Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo – a.a. 2021/2022

Francesco Scroccarello (Codice Persona 10723028, Matricola 936982)

Indice

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto consiste nell'implementare un componente hardware descritto in VHDL che si interfacci con una memoria e che converta le parole lette in maniera continua secondo un codice convoluzionale 1:2. Ad ogni parola letta pertanto corrisponderanno due parole in uscita.

1.2 Algoritmo di convoluzione

L'algoritmo che viene realizzato è il seguente:

- siano x il k-esimo bit in ingresso, q1 il (k-1)-esimo bit e q2 il (k-2)-esimo bit.

Al k-esimo bit in elaborazione corrisponderanno due bit in uscita, calcolati come:

- Pk1=x xor q2,
- Pk2= x xor(q1 xor q2).

L'output corrispondente sarà dato dalla concatenazione di questi due bit, con Pk1 MSB della sequenza relativa in uscita. Si noti che all'inizio della computazione i bit (k-1) e (k-2) corrispondono a due zeri "fantasma".
Ad esempio, se come sequenza da convertire ho:

10100010 01001011

In uscita la sequenza corrispondente sarà:

11010001 11001101 11110111 11010010

(esempio tratto dalla tb_example fornita dal docente).

1.3 Struttura della memoria

La memoria con cui si interfaccia il componente ha la seguente struttura.
Il numero di parole da leggere è codificato a partire dall'indirizzo (0). A seguire sono memorizzate le parole da convertire. L'output del componente è infine memorizzato nella memoria a partire dall'indirizzo (1000). Si noti che il progetto non comprende la progettazione della memoria e che quella fornita come test segue la tipologia e protocollo "*Single-Port Block RAM Write-First Mode*".
Di seguito un breve schema della struttura memoria:

0	#Parole da codificare
1	Parola #1
2	Parola #2
…	…
N	Parola #n
…	…
1000	Output #1
1001	Output #2
1002	Output #3
…	…
1000+2N	Output #2n
…	…

1.4 Interfaccia e collegamento

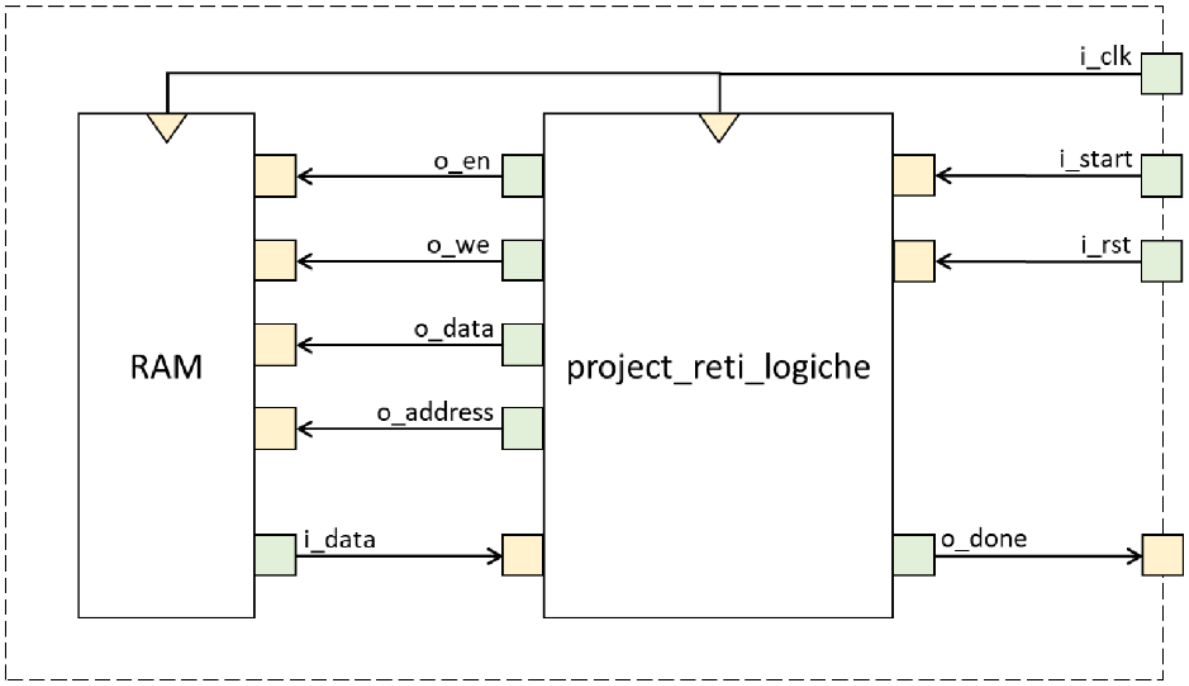
Il componente da descrivere ha un'interfaccia così definita:

```
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;
```

