Prova finale corso di reti logiche

Alessio Petrini e Mattia Romano

Anno Accademico: 2021/2022

Indice

1 Introduzione

1.1 Presentazione

Progetto del corso di reti logiche svolto dagli studenti Alessio Petrini e Mattia Romano, codici persona rispettivamente 10729423 e 10705030.
Anno accademico: 2021/2022
Professore: Gianluca Palermo
FPGA target: Artix-7 FPGA xc7a200tfbg484-1 (quella consigliata)

1.2 Descrizione della specifica

Questo circuito ha lo scopo di trasformare un certo numero di parole da 8 bit, presenti in una memoria di tipo RAM sincrona a partire dalla cella con indirizzo 1, in un flusso continuo di singoli bit che variano ad ogni ciclo di clock, partendo da quello più significativo fino al meno.

Questi bit verranno utilizzati per generarne altri due seguendo lo schema convolutivo rappresentato dalla macchina a stati in Figura 1 la quale terrà costantemente memoria dello stato in cui si trova, resettandosi solo qualora si dovesse far partire una nuova elaborazione.

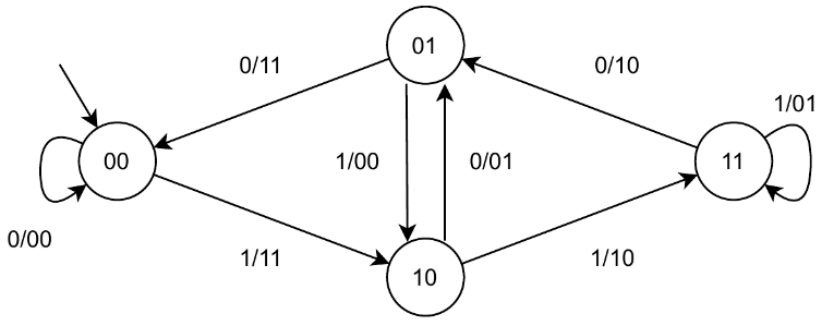

Figure 1: Convolutore

I due bit così generati verranno utilizzati per la creazione di altre parole da 8 bit inserendo il primo come bit più significativo della parola in uscita, il secondo come il secondo più significativo e così via fino ad ottenere due parole in uscita. Queste parole verranno salvate nella stessa memoria da cui si sono lette quelle in ingresso a partire dall'indirizzo 1000. Il numero delle parole in ingresso è esplicitato nella cella d'indirizzo 0 della memoria.

1.3 Esempio

Se la parola d'ingresso fosse 10100010 mi aspetterei i seguenti valori come ingressi della macchina a stati:

Ciclo di clock	1	2	3	4	5	6	7	8
u	1	0	1	0	0	0	1	0

E conseguentemente questi valori in uscita:

Ciclo di clock	1	2	3	4	5	6	7	8
p1k	1	0	0	0	1	0	1	0
p2k	1	1	0	1	1	0	1	1

Quindi le due parole da aspettarsi in uscita sono 11010001 e 11001101.

1.4 Interfaccia del componente e descrizione dei segnali

Segnali di input:

- Clock: segnale di clock a frequenza 100 GHz
- Reset: segnale che riporta il circuito nella sua condizione iniziale facendogli perdere memoria di qualsiasi evento precedente
- Start: segnale che dà il via ad una nuova elaborazione
- Input Data: segnale da 8 bit che contiene valori presi dalla memoria

Segnali di output:

- Address: segnale a 16 bit rappresenta una cella della memoria, si usa sia per la scrittura che per la lettura
- Done: segnale che viene alzato nel momento in cui l'elaborazione finisce, una volta che anche il segnale di start viene abbassato la macchina si prepara a ricevere un nuovo segnale di start per far partire una nuova elaborazione.
- Output Data: segnale a 8 bit che contiene quei valori destinati ad essere scritti in memoria
- Enable: quando viene alzato permette di leggere, al ciclo successivo, in Input Data il valore salvato nella cella con indirizzo pari a quello presente in Address
- Write Enable: quando viene alzato insieme ad Enable permette di scrivere, al ciclo successivo, nella cella con indirizzo pari a quello presente in Address il valore presente in Output Data

