
3	31	// Indirizzo Base WZ 3
4	37	// Indirizzo Base WZ 4
5	45	// Indirizzo Base WZ 5
6	77	// Indirizzo Base WZ 6
7	91	// Indirizzo Base WZ 7
8	33	// ADDR da codificare
9	180	// Valore codificato con in OUTPUT (1 - 011 - 0100)

Interfaccia del Componente
Il componente da descrivere deve avere 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;
```

In particolare:
- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_start è il segnale di START generato dal Test Bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_data è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_address è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- o_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.

APPENDICE: Descrizione Memoria
NOTA: **La memoria è già istanziata all'interno del Test Bench e non va sintetizzata**

La memoria e il suo protocollo può essere estratto dalla seguente descrizione VHDL che fa parte del test bench e che è derivata dalla User guide di VIVADO disponibile al seguente link:
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado-synthesis.pdf

```
-- 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;
```