Progetto finale di Reti Logiche - Anno 2021/2022

Dario d'Abate (Codice Persona 10686115 - Matricola 935161)
Prof. William Fornaciari - Prof. Federico Terraneo

Politecnico di Milano

Indice

0.1 Introduzione

0.1.1 Obiettivo del progetto

L'obiettivo del progetto consiste nell'implementare un codificatore convoluzionale[1] con tasso di trasmissione $\frac{1}{2}$. Si tratta, perciò, di specificare un componente hardware, descritto in VHDL, che si interfaccia ad una memoria.

0.1.2 Specifica generale

Al modulo verrà fornita una sequenza continua di parole da 8 bit. Ogni parola verrà poi serializzata, in modo da generare un flusso continuo da 1 bit. Ad ogni bit si applicherà la codifica convoluzionale $\frac{1}{2}$, come descritto in Figura 1. Si dovrà, poi, attuare una deserializzazione del flusso continuo in uscita, ottenendo parole da 8 bit. Poichè la frequenza dei bit in uscita al codificatore è il doppio di quella in entrata, per ogni parola che viene fornita se ne otterranno due.

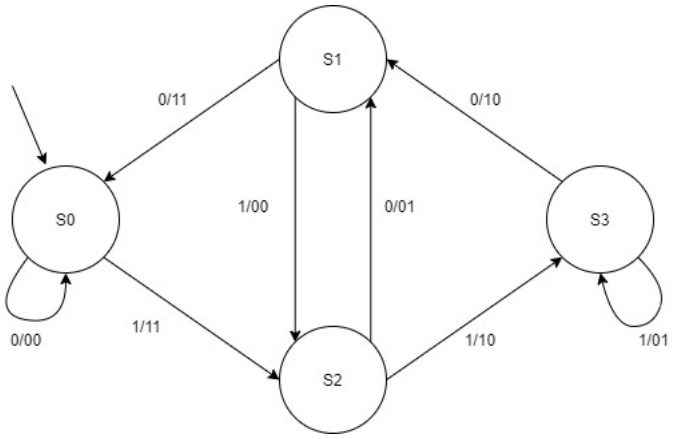

Figura 1: Codifica convoluzionale $\frac{1}{2}$

Il convolutore è, di fatto, una macchina sequenziale sincrona sensibile ad un segnale di clock e reset. È naturale quindi una sua rappresentazione mediante macchina a stati finiti, in questo caso una macchina di Mealy: Figura 1. Segue un esempio di codifica convoluzionale:

- PAROLA IN INGRESSO: 10100010
- PAROLE IN USCITA: 11010001 e 11001101

[1] La codifica convoluzionale è un tipo di codice per la correzione d'errore. Viene applicata in particolar modo nelle telecomunicazioni, ma anche in altri ambiti come ad esempio in alcuni riconoscitori testuali.

0.1.3 Interfaccia del componente

Il componente da descrivere possiede 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_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) che arriva dalla memoria in seguito ad una richiesta di lettura;
- **o_address** è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- **o_done** è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- **o_en** è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- **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 uscita dal componente verso la memoria.

0.1.4 Dati e descrizione della memoria

La memoria ha un indirizzamento al Byte e, analizzando il codice VHDL, si nota che essa è una memoria sincrona.

- All'indirizzo 0 è memorizzata la quantità di parole W da codificare, dove W $\leq$ 255;
- Il primo byte della sequenza da codificare è memorizzato a partire dall'indirizzo 1;
- Il primo byte dello stream di uscita è memorizzato a partire dall'indirizzo 1000(mille).