2 Architettura

2.1 Datapath

- Serializzatore: Questo modulo ha lo scopo di serializzare una singola parola in ingresso in un flusso continuo di singoli bit, i quali vengono inoltrati al convolutore. Si compone di un registro a scorrimento di tipo PISO (Parallel input - Serial output): la parola ricevuta in ingresso viene salvata su otto Flip-Flop e dopo ogni ciclo di clock il bit più significativo viene mandato in uscita verso il convolutore, mentre gli altri vengono spostati a sinistra e al posto del bit meno significativo viene aggiunto uno zero.Il segnale di load specifica al modulo quando è necessario memorizzare una nuova parola (presente in i_data) nei Flip-Flop, in caso il segnale sia basso il modulo procede con lo shift della parola salvata in precedenza.Il segnale init_1 si occupa di resettare il modulo inserendo tutti zeri nei vari Flip-Flop.

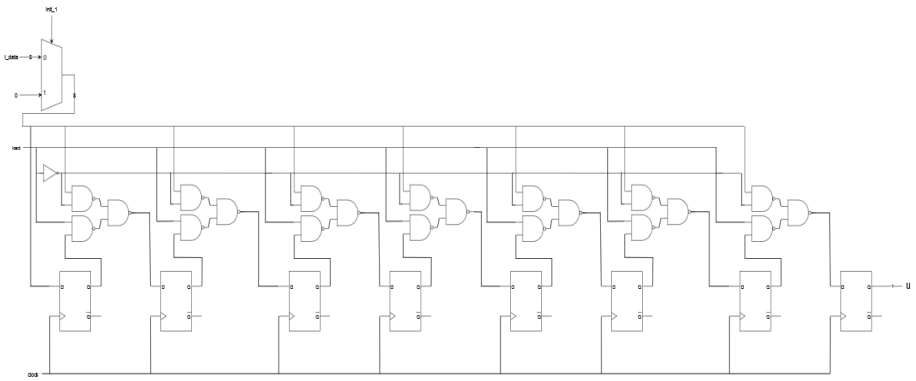

Figure 2: Serializzatore

- Convolutore: Questo modulo esegue la trasformazione del bit ricevuto dal serializzatore in 2 bit seguendo il codice convoluzionale $\frac{1}{2}$ precedentemente riportato in Figura 1, questi bit saranno poi forniti in ingresso al parallelizzatore.
  Il segnale init_2 serve a riportare il convolutore allo stato 00.

- Parallelizzatore: Questo modulo esegue il processo di parallelizzazione delle coppie di bit ricevute dal convolutore per costruire le parole in uscita. Si compone di un registro a scorrimento di tipo PIPO (Parallel input - Parallel output): gli otto Flip-Flop vengono fatti scorrere a sinistra di due bit, dopodiché i due bit ricevuti in ingresso vengono salvati sui due Flip-Flop che rappresentano i bit meno significativi, il risultato è collegato al segnale o_data.

