

Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo

Anno 2021-2022

Enrico Simionato (Matricola – Codice Personaw)

Indice

1 Introduzione

1.1 Scopo del progetto

1.2 Specifica di progetto

1.3 Interfaccia del componente

1.4 Memoria e interazione tra la memoria e il componente

2 Architettura

2.1 Design generale del componente

2.2 Macchina a stati del convolutore

2.3 Componente generale e stati di funzionamento

2.4 Considerazioni sul componente

3 Risultati sperimentali

3.1 Introduzione ai test

3.2 Test generici

3.3 Test dei casi particolari

4 Conclusioni

4.1 Conclusioni

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto è la realizzazione di un componente hardware descritto in linguaggio VHDL che legga l’input da una memoria e che applichi sui dati in ingresso un codice convoluzionale con tasso di trasmissione ½ secondo una uno schema in seguito descritto. Il componente da realizzare dovrà infine scrivere il risultato dell’elaborazione in memoria ed tornare in una configurazione che permetta di eseguire l’elaborazione di nuovi dati.

1.2 Specifica del progetto

Il modulo richiesto dalla specifica deve essere in grado di interfacciarsi con la memoria ed eseguire su i dati in essa contenuti una trasformazione secondo un codice convoluzionale con rapporto ½.

Il convolutore è una macchina sequenziale sincrona dotata di un segnale di reset. Per codice convoluzionale m/n si intende un codice che trasformi m bit della stringa in ingresso in n bit della stringa in uscita secondo una specifica funzione di trasformazione. Come si vede dalla immagine la trasformazione in esame si basa sulla presenza di un registro di dimensione due bit che memorizza i valori, serializzati in ingresso dalla memoria. Tale componente è sincronizzato dunque a ogni ciclo di clock viene inserito un nuovo valore nel convolutore e i valori contenuti nei flip-flop vengono fatti scorrere. Le uscite e gli stati, i quali sono gli ingressi al componente ai cicli precedenti, sono calcolate come segue:

immagine convolutore

con Uk ingresso al convolutore all’istante k e Yk = [P1k P2k] uscita dal convulutore

* Q1 = U(k-1)
* Q2 = U(k-2)
* P1k = Uk xor U(k-2)
* P2k = Uk xor U(k-1) xor U(k-2)

Oltre a eseguire la codifica il componente della specifica deve anche gestire l’interazione con la memoria. Esso leggerà W parole da 8 bit dalla memoria e restituirà in uscita Z = 2 \* W parole da 8 bit che verranno scritte in RAM ad un indirizzo noto.

In quanto il componente illustrato necessita di uno stream continuo di bit in ingresso e fornisce in uscita uno stream di bit di lunghezza doppia rispetto a quella degli input l’entità da implementare dovrà gestire anche la serializzazione degli ingressi e la de-serializzazione delle uscite. Le parole di memoria verranno serializzate e poste in ingresso al convolutore mentre in uscita da esso lo stream verrà deserializzato per scrivere il risultato in memoria.

A partire dal componente è possibile costruire la relativa macchina a stati finiti che viene fornita da specifica e semplifica l’implementazione in VHDL.

Immagine macchina a stati

1.3 Interfaccia del componente

Il componente implementato ha 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 specificati nell’interfaccia del componente hanno la seguente semantica:

* 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 di dimensione 1 byte) che arriva dalla memoria in seguito ad una richiesta di lettura;
* o\_address è il segnale (vettore di dimensione 2 byte) di uscita che manda l’indirizzo alla memoria;
* o\_done è il segnale di uscita che comunica la fine dell’elaborazione e la presenza del dato di uscita in memoria;
* 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 dimensione 1 byte) di uscita dal componente verso la memoria.

1.4 Memoria e interazione tra la memoria e il componente

Il componente da realizzare si interfaccerà con una memoria dalla quale leggerà i dati da processare e all’interno della quale scriverà il risultato dell’elaborazione dell’ingresso.

La memoria che utilizza il componente è indirizzata al byte. I dati in ingresso al componente da essa hanno dimensione 1 byte e saranno posti nell’ingresso i\_data. Allo stesso modo anche i dati scritti in memoria hanno dimensione 1 byte e vengono posti in o\_data. L’indirizzo a cui letti o scritti i dati viene indicato nell’uscita o\_address e la scrittura o lettura è regolata dall’uscita o\_we.

La memoria massima a cui il componente può accedere sarà di 65536 byte (il progetto ne userà molti di meno per letture e scritture), in quanto o\_address è di 2 byte.

Il progetto assume che i dati da leggere saranno posizionati in memoria a partire dall’indirizzo 0 e in particolare all’indirizzo 0 sarà presente il numero di parole da leggere e dall’indirizzo 1 all’indirizzo 256 saranno presenti i byte da elaborare.

