Progetto di Reti Logiche, anno 2021/2022

Verdicchio Giacomo e Zarbo Nicola

30 marzo 2022

Indice

1 Introduzione

Progetto svolto da Nicola Zarbo(10677923) e Giacomo Verdicchio(10703196).

Lo scopo del progetto è stato creare un componente hardware che possa leggere dalla memoria un flusso di parole da 8 bit, convertirlo tramite un codificatore convoluzionale e scriverlo in memoria, rispettando il constraint di clock di almeno 100 ns, come da specifica di progetto.

Durante la fase di implementazione:
-abbiamo cercato di garantire la più alta leggibilità del codice e riadattabilità del modulo, nel caso di possibili modifiche e ampliamenti futuri, mantenendo comunque una buona efficienza in termini di word per clock elaborate;
-abbiamo prestato poca attenzione alla quantità di componenti (look up table, flip flop), ma ponendone molta nell'evitare un qualsiasi utilizzo (erroneo o volontario) di latch

(Il modello di fpga usato è il 'xc7a200tffq1156-1').

2 Architettura

Il nostro hardware è composto da un modulo principale (modulo 1), di cui sotto specificato, che utilizza al suo interno il codificatore convoluzionale (modulo 2) che legge e scrive dati da 2 registri gestiti dal modulo 1.

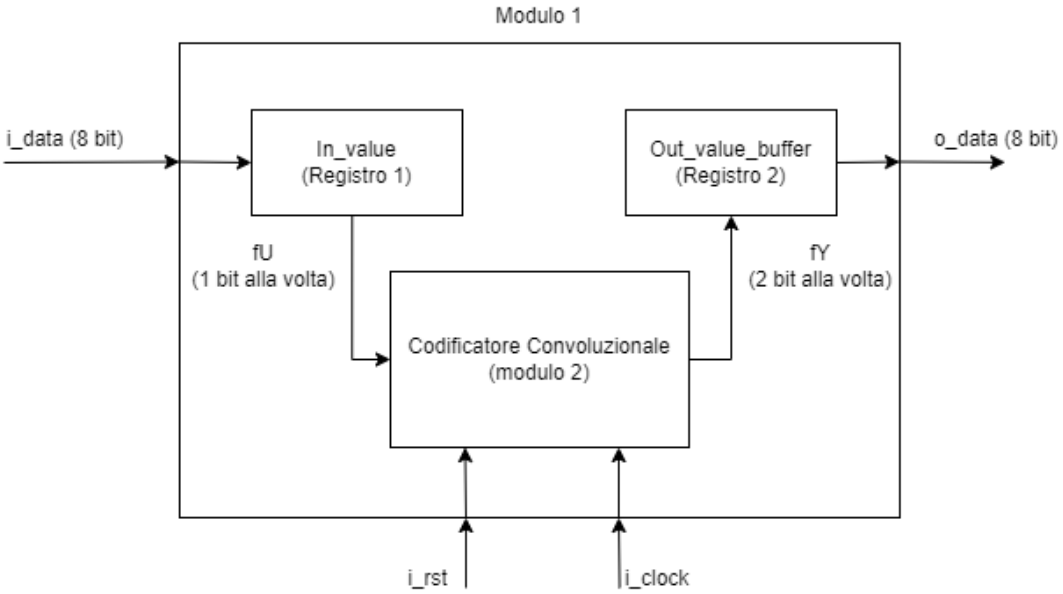


Figura 1: Schematico ad alto livello della composizione generale del modulo che mostra solo l'utilizzo del componento interno (codificatore convoluzionale.)

2.1 Modulo 1

Il modulo gestisce la lettura e scrittura da memoria e i segnali del modulo 2 (il 'codificatore convoluzionale').

