Prova Finale
Progetto di Reti Logiche

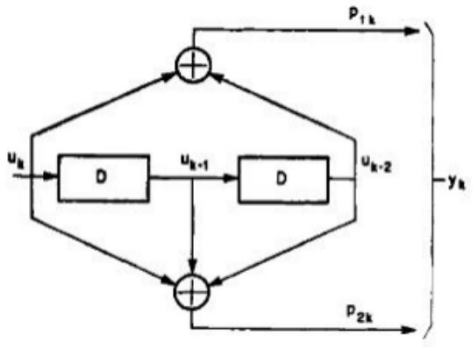
Anno 2021/2022
Prof. Fabio Salice

Mosconi Alessandro
(Codice Persona 10681624 - Matricola 932922)

INDICE

1. ***INTRODUZIONE***

Lo scopo del progetto consiste nell'implementare un componente hardware, descritto in VHDL, che riceva in ingresso una serie di parole da 8 bit, le serializzi in un flusso continuo di singoli bit i quali, una volta codificati tramite codice convoluzionale ½, vengano riuniti a formare parole da 8 bit che verranno scritte in memoria. Il primo valore letto da memoria contiene il numero di parole da codificare. Ogni parola letta verrà quindi codificata in due come scritto nel seguente schema (uk = stream di singoli bit in ingresso, yk= bit codificati in uscita)



Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$.

1.1 Memoria

L'accesso in memoria è effettuato utilizzando parole di 8 bit come segue, la lettura viene effettuata a partire dall'indirizzo 0, mentre la scrittura viene effettuata a partire dall'indirizzo 1000. L'accesso in memoria

Lettura:	
0	Numero di parole da codificare
1	Parola #1
2	Parola #2
3	….
4	….

Scrittura:	
1000	Parola codificata #1.1
1001	Parola codificata #1.2
1002	Parola codificata #2.1
1003	Parola codificata #2.2
1004	…