Il risultato dell’elaborazione dei dati in ingresso sarà salvato a partire dall’indirizzo 1000 e, data la specifica da implementare, sarà di una lunghezza doppia rispetto ai byte da elaborare in ingresso (non considerando la parola che contiene il numero di byte da processare).

Schema della memoria (da fare in latex)

Va sottolineato come quale è il protocollo con cui comunica la memoria, almeno dal testbench e i suoi tempi (cicli di clock) di risposta.

2 Architettura

2.1 Design generale del componente

Le funzioni che devono essere realizzate dal componente sono:

* lettura delle parole di memoria
* serializzazione delle parole in un flusso di bit in ingresso alla FSM
* applicazione della codifica convoluzionale ½
* scrittura dei risultati in memoria

La scelta progettuale che ho fatto è quella di allocare tali funzioni a due soli moduli che si interfacciano per realizzare il componente, essi sono:

* una macchina a stati finiti che effettui la codifica convoluzionale ½
* una macchina a stati finiti che si occupa dell’interazione con la memoria e della serializzazione dei dati in ingresso per poter essere elaborati dal convolutore

2.2 Macchina a stati del convolutore

La macchina a stati del convolutore esegue la computazione richiesta al componente ovvero esegue la codifica dello stream di byte in ingresso secondo la codifica convoluzionale ½. Tale FSM riceve in ingresso i bit serializzati dalla memoria e restituisce il risultato della trasformazione che verrà scritto in memoria dalla restante parte del componente.

La macchina a stati del convolutore implementa il funzionamento che il componente deve avere a fronte degli ingressi posti in input.

L’implementazione in VHDL della macchina a stati del convolutore è di facile lettura in quanto è presenta una struttura standard. Va però evidenziata la presenza di un segnale che ferma e “salva” la condizione di funzionamento della macchina, tale segnale è stop\_fsm;

Il segnale stop\_fsm ha la seguente semantica: se settato a 0 la macchina esegue il suo funzionamento standard usando la funzione di stato e delle uscite per calcolare lo stato prossimo e le nuove uscite a ogni ciclo di clock mentre se è settato a 1 la macchina interrompe il funzionamento non modificando più lo stato o le uscite. La FSM viene congelata nello stato che aveva quando l’ingresso stop\_fsm è stato portato a 1. La computazione degli ingressi ricomincerà quando il segnale verrà riportato al valore 0 e in tale istante lo stato della macchina sarà quello in cui si era precedentemente stata interrotta.

2.3 Componente generale e stati di funzionamento

Il funzionamento complessivo del componente è gestito da una macchina a stati che modifica il suo stato sulla base degli eventi che si susseguono durante la computazione. Ogni stato della macchina a stati esegue un certo insieme di compiti che permettono la prosecuzione del funzionamento. In particolare si ha che quando la macchina è nello stato ESECUZIONE\_E\_STAMPA\_PRIMO\_BYTE viene posto all’ingresso alla macchina a stati del convolutore un bit dello stream di byte da computare e avviene la vera e propria esecuzione.

Da specifica di progetto prima di che il componente inizi a leggere da memoria i dati e ad eseguirli esso riceverà un segnale di reset (i\_rst posto a 1). Tale segnale porterà la macchina nello stato ATTESA\_START che è lo stato di atteso del segnale i\_start, è considerabile come stato iniziale della macchina. Quando il segnale i\_start in ingresso al componente viene portato a 1, esso inizia l’elaborazione. Una volta terminata la computazione, dopo avere scritto il risultato in memoria, il componente pone a 1 il segnale o\_done. La memoria (test bench) risponde abbassando i\_start e,a seguito di tale azione, il componente riporta a 0 o\_done; il componente ritorna nello stato ATESA\_START, in attesa che il segnale i start torni alto.

ATTESA\_START

Lo stato ATTESA\_START è lo stato di reset del componente. Il componente inizialmente si troverà in tale stato e vi rimarrà fino a quando non viene posto il segnale START al valore 1, che sta a significare che la memoria è pronta e che il componente dovrà iniziare a elaborare i dati.

Quando START = 1 il componente passerà nello stato LETTURA\_W, l’attesa nella lettura della parola all’indirizzo 0 è dovuta alla possibilità

LETTURA\_W

Lo stato LETTURA\_W è lo stato in cui avverrà la lettura della quantità W che, come precisato nella specifica, rappresenta il numero di parole da leggere dalla memoria e processare.

Questo stato imposta l’indirizzo di lettura a 1dec in modo tale che negli stati precedenti possa essere letta la prima parola da elaborare.

Lo stato successivo sarà sempre ATTESA\_LETTURA\_MEMORIA.