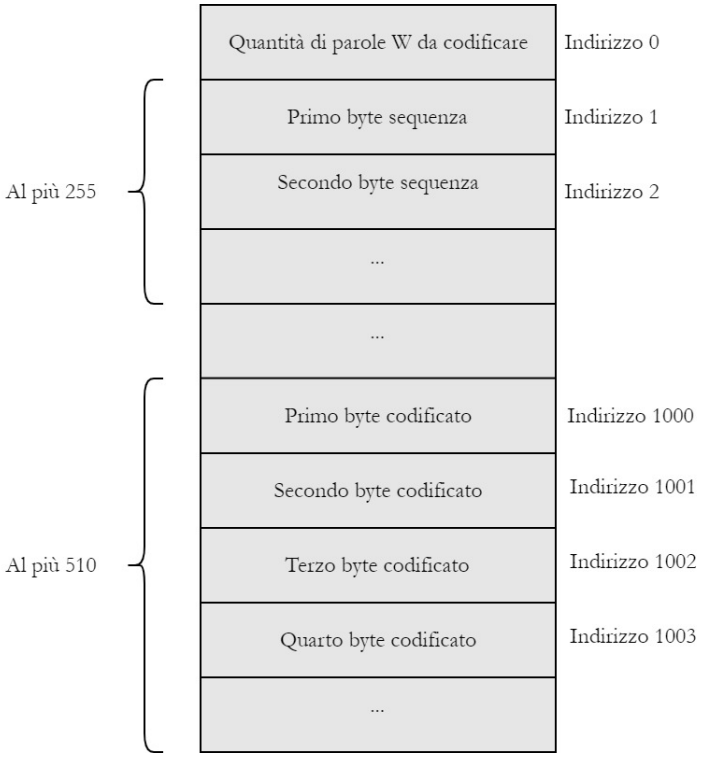

Figura 2: Rappresentazione indirizzi della memoria

0.2 Architettura

Quando il segnale i_start viene alzato, il modulo inizierà la computazione. i_start rimarrà alto fintantoché non verrà posto a 1 il segnale o_done, questo segnalerà la fine dell'elaborazione e la scrittura dello stream in memoria. Il segnale o_done verrà portato basso quando si fornirà i_start = 0, ritornando in uno stato iniziale in cui si attende il comando i_start = 1. Si nota, perciò, che il modulo è disponibile a codificare più flussi in ingresso, forniti uno dopo l'altro. Bisogna tener presente che, antecedentemente alla prima codifica, verrà alzato il segnale di i_rst, ma il modulo non verrà resettato nelle computazioni successive.
L'approccio seguito per l'implementazione del modulo è stato quello di definire un datapath e una unità di controllo. Di seguito è riportata la descrizione di entrambi gli elementi.

0.2.1 Datapath

Il Datapath si compone principalmente di due moduli: uno di essi è preposto al calcolo di o_address, mentre l'altro si occupa della computazione di o_data. Nelle figure non sono stati rappresentati i segnali i_clk e i_rst, seppur presenti nella reale implementazione.

Calcolo o_address

Come si può notare dalla Figura 3, questo modulo contiene due registri: nel registro reg4 verranno salvati ed incrementati gli indirizzi che conterranno le parole codificate, nel registro reg5 gli indirizzi che conterranno le parole da codificare.

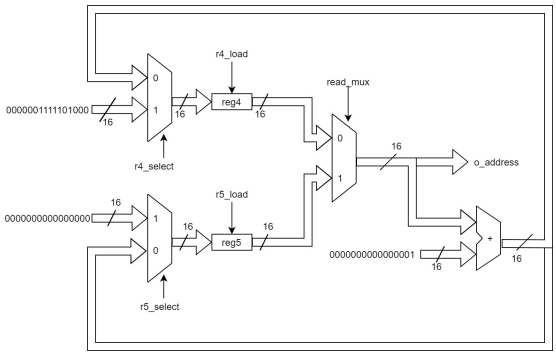

Figura 3: Modulo per calcolo di o_address

Calcolo o_data

Nella figura 8 è presentato il modulo per il calcolo di o_data. La descrizione di esso seguirà il processo logico col quale è stato progettato.

- *Gestione del numero W di parole da codificare*: nel registro reg0 si troverà la parola corrispondente a W, la quale verrà decrementata di una unità ogni volta che si leggerà una parola da memoria. É presente un segnale o_end che gestisce la condizione di terminazione dell'elaborazione.

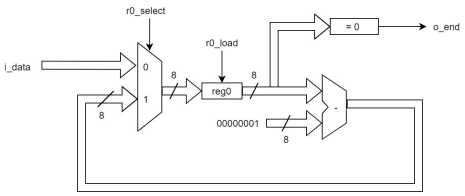

Figura 4: Modulo per gestione di W

- *Gestione della serializzazione di una parola*: si è scelto di accedere ad ogni bit della parola salvata in reg1 tramite un multiplexer, controllato da un contatore modulo 7