1.2 Interfaccia

Il componente progettato è descritto dalla 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;
```

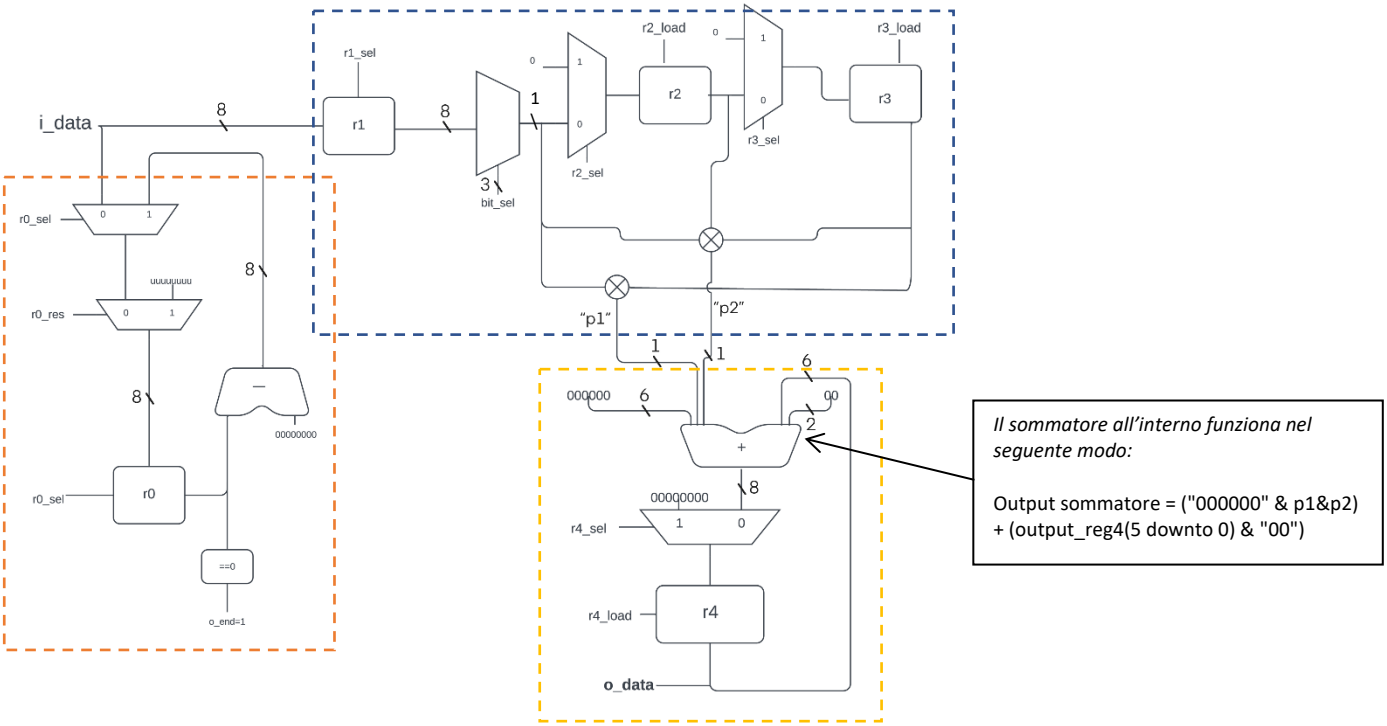
In particolare:

- il nome del modulo deve essere 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 TestBench;
- 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.

2. ***ARCHITETTURA***

È stato scelto di dividere il componente hardware in datapath e macchina a stati finiti per semplificare lo sviluppo.

2.1 Datapath:

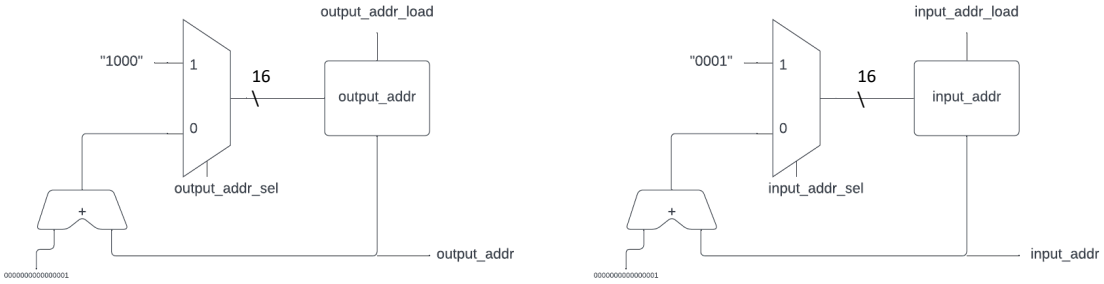


■Codificatore
■Contatore
■Normalizzatore

Il <u>contatore</u> riceve in ingresso da memoria il primo valore contenente il numero di parole da codificare e lo salva in r0. Il contenuto di questo registro viene decrementato alla fine della codifica di ogni parola, una volta arrivato a 0 viene impostato il segnale o_end a 1
Il <u>codificatore</u> riceve in ingresso una parola (salvata nel registro r1), la serializza in serie di bit tramite un multiplexer a 8 ingressi, applica la codifica descritta dalla specifica del progetto e restituisce due serie di bit codificati (p1 e p2)
Il <u>normalizzatore</u> riceve le due serie di bit e costruisce parole di 8 bit che vengono poi salvate in memoria

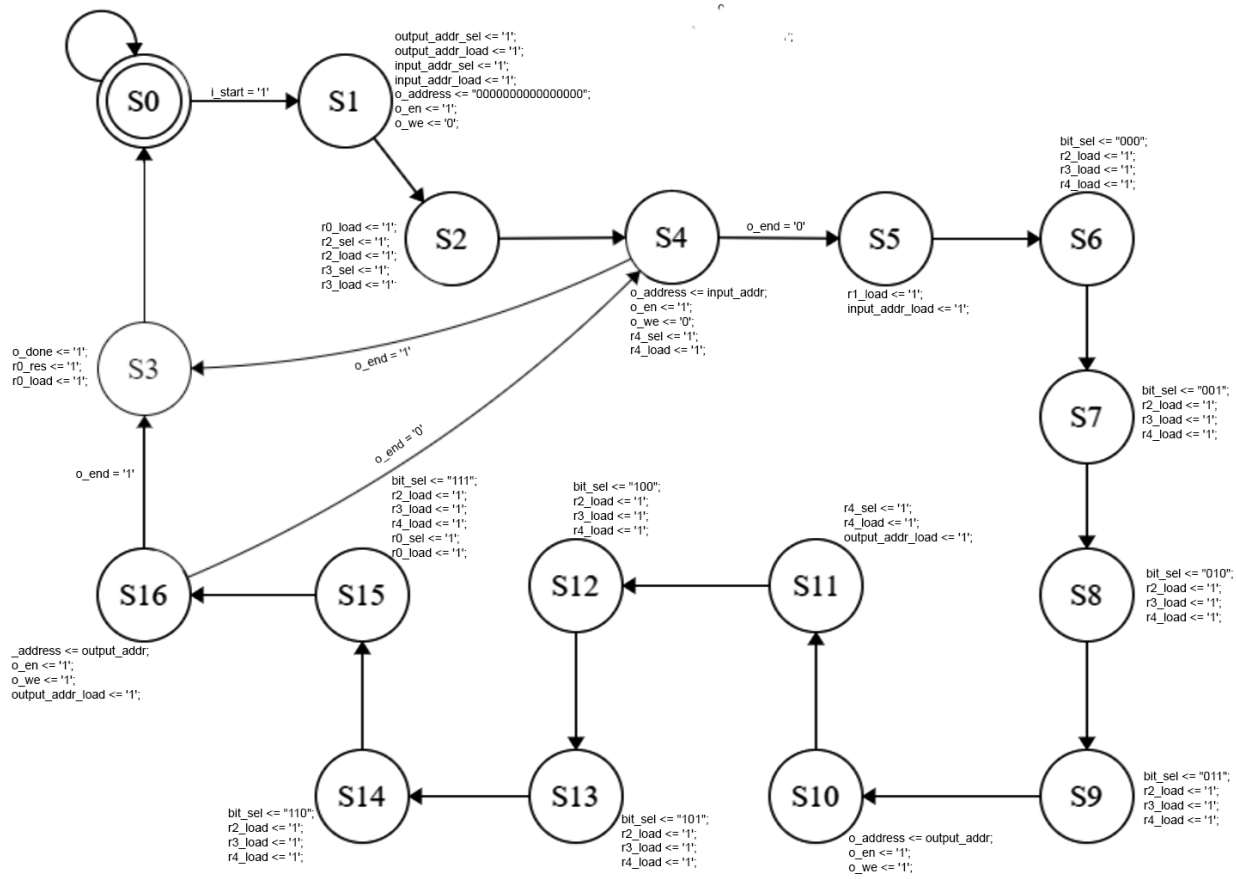
- Contenuto di r0: numero di parole da analizzare (analizzata la prima, questo registro verrà decrementato di 1)
- Contenuto di r1: parola di 8 bit da analizzare
- Contenuto di r2 e r3: bit precedenti del flusso, se non presenti contengono 0