2.1.1 In,Out Signals

- i_clk : segnale di clock
- i_rst : segnale di reset asincrono

- i_start : segnale di enable: '1'=> operativo, '0' => fermo (quindi il modulo viene resettato)
- i_data : bus 8 bit, dati di lettura da ram
- o_address : bus 16 bit, comunica alla ram l'indirizzo su cui eseguire lettura/scrittura
- o_done : seganale di finita elaborazione '1' => flusso elaborato/scritto in memoria, '0' =>altrimenti
- o_en : segnale di enable per ram
- o_we : segnale per comunicare alla ram quale operazione svolgere, '0'=> read, '1' => write
- o_data : bus 8 bit, dati in scritttura per ram

2.1.2 Registri

- stato_att, st_prox (8 bit): registri di stato per fsm
- in_value (8 bit): dove viene copiato la parola di 8 bit letta da i_data
- out_value_buffer (8 bit) : dove viene scritta la parola da scrivere, collegato a o_data
- in_addr (16 bit): per mantenere address per lettura e per controllo terminazione codifica
- in_a_prox (16 bit): per incrementare l'address per l'operazione di read
- out_addr, out_a_prox (16 bit):registri per mantenere e incrementare address per write
- nTerminazione (9 bit): usato per controllo terminazione, mantiene il valore della cella ram '0000' incrementato di 1 (min: 1, max 256)

2.1.3 Segnali per componente interno

- fU : flusso di bit in lettura da codificare
- fY : flusso di 2 bit in uscita da codificatore
- stop_en : segnale per fermare la macchina a stati del codificatore al di fuori degli stati di codifica

2.1.4 FSM

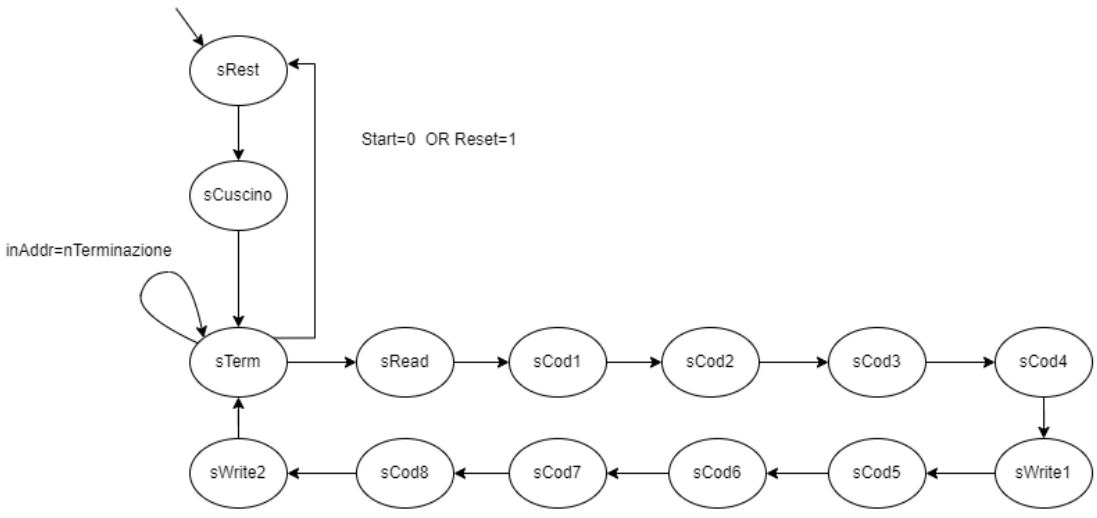


Figura 2: Rappresentazione della FSM

Descrizione stati:

- sReStart : stato di reset in cui viene anche salvato nTerminazione
- sCuscino : stato attraversato solo una volta in tutta l'operazione di codifica, serve per consentire il funzionamento quando il flusso da codificare è nullo (per ulteriori info vedere test Simulazione Minima)
- sTerm : controllo di terminazione della codifica, confronta il numero totale di parole da leggere (+ 1) con l'indirizzo di lettura successivo
- sRead : vengono forniti alla ram i segnali (o_adress, o_en, o_we) per leggere la prossima parola da elaborare
- sCod1 : stato di codifica in cui il primo bit della parola letta viene scritto in fU. Inoltre, solo in questo stato, viene anche salvata la parola appena letta da i_data nel registro in_value
- sCod2 to sCod4 : inserito in fU il nuovo bit da leggere preso da in_value, bit codificati da fY salvati in out_value_buffer nell'apposita posizione
- sWrite1 : inseriti bit da fY negli ultimi due bit del registro out_value_buffer e forniti segnali alla ram per scrivere la parola appena codificata, codificatore bloccato
- sCod5 to sCod8 : stati di codifica(funzionamento equivalente a sCod1-sCod4) usando gli utlimi 4 bit di in_value
- sWrite2 : equivalente a sWrite1