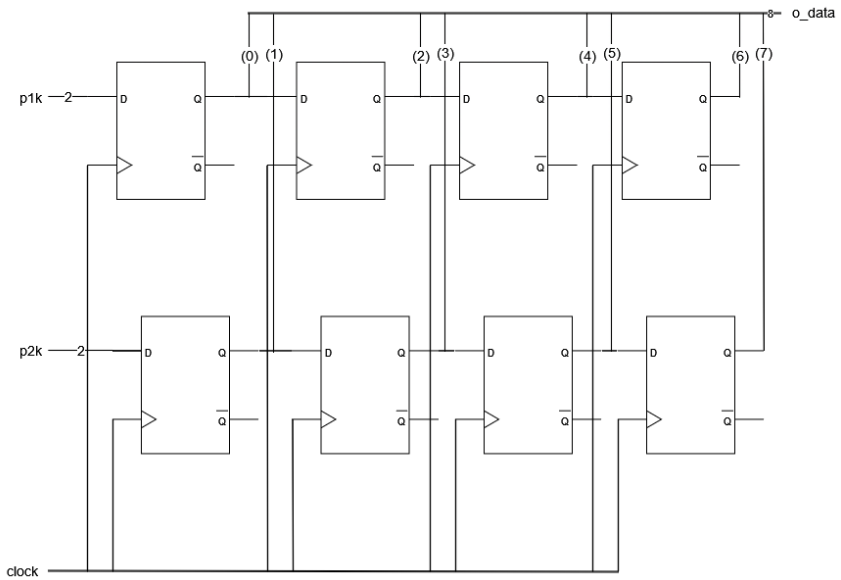

Figure 3: Parallelizzatore

- Calcolatore d'indirizzi: Questo modulo ha lo scopo di calcolare in parallelo l'attuale indirizzo di input su cui si vuole o si vorrà leggere e l'attuale indirizzo di output sul quale si vuole o si vorrà scrivere, questi indirizzi vanno in un multiplexer che si occupa di inserire in o_address l'indirizzo di input o quello di output a seconda delle esigenze. I vari segnali di load e select sono legati alla presenza dei vari multiplexer e registri.
  Per il calcolo degli indirizzi si può scegliere con i multiplexer iniziali il valore base (1000 per l'output e 0 per l'input) oppure il valore precedente incrementato di uno.