- Contenuto di r4: parola di 8 bit codificata da scrivere in memoria

All’interno del datapath sono stati progettati due componenti che realizzano i puntatori agli indirizzi delle celle di memoria da leggere e scrivere



- Contenuto di output_addr: indirizzo di 16 bit su cui scrivere la parola codificata
- Contenuto di input_addr: indirizzo di 16 bit da cui leggere la parola da codificare

2.2 Macchina a stati finiti:

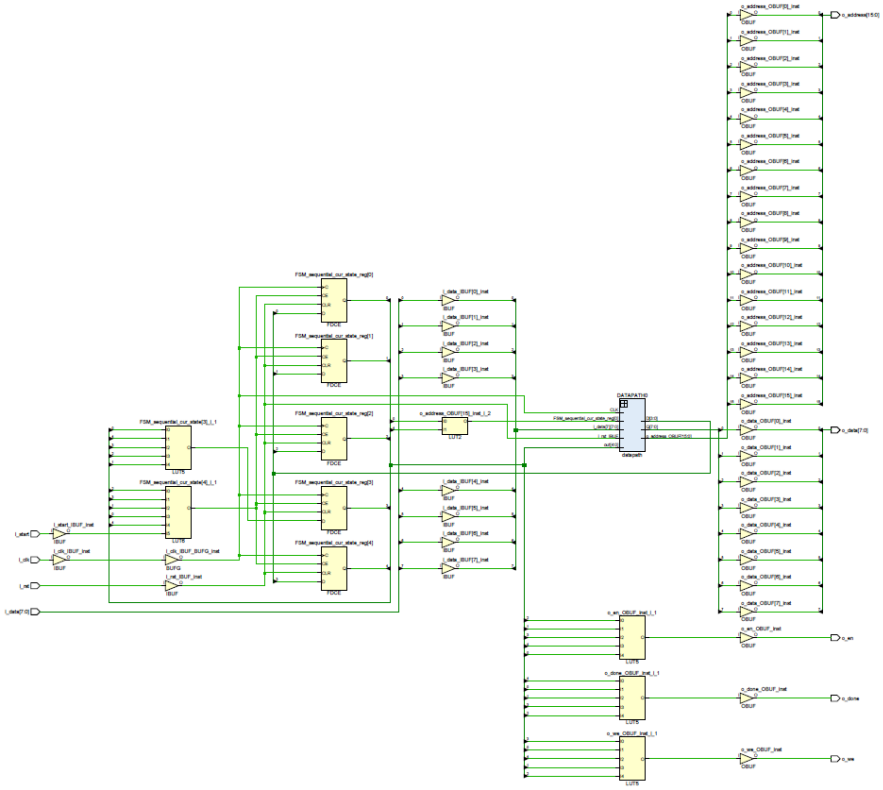


È stato utilizzato il segnale o_end per verificare se il numero di parole da codificare sia = 0.
Output_addr e input_addr sono utilizzati per recuperare dal datapath i valori progressivi dei puntatori alle celle di memoria, rispettivamente, di output e di input
I segnali non specificati sono inizializzati a 0 di default

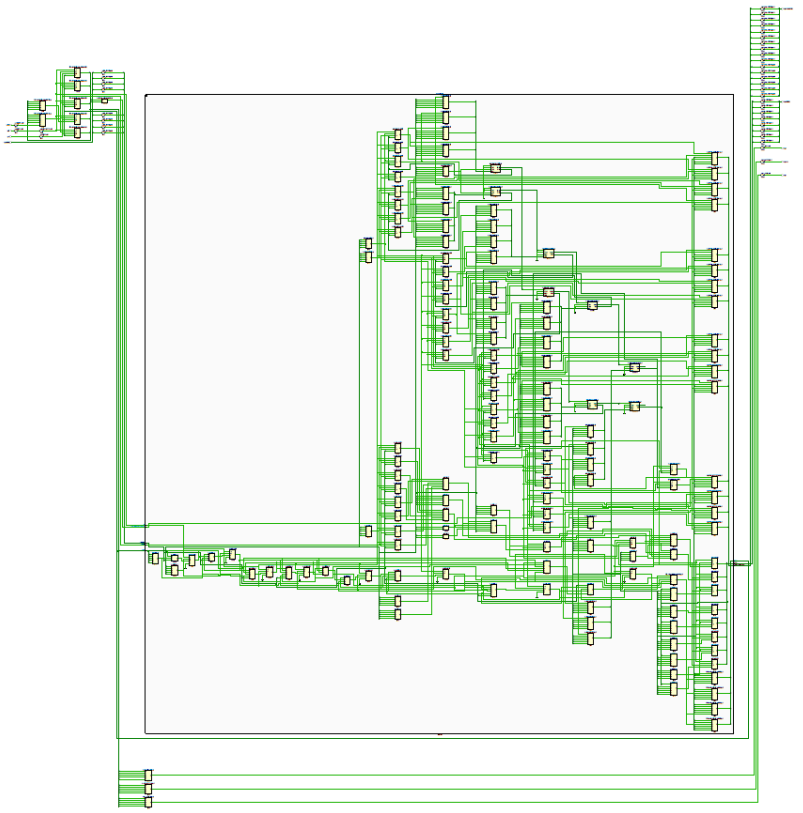
La lettura di un valore in ingresso avviene in due differenti stati: prima si effettua la richiesta e successivamente il dato viene memorizzato rendendolo così disponibile. Questo è dovuto alla specifica del progetto che richiede di realizzare un componente che lavori con una memoria sincrona.

- Lo stato S1 acquisisce dall'indirizzo 00000000 della memoria il numero N di parole che dovranno essere convertite e imposta gli indirizzi di scrittura e lettura rispettivamente a 1000 e a 1.
- Lo stato S2 salva il valore di N in r0 e salva in r2 e r3 il valore 0.
- Dallo stato S4, in caso il numero di parole da convertire N sia 0 (o_end=1), si passa allo stato S3. In caso contrario (o_end=0) viene acquisita la prima parola e salvata in r1 passando poi allo stato S5.
