Prova Finale (Progetto di Reti Logiche)

Corti Riccardo - Matr. 936119
Cod. Persona 10669322

Prof. W. Fornaciari - A.A. 2021/2022
Dicembre 2022

POLITECNICO
MILANO 1863

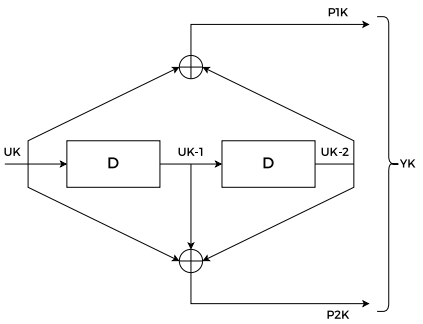
Indice

1 Introduzione

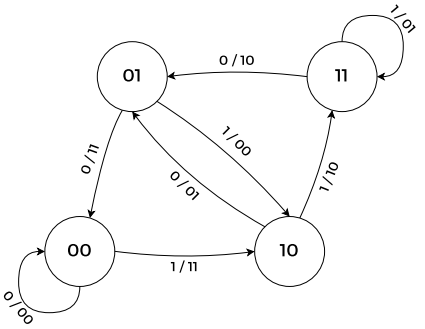
1.1 Scopo del progetto

Il progetto ha come obiettivo la descrizione, in linguaggio VHDL, di un componente hardware che, interfacciandosi con una memoria, riceva un numero N di parole da 8 bit in ingresso, tali parole sono serializzate in un flusso continuo da 1 bit che viene applicato ad un codice convoluzionale con rapporto 1:2. Al termine dell'elaborazione, si ricostruiscono 2 * N parole da 8 bit che vengono salvate nella memoria.

Il codificatore è descritto dalla Fig. 1a, la macchina a stati finiti di Fig. 1b ne mostra il funzionamento. Per ogni bit *Uk* in ingresso, esso produce due bit in uscita in ordine *P1k*, *P2k*. La parola viene serializzata con il bit più significativo (a sx) che viene elaborato per primo, mentre il bit meno significativo (a dx) per ultimo.



(a) Codificatore convoluzionale con tasso 1:2



(b) Macchina a stati finiti del codificatore

Figura 1: Descrizione del codificatore

Di seguito è mostrato un esempio di funzionamento del codice convoluzionale con la parola 01001011_2 in ingresso.

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

Concatenando i valori *P1k* e *P2k*, il risultato è dato dai due byte in uscita: 00110111 11010010.

