Progetto finale di Reti Logiche

Prof. Fornaciari, Prof. Palermo e Prof. Salice

(CHANGELOG V3)

Versione (12 gennaio -> 18 gennaio)
MODIFICA:
Aggiunti 2 chiarimenti

1. Nella sezione DATI è stata aggiunta la seguente frase:
   "La dimensione massima della sequenza di ingresso è 255 byte."

2. Nella sezione ULTERIORI NOTE, è stata aggiunta la seguente frase facendo riferimento all'azione di una codifica successiva:
   "La quantità di parole da codificare sarà sempre memorizzata all'indirizzo 0 e l'uscita deve essere sempre memorizzata a partire dall'indirizzo 1000."

Versione (10 dicembre -> 12 gennaio)
MODIFICA:
E' stato cambiato il diagramma degli stati invertendo l'ordine delle uscite (precedentemente p2k, p1k ora p1k, p2k) per essere più coerenti con il loro riordino per la generazione del flusso di uscita. Il testo è stato aggiornato di conseguenza come segue (la parte in grassetto è stata aggiunta).

> *"Il convolutore è una macchina sequenziale sincrona con un clock globale e un segnale di reset con il seguente diagramma degli stati che ha nel suo 00 lo stato iniziale,* ***con uscite in ordine P1K, P2K (ogni transizione è annotata come Uk/p1k, p2k)****"*

Progetto finale di Reti Logiche

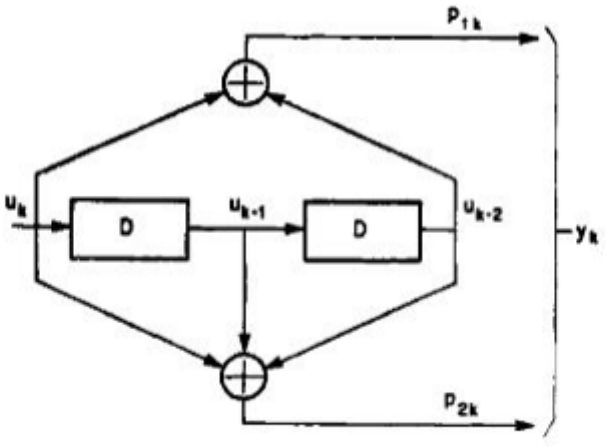
Prof. Fornaciari, Prof. Palermo e Prof. Salice

(AGGIORNATO AL 12 Gennaio 2021)

Descrizione generale

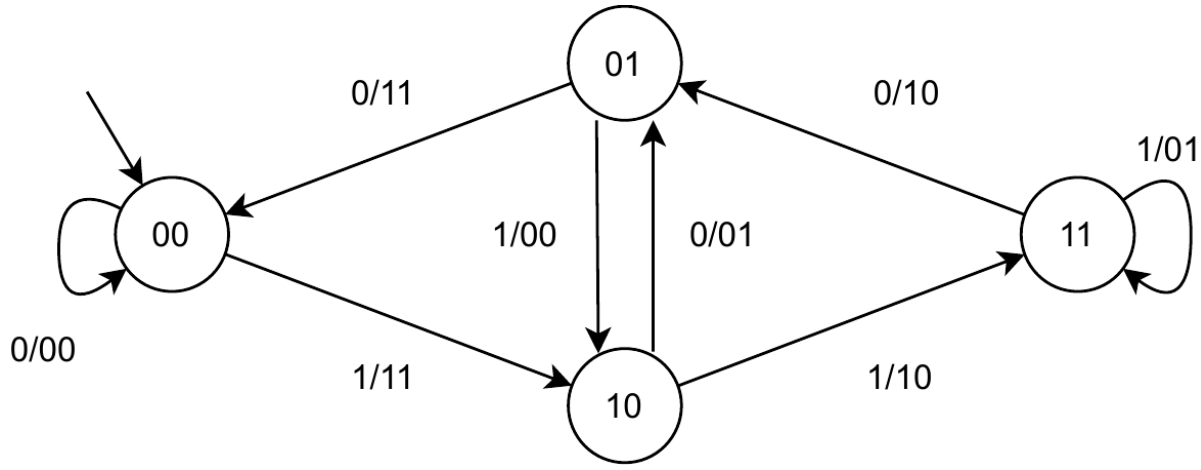
La specifica della "*Prova Finale (Progetto di Reti Logiche)*" 2021/2022 chiede di implementare un modulo HW (descritto in VHDL) che si interfacci con una memoria e che segua la seguente specifica.

Il modulo riceve in ingresso una sequenza continua di W parole, ognuna di 8 bit, e restituisce in uscita una sequenza continua di Z parole, ognuna da 8 bit. Ognuna delle parole di ingresso viene serializzata; in questo modo viene generato un flusso continuo U da 1 bit. Su questo flusso viene applicato il codice convoluzionale ½ (ogni bit viene codificato con 2 bit) secondo lo schema riportato in figura; questa operazione genera in uscita un flusso continuo Y. Il flusso Y è ottenuto come concatenamento alternato dei due bit di uscita. Utilizzando la notazione riportata in figura, il bit uk genera i bit p1k e p2k che sono poi concatenati per generare un flusso continuo yk (flusso da 1 bit). La sequenza d'uscita Z è la parallelizzazione, su 8 bit, del flusso continuo yk.



Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$.

La lunghezza del flusso U è 8*W, mentre la lunghezza del flusso Y è 8*W*2 (Z=2*W).
Il convolutore è una macchina sequenziale sincrona con un clock globale e un segnale di reset con il seguente diagramma degli stati che ha nel suo 00 lo stato iniziale, con uscite in ordine P1K, P2K (ogni transizione è annotata come Uk/p1k, p2k).



Un esempio di funzionamento è il seguente dove il primo bit a sinistra (il più significativo del BYTE) è il primo bit seriale da processare:

- BYTE IN INGRESSO = 10100010 (viene serializzata come 1 al tempo t, 0 al tempo t+1, 1 al tempo t+2, 0 al tempo t+3, 0 al tempo t+4, 0 al tempo t+5, 1 la tempo t+6 e 0 al tempo t+7)

Applicando l'algoritmo convoluzionale si ottiene la seguente serie di coppie di bit:

T	0	1	2	3	4	5	6	7
Uk	1	0	1	0	0	0	1	0
P1k	1	0	0	0	1	0	1	0
P2k	1	1	0	1	1	0	1	1

Il concatenamento dei valori Pk1 e Pk2 per produrre Z segue il seguente schema: Pk1 al tempo t, Pk2 al tempo t, Pk1 al tempo t+1 Pk2 al tempo t+1, Pk1 al tempo t+2 Pk2 al tempo t+2, … cioè Z: 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 1

- BYTE IN USCITA = 11010001 e 11001101

NOTA: ogni byte di ingresso W ne genera due in uscita (Z)

Dati

Il modulo da implementare deve leggere la sequenza da codificare da una memoria con indirizzamento al Byte in cui è memorizzato; ogni singola parola di memoria è un byte. La sequenza di byte è trasformata nella sequenza di bit **U** da elaborare. La quantità di parole W da codificare è memorizzata nell'indirizzo 0; il primo byte della sequenza W è memorizzato all'indirizzo 1. Lo stream di **uscita Z** deve essere memorizzato a partire dall'indirizzo 1000 (mille). La dimensione massima della sequenza di ingresso è 255 byte.

Note ulteriori sulla specifica

1. 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.

2. Il modulo deve essere dunque progettato per poter codificare più flussi uno dopo l'altro. Ad ogni nuova elaborazione (quando START viene riportato alto a seguito del DONE basso), il convolutore viene portato nel suo stato di iniziale 00 (che è anche quello di reset). La quantità di parole da codificare sarà sempre memorizzata all'indirizzo 0 e l'uscita deve essere sempre memorizzata a partire dall'indirizzo 1000.
3. Il modulo deve essere progettato considerando che prima della prima codifica verrà *sempre* dato il RESET al modulo. Invece, come descritto nel protocollo precedente, una seconda elaborazione non dovrà attendere il reset del modulo ma solo la terminazione della elaborazione.

Interfaccia del Componente

Il componente da descrivere deve avere 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_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:

- il nome del modulo deve essere project_reti_logiche
- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_start è il segnale di START generato dal Test Bench;
- 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.

ESEMPI

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.

Esempio1: (Sequenza lunghezza 2)
W: 10100010 01001011
Z: 11010001 11001101 11110111 11010010

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	2	\\ Byte lunghezza sequenza di ingresso
1	162	\\ primo Byte sequenza da codificare
2	75	
[...]		
1000	209	\\ primo Byte sequenza di uscita
1001	205	
1002	247	
1003	210	

Esempio2: (Sequenza lunghezza 6)
W: 10100011 00101111 00000100 01000000 01000011 00001101
Z: 11010001 11001110 10111101 00100101 10110000 00110111 00110111 00000000 00110111 00001110 10110000 11101000

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	6	\\ Byte lunghezza sequenza di ingresso
1	163	\\ primo Byte sequenza da codificare
2	47	
3	4	
4	64	
5	67	
6	13	
[...]		
1000	209	\\ primo Byte sequenza di uscita
1001	206	
1002	189	
1003	37	
1004	176	
1005	55	
1006	55	
1007	0	
1008	55	
1009	14	
1010	176	
1011	232	

Esempio3: (Sequenza lunghezza 3)
W: 01110000 10100100 00101101
Z: 00111001 10110000 11010001 11110111 00001101 00101000

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	3	\\ Byte lunghezza sequenza di ingresso
1	112	\\ primo Byte sequenza da codificare
2	164	
3	45	
[...]		
1000	57	\\ primo Byte sequenza di uscita
1001	176	
1002	209	
1003	247	
1004	13	
1005	40	

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 after 2 ns;
        else
          do <= RAM(conv_integer(addr)) after 2 ns;
        end if;
      end if;
    end if;
  end process;
end syn;
```