- Lo stato S5 permette di incrementare il valore contenuto in input_addr.
- Lo stato S3 fa terminare il processo ponendo o_done=1.
- Negli stati da S6 a S9 vengono presi in analisi i primi 4 bit della parola da convertire aggiornando i registri r2, r3 e r4.
- Negli stati S10 e S11 il valore codificato, contenuto in r4, viene scritto in memoria e ne viene azzerato il registro.
- Negli stati da S12 a S15 vengono presi in analisi gli ultimi 4 bit della parola da convertire, in particolare allo stato S15 viene decrementato il valore salvato nel registro r0 (che ora conterrà il numero di parole da convertire N -1).
- Nello stato S16 viene scritto in memoria il valore precedentemente codificato e, nel caso in cui il registro r0 sia stato decrementato a 0 (sono state codificate tutte le parole di ingresso previste), il registro viene resettato e il processo viene terminato passando allo stato S3 che pone o_done =1. In caso le parole da analizzare non siano ancora finite (o_end > 0) si riparte dallo stato S4.

2.3 Schema dell'implementazione



Scheme with datapath collapsed



Scheme with all components expanded

3. RISULTATI SPERIMENTALI

3.1 Sintesi

Il componente è correttamente sintetizzabile e implementabile con un totale di 91 LUT, 63 FF e senza la presenza di alcun Latch

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	91	0	134600	0.07
LUT as Logic	91	0	134600	0.07
LUT as Memory	0	0	46200	0.00
Slice Registers	63	0	269200	0.02