ATTESA\_LETTURA\_MEMORIA

Lo stato ATTESA\_LETTURA\_MEMORIA è uno stato fa trascorrere un ciclo di clock senza effettuare modifiche delle uscite o proseguire l’esecuzione in modo tale che la memoria abbia il tempo di caricare il valore corretto del dato all’indirizzo o\_address richiesto il ciclo precedente.

Tale stato resetta inizializza la fase di esecuzione di una parola letta da memoria.

Lo stato successivo sarà sempre LETTURA\_INPUT\_O\_TERMINE\_ESECUZIONE.

LETTURA\_INPUT

Lo stato LETTURA\_INPUT presenta un if statement che prevede la lettura del byte da elaborare dalla memoria, se ve ne sono ancora da processare, mentre prevede il reset dei componenti e il set a 1 dell’uscita o\_done, che segnala il termine dell’esecuzione sull’insieme di dati passati in input, se tutte le parole sono state lette.

Nel caso in cui siano presenti altre parole da eseguire il prossimo stato sarà ESECUZIONE\_E\_STAMPA\_PRIMO\_BYTE mentre nel caso in cui siano state elaborate tutte le W parole lo stato successivo sarà RESET\_DONE.

ESECUZIONE\_E\_STAMPA\_PRIMO\_BYTE

Lo stato ESECUZIONE\_E\_STAMPA\_PRIMO\_BYTE è lo stato in cui viene gestita l’esecuzione della macchina a stati che realizza il convolutore. Mediante un contatore verranno scanditi tutti e otto i bit della parola letta dalla memoria e ognuno di essi verrà posto in ingresso al componente “FSM”, realizzando così la serializzazione della parola letta.

In tale stato verranno anche memorizzate le uscite del convolutore frutto dell’elaborazione dei dati posti in ingrasso ai cicli di clock precedenti.

Una volta ottenuti 8 bit in output dal convolutore essi verranno scritti in memoria a partire dall’indirizzo 1000 ogni due parole di memoria

STAMPA\_SECONDO\_BYTE

Preparazione dei segnali necessari alla stampa del secondo byte generato dalla computazione della macchina a stati relativa al convolutore. In particolare in tale stato viene impostato l’indirizzo di scrittura in cui dovrà essere scritta la parola di memoria e viene settato a 1 il valore del segnale o\_we, il quale indica alla memoria che vi è una richiesta di scrittura.

La scrittura avverrà a partire dall’indirizzo 1001 e ogni due parole di memoria.

Lo stato successivo è sempre IMPOSTAZIONE\_INDIRIZZO\_LETTURA.

IMPOSTAZIONE\_INDIRIZZO\_LETTURA

Lo stato IMPOSTAZIONE\_INDIRIZZO\_LETTURA prepara la memoria alla lettura del dato successivo. Le operazioni principali effettuate in tale stato sono l’impostazione del nuovo indirizzo di memoria da leggere e il reset di o\_we a 0 in modo tale che venga richiesta alla memoria una lettura e non una scrittura.

Lo stato successivo è sempre LETTURA\_INPUT\_O\_TERMINE\_ESECUZIONE.

RESET\_DONE

Lo stato RESET\_DONE è lo stato in cui si trova la macchina a fine esecuzione. Tale stato attende che il segnale i\_start venga resettato a 0 per poi abbassare o\_done e riportarsi nelle condizioni iniziali di esecuzione dando così la possibilità di eseguire una nuova elaborazione dall’inizio.

Quando il segnale di start verrà riportato a 0 la macchina cambierà stato andando in ATTESA\_START.

2.4 Considerazioni sul componente e note

La disattivazione della memoria era superflua in quanto nella specifica non sono presenti vincoli riguardo l’utilizzo e il tempo di utilizzo della memoria.

Il componente poteva essere implementato anche con due process invece che mediante due componenti distinti ma inizialmente mi sembrava sensato implementare la specifica in modo modulare perché il convolutore e il modulo di interazione con la memoria sono entità distinte.

3 Risultati sperimentali

3.1 Introduzione ai test

Al fine di verificare il corretto funzionamento del componente implementato sono stati effettuati molteplici test suddivisibili in test generici, i quali testano la correttezza dei risultati in casi standard, e test dei casi particolari, i quali testano situazioni di esecuzione particolari. Se per i primi test un implementazione corretta della macchina a stati è il punto fondamentale prendere in considerazione per i secondi va valutata a fondo l’esecuzione della macchina nel suo complesso e come essa si deve comportare a fronte di certi eventi

3.2 Test generici

Sono stati

TEST con 0 parole

TEST con 255 parole

TEST numero intermedio

3.3 Test dei casi particolari

TEST con reset in mezzo alla computazione

TEST multipli in sequenza

TEST memorie multiple

TEST con start e reset che variano sullo stesso fronte

TEST

4 Conclusioni

4.1 Conclusioni

Il componente implementato e sintetizzato supera correttamente i test specificati nella