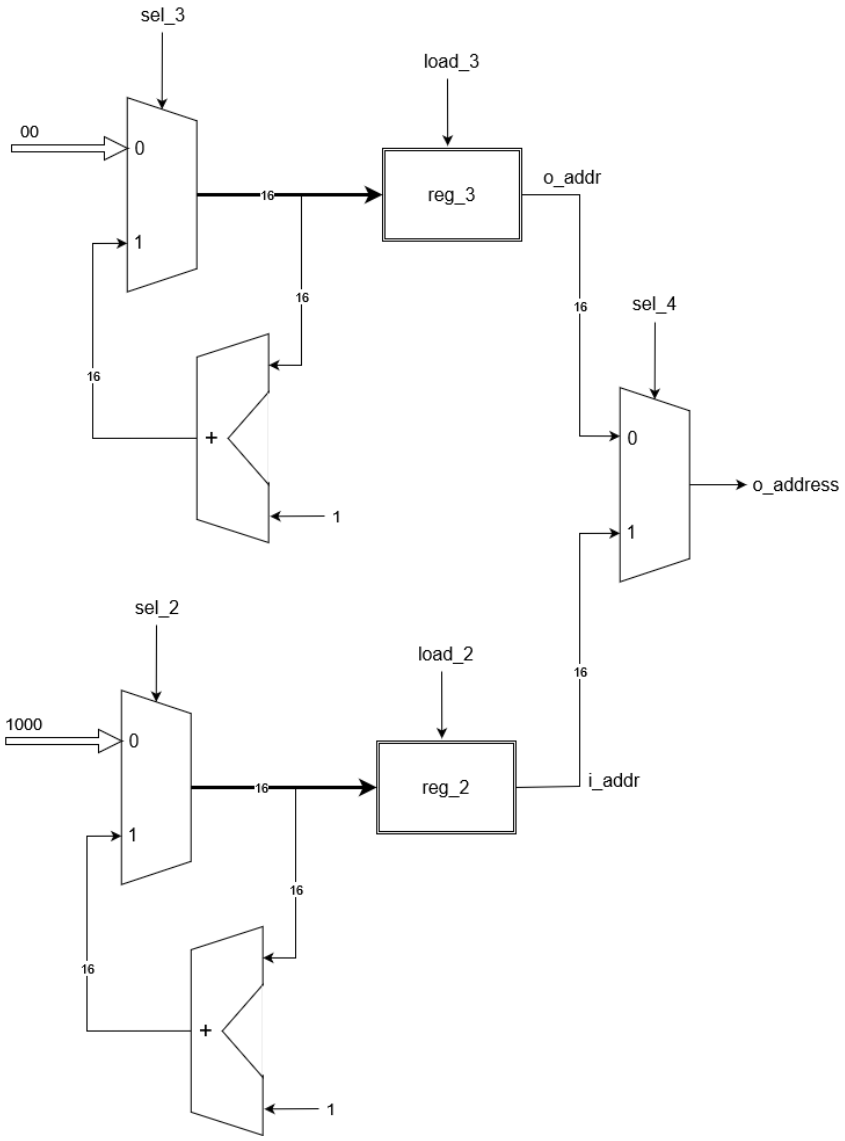

Figure 4: Calcolatore d'indirizzi

- Contatore numero di parole: componente che monitora il numero di parole rimanenti ancora da elaborare durante l'esecuzione e che fornisce, al raggiungimento di zero parole rimanenti, il segnale di DONE alto. Il multiplexer permette di salvare nel registro sia il valore iniziale proveniente da i_data che il valore precedentemente salvato decrementato di uno. All'uscita del registro un componente si chiede se il valore del registro ha raggiunto lo zero e in caso affermativo alza DONE.

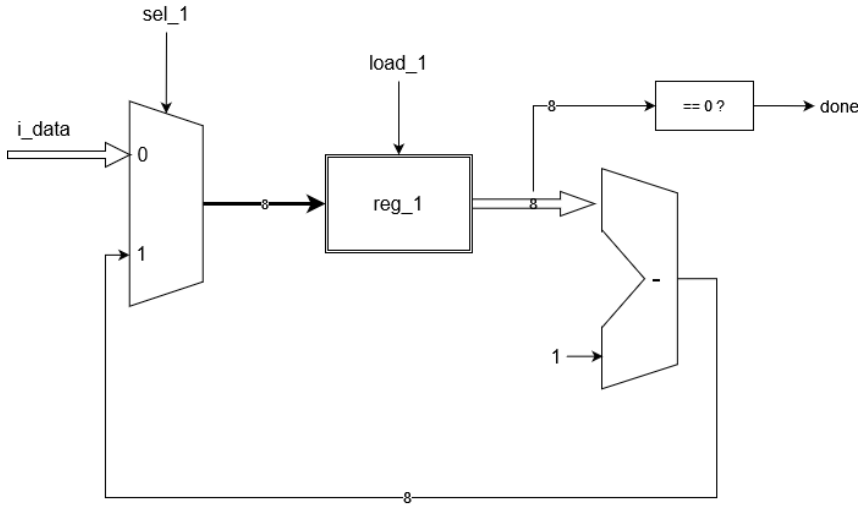

Figure 5: Contatore numero di parole restanti

2.2 FSM

La macchina a stati che governa il comportamento del componente è rappresentata in Figura 6.
Il valore di default per i vari segnali quando non specificato è sempre zero, di seguito una breve descrizione dei vari stati:

- START: carica sui registri degli indirizzi i valori iniziali e attende il segnale di START per iniziare.
- INITIALIZATION: si prepara a leggere (memoria sincrona/asincrona) il numero di parole all'indirizzo 0, resetta serializzatore, convolutore e fa scorrere in avanti l'indirizzo di lettura.
- IDLE_1: carica nel registro del numero di parole il risultato della lettura e prepara la lettura della prima parola.
- NUMBER_OF_WORDS: carica la parola letta nel registro del serializzatore, da qui si usa il segnale di done per capire se procedere verso IDLE_2 (elaborazione di una parola) o verso ENDING.
- IDLE_2: il serializzatore carica la parola e produce il primo bit in uscita.
- IDLE_3: il convolutore aggiorna il suo stato e produce in uscita i primi valori di p1k e p2k, intanto il serializzatore produce il secondo bit in uscita.
- FIRST_BIT: i primi p1k e p2k vengono serializzati e aggiunti come LSB di o_data, intanto il serializzatore produce il terzo bit in uscita e il convolutore aggiorna il suo stato producendo i secondi p1k e p2k.