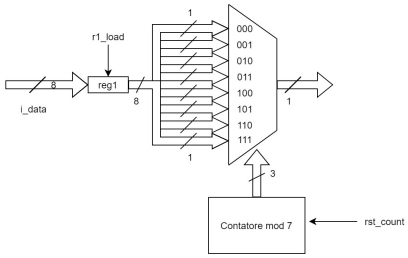

Figura 5: Modulo per serializzazione

- *Codifica convoluzionale*: questo modulo è stato implementato mediante la macchina di Mealy rappresentata in Figura 1;

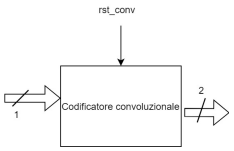

Figura 6: Modulo di codifica convoluzionale

- *Gestione della deserializzazione del flusso codificato*: si ottiene una parola concatenando fra loro 4 gruppi da 2 bit, è perciò necessario un sommatore e uno shift di 2 bit a sinistra. Il multiplexer, controllato da r3_load, viene usato per resettare il contenuto del registro reg3.

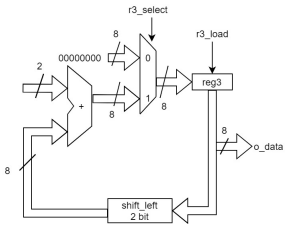

Figura 7: Modulo per deserializzazione

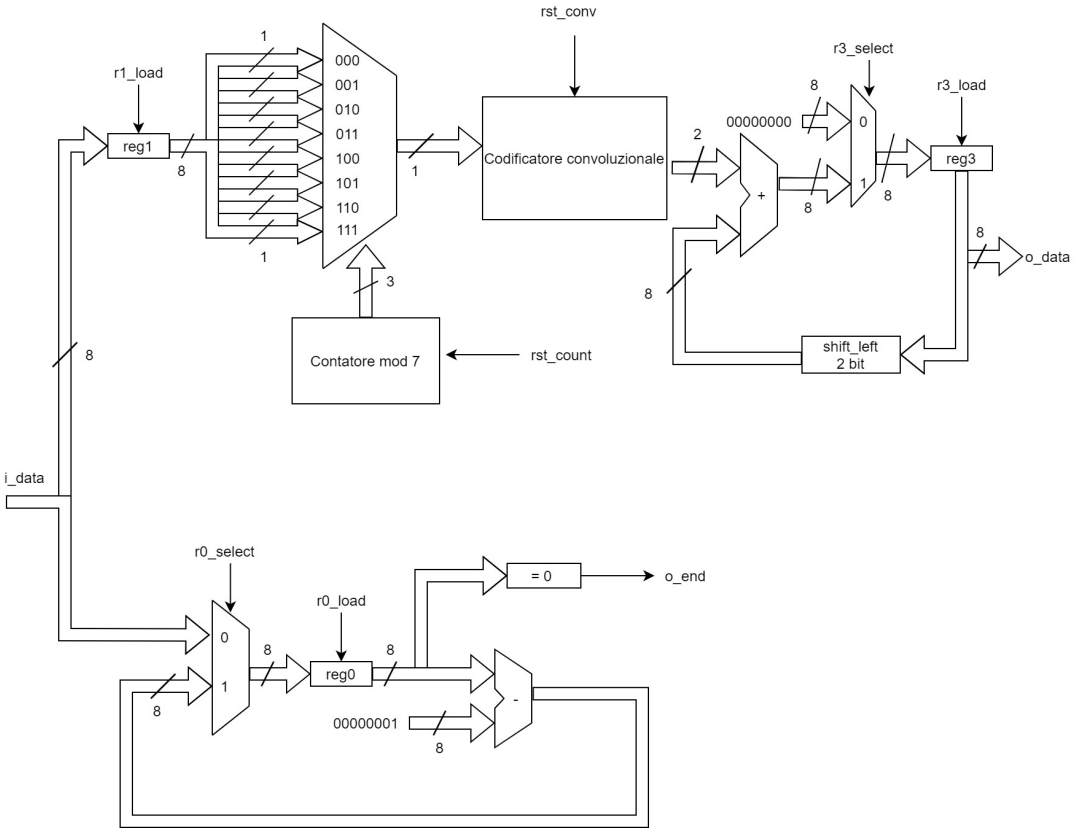

Figura 8: Modulo per calcolo di o_data

0.2.2 Unità di controllo

La macchina a stati che controlla il Datapath è una macchina di Moore, la quale è composta da 18 stati (Figura 14). Per scelta progettuale si è preferita una macchina con numerosi stati, in modo tale da avere una separazione più netta tra le fasi del ciclo di elaborazione.
Prima di passare ad una breve descrizione della macchina a stati, è doveroso fare una considerazione: leggendo il protocollo di funzionamento della memoria, si evince che essa è di tipo sincrono; le letture in memoria forniranno, perciò, il dato solamento nel ciclo di clock successivo a quello in cui avviene suddetta operazione.