con:

i_clk segnale di clock della memoria.
i_rst segnale di reset che precede l'inizio della computazione.
i_start segnale di start che avvia la computazione.
i_data parola correntemente letta dalla memoria.
o_address indirizzo di memoria a cui viene chiesto di svolgere l'operazione.
o_done segnale che indica la fine della computazione.
o_en segnale che attiva la memoria.
o_we segnale che abilita la scrittura su memoria.
o_data parola correntemente scritta sulla memoria.

Il componente risulta essere così connesso alla memoria:



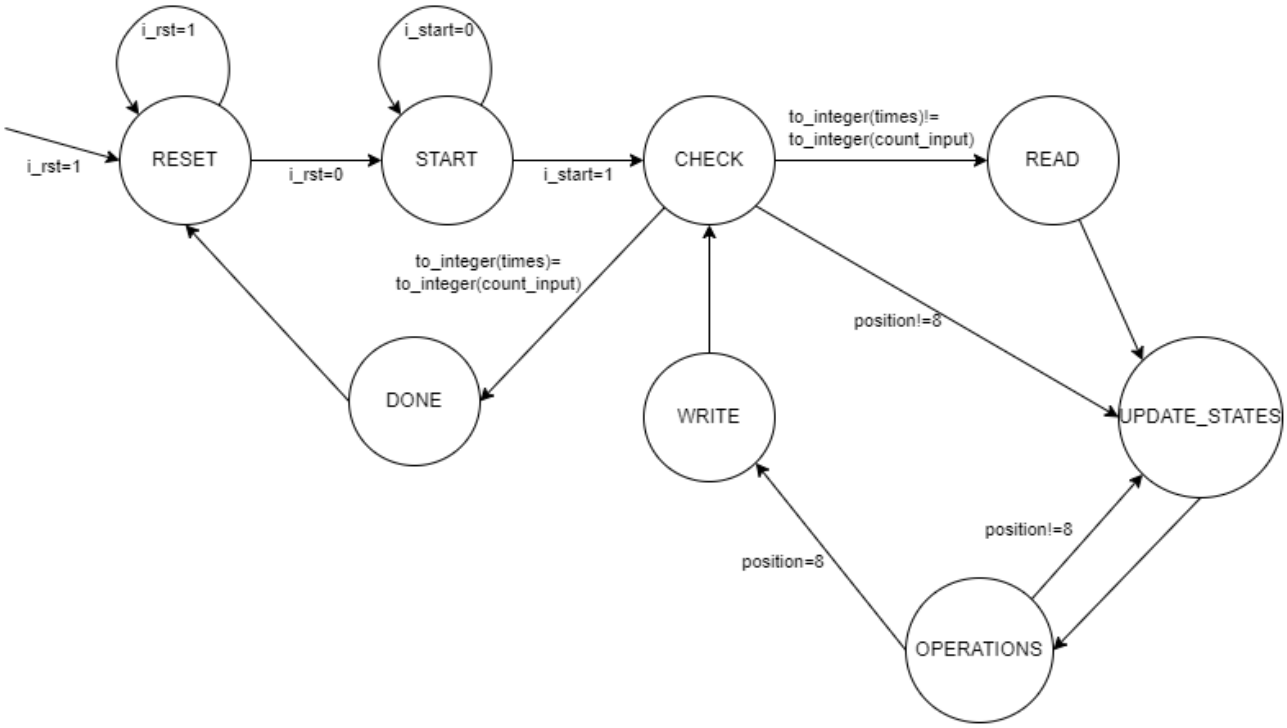
2 Architettura

2.1 Modello

Il problema è stato modellizzato con una macchina a stati finiti che in VHDL è stata realizzata con due processi, SYNC_PROC che sincronizza gli stati della macchina e si occupa di gestire gli eventuali segnali di reset in ingresso, e NEXT_STATE_DECODE che si occupa di tutta la computazione, stato per stato.