- SECOND_BIT: Si procede con l'elaborazione da parte di serializzatore, convolutore e parallelizzatore, intanto si aggiorna l'indirizzo di lettura.
- THIRD_BIT: Si procede con l'elaborazione da parte di serializzatore, convolutore e parallelizzatore.
- FOURTH_BIT: Si procede con l'elaborazione da parte di serializzatore, convolutore e parallelizzatore, a questo punto o_data contiene la prima parola in uscita per cui si prepara la scrittura (o_en e o_we), si seleziona l'indirizzo di scrittura in o_address e intanto lo si aggiorna anche (dato che comunque l'effettiva modifica avviene dopo un ciclo di clock).
- FIFTH_BIT: Si procede con l'elaborazione da parte di serializzatore, convolutore, parallelizzatore e intanto ci si prepara a leggere la prossima parola.
- SIXTH_BIT: Si procede con l'elaborazione da parte di serializzatore, convolutore e parallelizzatore, a questo punto il serializzatore avrà inviato l'ultimo bit della prima parola per cui si prepara il caricamento della seconda alzando il segnale di load.
- SEVENTH_BIT: Si procede con l'elaborazione da parte di serializzatore, convolutore e parallelizzatore, in più si decrementa il valore del numero di parole già elaborate.
- EIGHTH_BIT: Si procede con l'elaborazione da parte di serializzatore, convolutore e parallelizzatore, a questo punto o_data contiene la prima parola in uscita per cui si prepara la scrittura (o_en e o_we), si seleziona l'indirizzo di scrittura in o_address e intanto lo si aggiorna anche (dato che comunque l'effettiva modifica avviene dopo un ciclo di clock), da questo stato in base al valore di done si decide se continuare l'elaborazione tornando in FIRST_BIT o se terminare andando in ENDING.
- ENDING: Si alza il segnale di done e si rimane in questo stato finché start non viene riportato a zero dopodiché si ritorna in START pronti per un'eventuale nuova elaborazione.