- *Lettura del numero di parole W*: In questa sezione della macchina a stati si effettuano tutte le operazioni di reset del modulo e accesso al prima cella di memoria in cui è contenuto il numero di parole W.

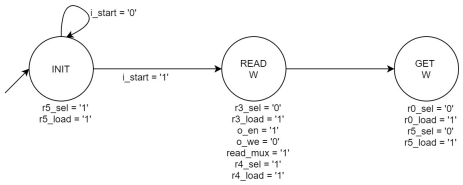

Figura 9: Sezione per la lettura di W

- *Lettura della parola da codificare*: Come prima cosa si controlla W, per capire se ci sono ancora parole da leggere in memoria. Se W ≥ 1 si passa alla codifica della parola, altimenti si dichiara terminata la computazione del flusso in entrata.

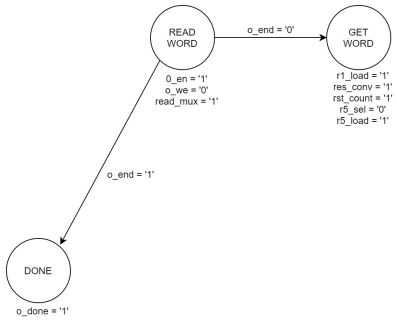

Figura 10: Sezione per la lettura di una parola

- *Codifica dei primi 4 bit della parola*: Si procede ad applicare la codifica convoluzionale alla prima metà della parola e a scrivere il risultato in memoria. Da notare che la memoria sincrona pormette la scrittura in memoria in un unico ciclo di clock.

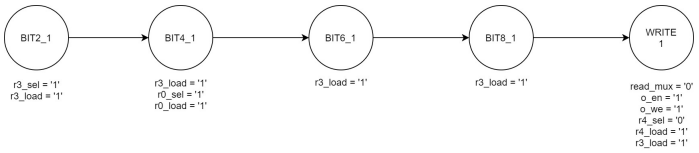

Figura 11: Sezione per la scrittura della prima parola in uscita

- *Codifica degli ultimi 4 bit della parola*: Nello stato Write1 si saranno già prodotti i primi due bit della seconda parola da scrivere in memoria. Dallo stato Bit4_2 ci sono due diramazioni per la gestione di W pari e W dispari; sebbene questa distinzione poteva essere trascurata, per scelta progettuale, si è preferita maggiore chiarezza. Nel caso in cui non ci siano altre parole da codificare, si andrà nello stato di DONE, altrimenti si passerà allo stato Bit4_1.

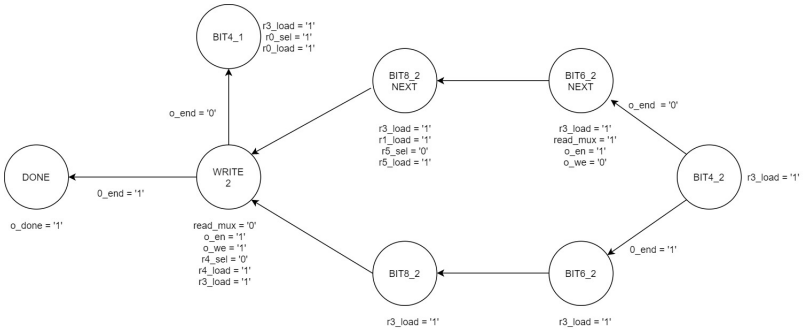

Figura 12: Sezione per la scrittura della seconda parola in uscita

- *Terminazione dell'elaborazione*: Questa parte della macchina a stati rappresenta quanto descritto nella sottosezione "Architettura". Si ribadisce che il modulo progettato è capace di codificare più flussi in entrata, uno dopo l'altro.