2.2 Stati della macchina

La macchina si compone di 8 stati totali e il funzionamento descritto dal grafo riportato. Si noti che per rendere il disegno più leggero e comprensibile alcune condizioni sono state semplificate:



Di seguito una breve descrizione di ciascuno stato:

RESET	stato in cui avviene il reset del componente. Tutti i registri interni sono inizializzati con il loro valore di default.
START	stato in cui ha inizio la computazione. Viene letto il numero di parole da computare.
CHECK	stato di controllo in cui viene deciso se dar fine alla computazione o se andare avanti.
READ	stato in cui avviene la lettura della parola da elaborare.
UPDATE_STATES	stato in cui viene aggiornato il valore dei bit k, (k-1) e (k-2).
OPERATIONS	stato in cui viene realizzato l'algoritmo di convoluzione.

WRITE stato in cui avviene la scrittura di una parola in memoria.

DONE stato in cui termina la computazione.

2.3 Segnali interni

Di seguito una breve descrizione dei segnali interni al componente:

state variabile di stato corrente.

next_state variabile di stato prossimo.

input registro contenente l'input da elaborare.

output registro contenente l'output elaborato.

x conserva il k-esimo bit dell'elaborazione.

q1 conserva il (k-1)-esimo bit letto. Nel primo ciclo di elaborazione è settato a 0.

q2 conserva il (k-2)-esimo bit letto. Nel primo ciclo di elaborazione è settato a 0.

times conserva il numero di parole da leggere.

count_input contatore di quante parole in input sono state lette.

count_output contatore di quante parole in output sono state scritte.

position posizione corrente nella parola correntemente processata.

3 Risultati sperimentali

3.1 Sintesi

Il componente è sintetizzabile da tool con 157 LUT e 134 FF.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
synth_1	constrs_1	synth_design Complete!								157	134

Inoltre, le simulazioni in post-sintesi eseguite risultano corrette.

3.2 Test eseguiti

Di seguito una lista di test eseguiti sul componente:

- **tb_example**:
  test di esempio fornito dal docente. Passato.
- **tb1_one_more**:
  modifica al test di esempio con l'aggiunta di una parola in più. Passato.
- **tb2_reset_in_the_middle**:
  il componente subisce un segnale di reset nel mezzo della sua computazione, riprende da capo quindi la computazione e la porta a termine correttamente. Passato.
- **tb3_start_twice**:
  il componente subisce un ulteriore segnale di start non appena ha abbassato il segnale di done senza che venga esplicitamente dato un segnale di reset. Passato.
- **tb4_zero_words**:
  primo caso limite. Il numero di parole da codificare è 0; il componente non deve leggere, né tantomeno scrivere, alcuna parola. Passato.
- **tb5_max_words**:
  secondo caso limite. Il componente deve convertire il massimo numero di parole possibile, ossia 255. Passato.

4 Conclusioni

4.1 Scelte progettuali

- Si è optato per una macchina a stati per disaccoppiare le varie operazioni e rendere la computazione il più sequenziale possibile, con un codice più leggero e facilmente comprensibile al costo di avere un ritardo di 21 cicli di clock per parola.
- Il progetto originale prevedeva 12 stati che sono stati ottimizzati e ridotti ad 8 permettendo di risparmiare 4 cicli di clock per parola.
- L'aggiornamento dello stato corrente e la codifica del prossimo stato avvengono su due fronti di commutazione del clock diversi, in modo che i processi siano eseguiti in due momenti diversi e non si facciano interferenza fra loro.
- Parola letta e parola da scrivere sono memorizzate in dei registri interni che fungono da shift registers, semplificando le operazioni di serializzazione dell'input e di composizione dell'output.