2.2 Modulo 2 : Codificatore Convoluzionale

Il codificatore convoluzionale è il modulo che si occupa dell'effettiva codifica dei dati in ingresso

2.2.1 In,Out Signals

- i_U : flusso di bit in ingresso
- i_start : segnale di enable del componente, se off il componente viene resettato
- i_rst : segnale di reset asincrono
- stop_en : segnale di enable del componente, se off viene mantenuto in standby
- o_Y : flusso di 2 bit da codifica in uscita

2.2.2 Registri

st_att, s_pros : regsitri per stati della fsm di mealy

2.2.3 FSM

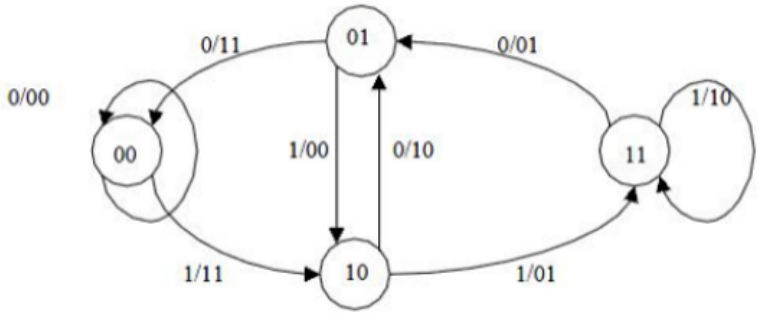


Figura 3: Disegno della macchina stati del codificatore convoluzionale

3 Risultati sperimentali

3.1 Sintesi

Di seguito abbiamo estratto le parti del report di sintesi più significativi ai fini della descrizione del modulo.

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	84	0	134600	0.06
LUT as Logic	84	0	134600	0.06
LUT as Memory	0	0	46200	0.00
Slice Registers	97	0	269200	0.04
Register as Flip Flop	97	0	269200	0.04
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
8	Yes	-	Reset
7	Yes	Set	-
82	Yes	Reset	-

```
7. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDRE     |   82 |        Flop & Latch |
| LUT1     |   35 |                 LUT |
| LUT6     |   34 |                 LUT |
| OBUF     |   27 |                  IO |
| IBUF     |   11 |                  IO |
| LUT4     |   10 |                 LUT |
| CARRY4   |   10 |          CarryLogic |
| FDCE     |    8 |        Flop & Latch |
| FDSE     |    7 |        Flop & Latch |
| LUT3     |    4 |                 LUT |
| LUT2     |    4 |                 LUT |
| LUT5     |    1 |                 LUT |
| BUFG     |    1 |               Clock |
+----------+------+---------------------+
```

```
Hierarchical RTL Component report
Module project_reti_logiche
Detailed RTL Component Info :
+---Adders :
       2 Input     16 Bit       Adders := 2
       2 Input      9 Bit       Adders := 1
+---Registers :
                   16 Bit    Registers := 2
                    4 Bit    Registers := 1
+---Muxes :
       2 Input     16 Bit        Muxes := 2
       2 Input      8 Bit        Muxes := 3
       2 Input      4 Bit        Muxes := 1
      17 Input      4 Bit        Muxes := 1
       2 Input      1 Bit        Muxes := 5
Module codificatore_convoluzionale
Detailed RTL Component Info :
+---Registers :
                    2 Bit    Registers := 2
+---Muxes :
       6 Input      2 Bit        Muxes := 1
       4 Input      1 Bit        Muxes := 2
```