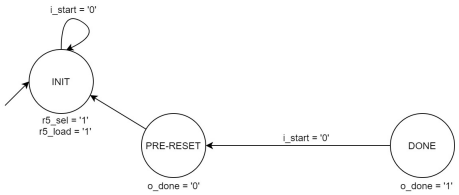

Figura 13: Gestione terminazione dell'elaborazione

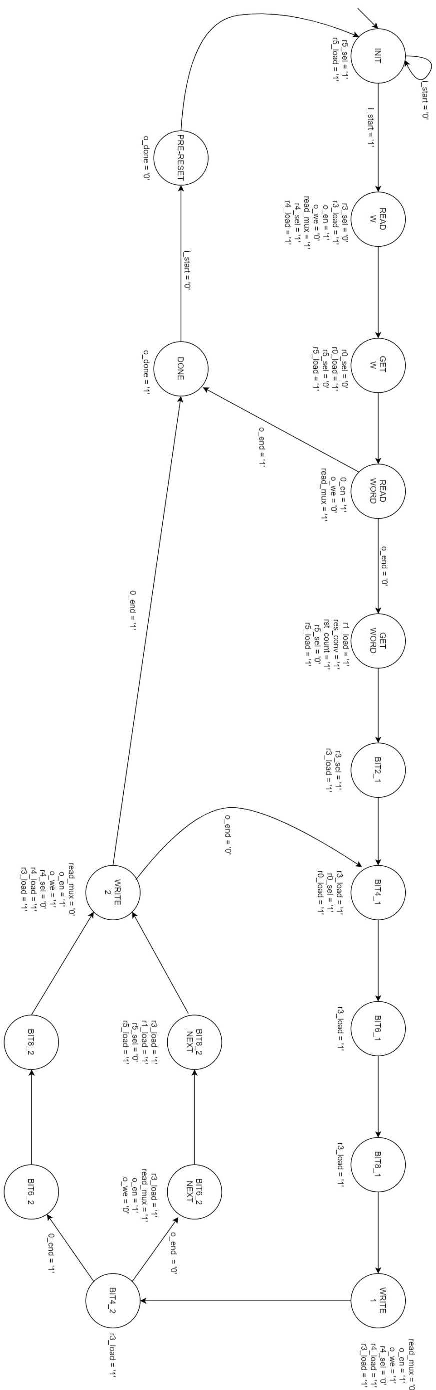

Figura 14: Macchina a stati per controllo del Datapath

0.3 Risultati sperimentali

0.3.1 Sintesi

Il modulo è stato sintetizzato mediante il software Vivado, utilizzando come FPGA target Artix-7 FPGA xc7a200tfbg484-1. Seguono i componenti della FPGA utilizzati.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	74	0	0	134600	0.05
LUT as Logic	74	0	0	134600	0.05
LUT as Memory	0	0	0	46200	0.00
Slice Registers	66	0	0	269200	0.02
Register as Flip Flop	66	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	1	0	0	67300	<0.01
F8 Muxes	0	0	0	33650	0.00

Degno di nota è il fatto che, correttamente, non sono stati inferiti Latch. Il ritardo massimo del Datapath risulta essere di 4.714 ns, con uno slack di 95.318 ns rispetto al periodo di clock richiesto di massimo 100 ns.

0.3.2 Simulazioni

Il modulo è stato sottoposto a diversi test, mediante il software Vivado, condotti sia in Behavioral Simulation che in Post-Synthesis Simulation. Durante i test, oltre alla correttezza del contenuto della memoria, è stato monitorato anche il comportamento dei segnali interni ed esterni prodotti dal componente durante tutta l'elaborazione.

Test bench di prova

Le prime simulazioni sono state eseguite utilizzando il Test bench di prova fornitoci, questo conteneva W=2.

Test mirati

Per valutare il comportamento eccezionale del modulo, sono stati forniti due Test Bench con W=0 e W=1.

Test per codifiche multiple

Il componente è stato testato per valutare il comportamento ripetto a codifiche multiple, una di seguito all'altra. Sono stati effettuate 100 codifiche successive, alcune delle quali con W=255.

0.4 Conclusioni

Il componente ha superato correttamente tutti i test specificati nelle simulazioni Behavioral e Post-Synthesis. Inoltre, come ci si aspettava, solo una piccola percentuale dell'FPGA è stata impiegata, date le dimensioni ridotte del progetto in questione. Infine, facendo riferimento a quanto riportato nella sottosezione "Simulazioni", il ritardo massimo del Datapath rientra ampiamente all'interno del margine di clock richiesto di 100 ns; questo ci suggerisce che può essere ridimensionato il periodo di clock.