1.2 Descrizione del componente

Il componente da realizzare è descritto con 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;
```

I segnali in ingresso al componente sono:

- `i_clk` è il segnale di clock generato dal TestBench. È richiesto che il componente funzioni almeno con un periodo di clock di 100 ns.
- `i_rst` è il segnale di reset. Prima della prima codifica viene sempre dato il segnale di reset per preparare il componente.
- `i_start` è il segnale di avvio generato dal TestBench. Il segnale di start viene portato al livello logico '1' quando dovrà partire l'elaborazione e rimarrà alto fino a che il segnale `o_done` non verrà portato anch'esso al livello logico '1', dopodiché sarà portato a '0'.
- `i_data` è il vettore di 8 bit che porta il valore presente in memoria nell'indirizzo specificato dal segnale `o_address` a seguito di una richiesta di lettura.

I segnali in uscita dal componente sono:

- `o_address` è il vettore di 16 bit che porta alla memoria l'indirizzo a cui si vuole accedere, in lettura o scrittura.
- `o_done` è il segnale che indica il termine dell'elaborazione, con tutti i risultati scritti in memoria. Il segnale `o_done`, dopo essere stato portato al livello logico '1', resterà alto fino a che il segnale `i_start` non sarà portato a '0', dopodiché anche `o_done` deve essere portato a '0' altrimenti non può essere dato un nuovo segnale di start.
- `o_en` è il segnale di enable che abilita la memoria sia in lettura che in scrittura.
- `o_we` è il segnale di write enable. Esso deve essere posto a '1' quando si intende accedere alla memoria in scrittura, mentre deve essere posto a '0' se si vuole solo leggere dalla memoria.
- `o_data` è il vettore di 8 bit che porta alla memoria la parola da scrivere all'indirizzo specificato dal segnale `o_address`.

2 Architettura

2.1 Descrizione della soluzione

Il comportamento del componente è descritto da quattro process:

- Il primo, sincronizzato sul fronte di discesa del clock, aggiorna i registri e lo stato della FSM (che verrà descritta più avanti).
- Il secondo aggiorna le porte di output del componente quando lo stato cambia.
- Il terzo legge e modifica i valori in input, ovvero il numero di parole da elaborare, che viene decrementato ad ogni parola letta, e la parola da 8 bit che viene shiftata a sinistra di un bit (a destra viene inserito uno '0') ad ogni operazione.
- Il quarto, in ascolto sui registri che sono aggiornati nel primo process, realizza il codice convoluzionale leggendo il bit più significativo del registro in cui è memorizzata la parola che si sta elaborando, shiftando i 16 bit del registro contenente il risultato di due posizioni a sinistra e, in base allo stato della FSM del codificatore, scrivendo i due bit meno significativi del risultato e selezionando lo stato successivo.

2.2 Descrizione della FSM

In Fig. 2 è rappresentato lo schema della FSM che descrive il funzionamento completo del componente.

Di seguito vengono descritti i singoli stati:

- IDLE – Stato di attesa del segnale di avvio. Questo stato viene raggiunto al termine di un'elaborazione e mantenuto fintanto che `i_start` è pari a '0', il componente prepara i segnali ad una nuova elaborazione. In caso si riceva un segnale di reset, in qualsiasi punto si trovi, il componente viene riportato in questo stato e preparato per la prima elaborazione.
- READ_NUM_OF_WORDS – Stato di richiesta alla memoria. In questo stato il componente manda, alla memoria, una richiesta di lettura all'indirizzo in cui è salvato il numero di parole che dovranno essere elaborate (indirizzo 0_{10}).
- WAIT_MEM – Stato di attesa della memoria. A seguito di una richiesta di lettura, la risposta della memoria è accessibile al ciclo di clock successivo, da qui la necessità di uno stato di attesa.
- READ_NTH_WORD – Stato di richiesta alla memoria. In questo stato il componente richiede alla memoria la successiva parola da elaborare. (le parole sono memorizzate a partire dall'indirizzo 1_{10})
- WAIT_MEM_2 – Stato di attesa della memoria. Analogo allo stato WAIT_MEM. In questo stato viene anche inizializzato un contatore al valore intero 8.

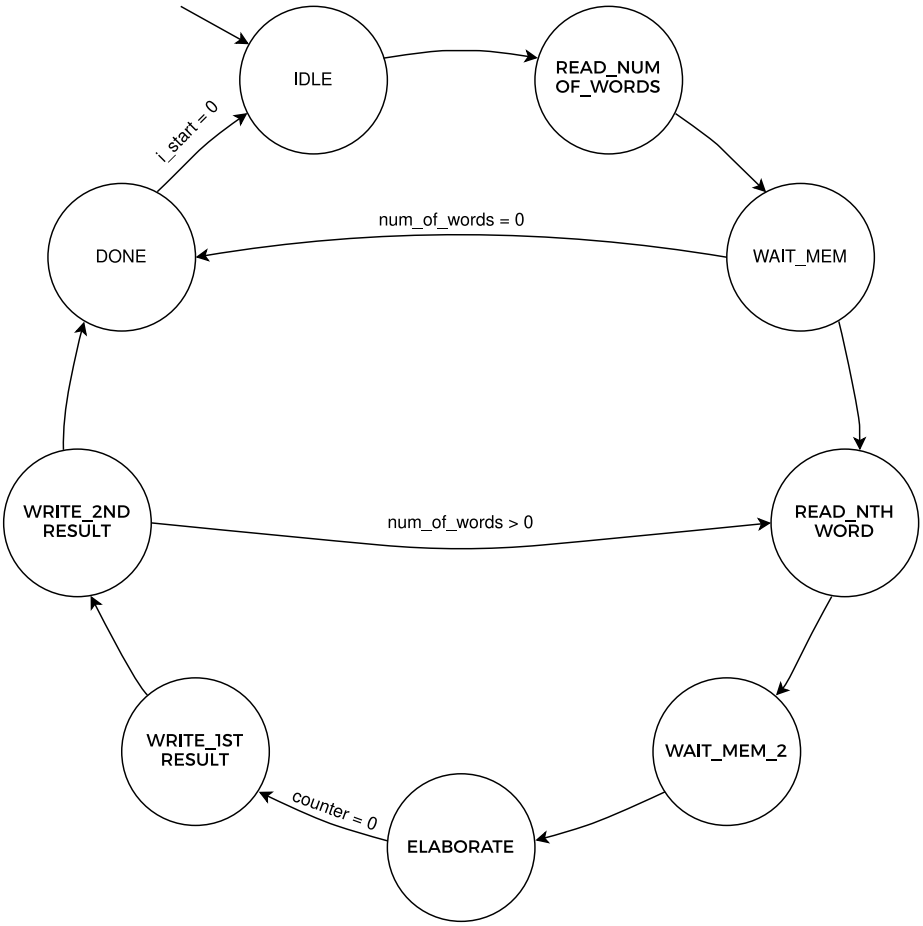


Figura 2: Diagramma della FSM

- ELABORATE – Stato di computazione della codifica. In questo stato il componente elabora gli 8 bit della parola letta, secondo il codice convoluzionale, scrivendo il risultato in un vettore da 16 bit. Per elaborare tutti i bit, il contatore inizializzato ad 8 viene decrementato ad ogni operazione ed il componente rimane in questo stato finché il contatore è maggiore di 0.
- WRITE_1ST_RESULT – Stato di scrittura del risultato in memoria. In questo stato il componente richiede alla memoria di scrivere gli 8 bit più significativi del vettore contenente il risultato della codifica (i risultati sono memorizzati a partire dall'indirizzo 1000_{10}).
- WRITE_2ND_RESULT – Stato di scrittura del risultato in memoria. In questo stato il componente richiede alla memoria di scrivere gli 8 bit meno significativi del vettore contenente il risultato della codifica (i risultati sono memorizzati a partire dall'indirizzo 1000_{10}). In questo stato si verifica inoltre che non ci siano più parole da codificare, in caso contrario si ritorna allo stato READ_NTH_WORD.

- DONE – Stato di fine elaborazione. Questo stato viene raggiunto quando sono state codificate tutte le parole ed il segnale `o_done` viene portato ad '1'. Il componente rimane in questo stato fino a che `i_start` non viene portato a '0'.

3 Risultati Sperimentali

3.1 Sintesi

A seguito della sintesi del componente, il report_utilization di Vivado mostra i seguenti risultati:

```
 Target FPGA xc7k70tfbv676-1
+-----------------------+------+ ----------+-------+
|       Site Type       | Used | Available | Util% |
+-----------------------+------+ ----------+-------+
| LUT as Logic          |   68 |     41000 |  0.17 |
| LUT as Memory         |    0 |     13400 |  0.00 |
| Register as Flip Flop |   97 |     82000 |  0.12 |
| Register as Latch     |    0 |     82000 |  0.00 |
| F7 Muxes              |    0 |     20500 |  0.00 |
| F8 Muxes              |    0 |     10250 |  0.00 |
+-----------------------+------+ ----------+-------+
```

In merito al numero di Flip Flop sintetizzati, hanno impattato alcuni registri utilizzati per semplificare le operazioni che il componente deve compiere.

3.2 Simulazioni

Il componente è stato testato sotto le due forme di simulazione richieste: Behavioral Simulation e Post-Synthesis Functional Simulation, passando tutti i casi di test descritti di seguito.

3.2.1 Test Generali

Una serie di test in cui viene verificato il corretto funzionamento del codificatore sottoponendogli diverse parole da elaborare.

3.2.2 Doppia esecuzione

In questo test, il componente elabora lo stesso flusso di parole due volte consecutivamente verificando che, al termine dell'esecuzione, si reimposti correttamente in attesa di una nuova elaborazione.

3.2.3 Reset asincrono

In questo test, mentre il componente sta lavorando viene mandato un segnale di reset, verificando che si arresti e azzeri correttamente ricominciando l'elaborazione da capo.

3.2.4 Sequenza massima

In questo test, il componente deve elaborare il massimo numero di parole possibile (255_{10} parole).

3.2.5 Sequenza minima

In questo test, il componente deve elaborare il minimo numero di parole possibile (0_{10} parole). Non solo il numero di parole è pari a 0_{10}, ma sono presenti in memoria delle parole che viene verificato non vengano elaborate.

3.2.6 Tre diversi flussi consecutivi

Questo test è simile al test di Doppia esecuzione ma i flussi di parole sono differenti, oltre ad essere più di due.

3.2.7 Tre diversi flussi consecutivi con reset asincrono

Questo test unisce il precedente con il test di Reset asincrono, pertanto al componente vengono forniti tre flussi diversi di parole consecutivamente, tuttavia l'elaborazione del primo flusso viene resettata con un segnale di reset.

4 Conclusioni

Il componente è correttamente sintetizzabile ed ha passato i numerosi test a cui è stato sopposto sia in Behavioral sia in Post-Synthesis Functional.

Il componente è stato anche testato con periodi di clock inferiori a 100 ns, fino a 10 ns sebbene non sia il limite di funzionamento e, molto probabilmente, si potrebbe diminuire ulteriormente.

La soluzione proposta può sicuramente essere ottimizzata maggiormente, ciononostante rimane efficace e semplice sia nel descrivere il comportamento del componente che nel caso di un intervento per modificare il codice.