Register as Flip Flop	63	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

3.2 Simulazioni

Per verificare il corretto funzionamento del componente, oltre ad averlo testato con i diversi testbench forniti dall'insegnante, è stato utilizzato un test che, utilizzando una ram scritta su file, esegue un test con 10000 differenti casi di prova costruiti grazie ad un generatore sviluppato con linguaggio python.

È stata verificata la correttezza del progetto anche nei seguenti casi limite:

- sequenza minima (prima cella contenente il numero di parole analizzate = 0)
- test contenente più reset successivi (test che verifica il corretto funzionamento del componente nel caso in cui un processo venga interrotto da un segnale di reset e un altro venga avviato)
- sequenza massima (255 parole consecutive)
- test con più sequenze di parole da codificare in successione

Esempio di testbench effettuato:

input:	
0	2
1	162
2	75
3	….
4	….

output:	
1000	209
1001	205
1002	247
1003	210
1004	…

Esempio corner case - sequenza minima:

input:	
0	0
1	…

output:	
1000	…

4. CONCLUSIONI

4.1 Scelte progettuali

Si è deciso di utilizzare un multiplexer a 8 ingressi per realizzare la serializzazione da parola di 8 bit a serie di singoli bit.
L'utilizzo di due distinti componenti per realizzare i puntatori alle celle di lettura e di scrittura è dovuto al fatto che per ogni parola letta, ne vengono scritte due in celle consecutive. Questa decisione ha concesso semplicità di progettazione e una maggiore chiarezza nell'architettura.