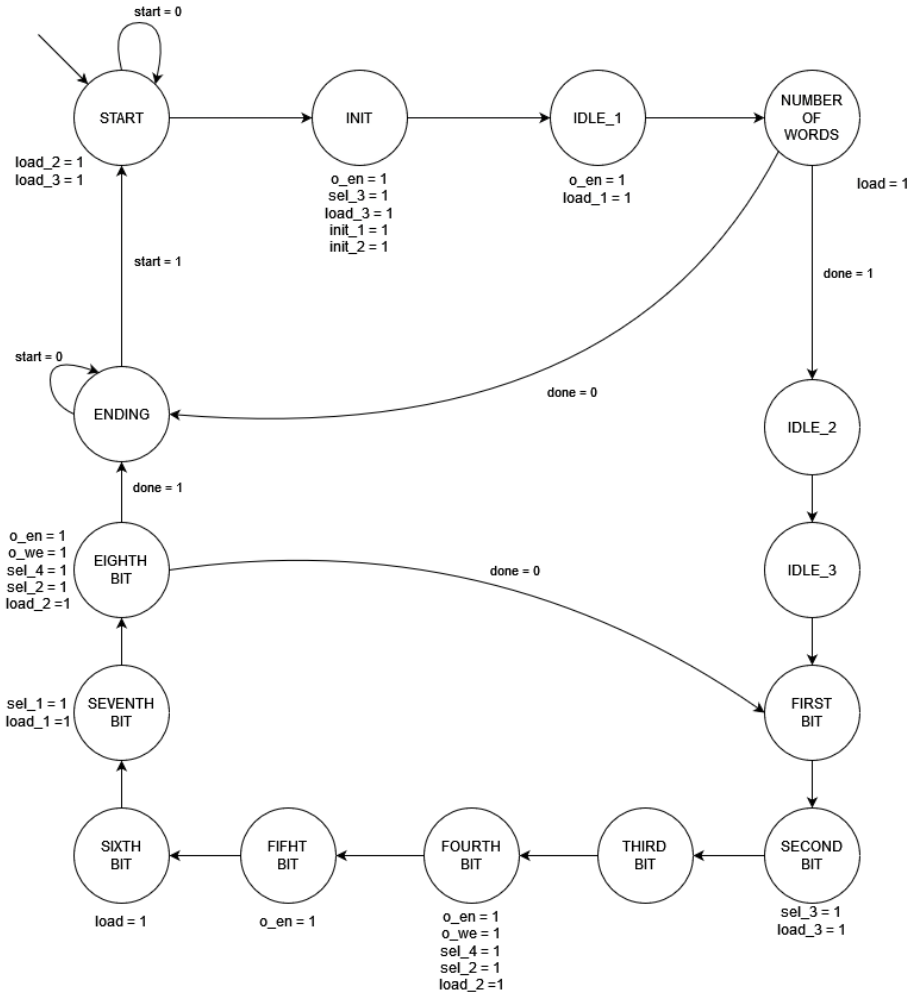

Figure 6: Macchina a stati

3 Risultati sperimentali

3.1 Sintesi

Tramite il comando report_utilization abbiamo verificato l'assenza di Latch, caratteristica fondamentale per evitare problemi in post sintesi.

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   83 |     0 |    134600 |  0.06 |
|   LUT as Logic          |   83 |     0 |    134600 |  0.06 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |   64 |     0 |    269200 |  0.02 |
|   Register as Flip Flop |   64 |     0 |    269200 |  0.02 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |    0 |     0 |     67300 |  0.00 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Figure 7: report_utilization

Tramite il comando report_timing abbiamo notato che il componente ha uno slack di 96.949 nanosecondi, questo ci dice che il clock potrebbe girare ad una frequenza significativamente maggiore.

```
Timing Report

Slack (MET) :             96.949ns  (required time - arrival time)
  Source:                 FSM_sequential_current_state_reg[3]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            convolver_current_state_reg[0]/CLR
                            (recovery check against rising-edge clock clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Path Group:             **async_default**
  Path Type:              Recovery (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        2.462ns  (logic 0.751ns (30.504%)  route 1.711ns (69.496%))
  Logic Levels:           1  (LUT5=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Figure 8: report_timing

3.2 Simulazioni

I test bench che abbiamo deciso di eseguire per testare i possibili corner case sono i seguenti:

- Caso base: lo scopo di questo test bench è quello di fornire un caso base

generico che permetta di verificare la correttezza del componente.

- Numero massimo di parole: in questo caso abbiamo deciso di spingere la macchina a stati ad effettuare il maggior numero di cicli possibili per valutare eventuali problemi di tempistiche inserendo come numero di parole da elaborare il massimo fornito dalla specifica ovvero 255. Dai risultati si evince che non c'è nessun problema di tempi o di correttezza dei risultati.
- Seconda elaborazione: qui procediamo ad effettuare due cicli di elaborazione senza fornire un segnale di reset ma semplicemente aspettando che il segnale o_done si abbassi e fornendo un secondo segnale di start come da specifica.
- Zero parole da elaborare: questo è il caso limite per cui fin da subito il numero di parole da elaborare presente nella cella di memoria 0 sia nullo. Ci aspettiamo che il componente termini istantaneamente senza scrivere nulla in memoria.
- Reset asincrono: questo test è pensato per assicurarsi che un eventuale segnale di reset fornito in un qualsiasi momento casuale dell'elaborazione riporti il componente al suo stato di partenza. Anche in questo caso il componente si comporta come previsto.

4 Conclusioni

Il componente si comporta come da specifica sia in Behavioral che in post synthesis con tempistiche ottime rispetto ai costraints forniti sulla frequenza del clock. Tutti i test bench hanno prodotto risultati positivi dimostrando la robustezza del codice ad eventuali casi limite.