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	45,744 ns	Worst Hold Slack (WHS):	0,142 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	169	Total Number of Endpoints:	169	Total Number of Endpoints:	98

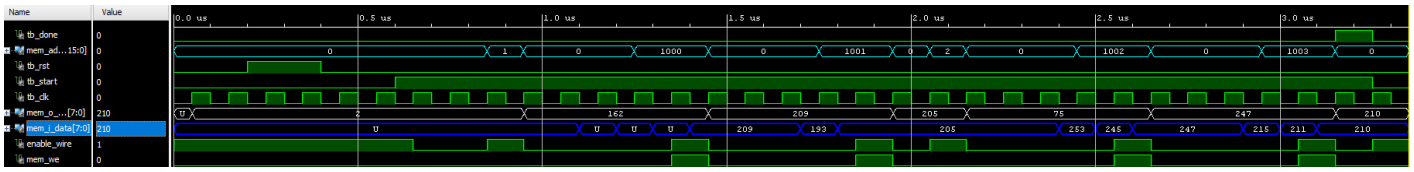
All user specified timing constraints are met.

Figura 4: Timing summery report

3.2 Simulazioni

Tutte le simulazioni testate funzionano sempre in pre e anche in post-sintesi. Di seguito sono riportate quelle significative per testare i casi limite.

3.2.1 Esempio generico (fornito dal professore)



Eseguo il programma con il testbench fornito durante il corso e verifico che tutti i segnali, soprattutto quelli in uscita, siano corretti:
-tb_done, per verificare la corretta terminazione della macchina
-mem_o_data, valori in uscita
-mem_i_data, valori letti
-mem_address, contiene l'indirizzo di memoria da cui leggere o scrivere.

3.2.2 Simulazione max

Lettura dalla RAM di 2^8 parole da 8 bit, cioè 255, e verifico che il programma gestisca correttamente tutti gli input, in particolare verificando con cura che non si realizzi un overflow nel registro di lettura e nel registro per il controllo della terminazione.

3.2.3 Simulazione min

Parto da una memoria con soli zeri e verifico che li gestisca correttamente, cioè senza scrivere in memoria nessun valore. Inoltre verifico che nessun segnale del componente vada in underflow.

Nota: abbiamo scelto di aggiungere uno stato (sCuscino) per ritardare di un clock il primo controllo di terminazione, in modo da permettere all' istruzione 'until tb_done=1' di essere eseguita cosicché venga osservato l'evento tb_done =>'1'.

3.2.4 Simulazione reset multipli

Il testbench attiva il segnale di reset più volte senza aspettare la terminazione della codifica e in seguito verifica che i dati scritti siano corretti.

3.2.5 Simulazione start multipli

Il testbench fa leggere, codificare e scrivere al componente più flussi di parole uno dopo l'altro, riattivando il segnale di start solo dopo la terminazione della codifica precedente.

4 Conclusioni

Il progetto :

- rispetta tutte le specifiche fornite sia in pre che post sintesi;
- codifica con una velocità di 12 clock per parola;
- può funzionare con un periodo di clock fino a 10ns (in simulazione su Vivado in Post Synthesis).

4.1 Note particolari modulo 1

La fsm usa molti stati equivalenti, scelta adottata per favorire la comprensibilità usando un approccio simile agli automi a stati finiti, evitando controlli complessi per la scelta del prossimo stato, tranne ovviamente in sTerm, dove viene verificata la terminazione della codifica.

4.2 Note particolari modulo 2

La codifica avviene tramite questo componente, pensato per rispecchiare il più possibile quello descritto dalla specifica e per essere facilmente utilizzabile in contesti diversi senza bisogno del Modulo 1.
Infatti questo tipo di codificatori sono usati nell'ambito delle telecomunicazioni per le trasmissioni di informazioni, quindi tale componente potrebbe essere utilizzato collegandone l'uscita direttamente ad un trasmettitore, invece di salvare il flusso codificato su una memoria, come avviene in questo progetto.