Progetto di Reti Logiche

Antonio Augello 10817792 Lorenzo Baggi 10846828

A.A. 2024 - 2025

POLITECNICO
MILANO 1863

Contents

1 Introduzione

1.1 Descrizione generale

L'obiettivo della Prova Finale di Reti Logiche dell'Anno Accademico 2024-2025 è la realizzazione di un modulo hardware per l'elaborazione e il filtraggio di una sequenza di byte presenti in memoria a partire da alcuni parametri dati. In particolare, il filtro deve presentare l'output come una sequenza di byte da apporre in memoria a seguito dell'input letto.

1.2 Presentazione delle specifiche

A seguito di un comando di start, al modulo è presentato l'indirizzo di memoria da cui iniziare l'elaborazione. I primi 2 byte rappresentano la lunghezza dell'ingresso (che chiameremo **W**), e quindi anche la lunghezza dell'uscita, dato che come già detto per ogni byte in input corrisponde un byte in uscita.
Il bit meno significato del byte che segue W rappresenta la tipologia di filtro da applicare alla sequenza in ingresso, che d'ora in poi chiameremo **S**.

- con S = 0, viene applicato un filtro di terzo ordine, con $j \in$ [-2, 2], normalizzato con n = 12
- con S = 1, viene applicato un filtro di quinto ordine con $j \in$ [-3, 3], normalizzato con n = 60

In base al valore di S, vengono scelti gli opportuni coefficienti, C_j, con $j \in [-3, 3]$, presenti nei 14 byte successivi
E' quindi possibile esplicitare la funzione del filtro differenziale da applicare come segue

$$f'(i) = \frac{1}{n} \sum_{j=-l}^{l} C_j \cdot f[j+i]$$

dove

- $f'(i)$ rappresenta il valore di uscita relativo all'i-esimo ingresso
- n il fattore di normalizzazione dipendente dal grado del filtro
- C_j il j-esimo coefficiente letto dalla memoria
- l : variabile libera fra $\in$ [-l, l]
- i : indice dell'ingresso

1.3 L'interfaccia del filtro

Come da specifica, di seguito è presentata l'interfaccia del componente in linguaggio VHDL

Listing 1: Interfaccia

```
entity project_reti_logiche is
 port (
     i_clk : in std_logic;
     i_rst : in std_logic;
     i_start : in std_logic;
     i_add : in std_logic_vector(15 downto 0);

     o_done : out std_logic;
     o_mem_addr : out std_logic_vector(15 downto 0);
     i_mem_data : in std_logic_vector(7 downto 0);
     o_mem_data : out std_logic_vector(7 downto 0);
     o_mem_we : out std_logic;
     o_mem_en : out std_logic
 );
end project_reti_logiche;
```

Di seguito una breve descrizione per ciascun segnale

- `i_clk` : segnale del clock, e' un'onda quadra con duty-cycle 50%
- `i_rst` : segnale di reset
- `i_start` : segnale di start del modulo
- `i_add` : indirizzo di memoria a cui è presente il primo byte valido
- `i_done` : segnale che comunica la fine dell'elaborazione
- `o_mem_addr` : indirizzo della cella di memoria a cui si vuole accedere, sia in fase di scrittura che in fase di lettura
- `i_mem_data` : dato fornito dalla memoria in fase di lettura
- `o_mem_data` : dato che si vuole scrivere in memoria
- `o_mem_we` : segnale di write enable, per abilitare la scrittura in memoria
- `o_mem_en` : segnale di enable, per abilitare l'utilizzo della memoria

1.4 Gestione della memoria

La memoria è formata da celle di 1 byte, fino a un massimo di 65535 indirizzi validi. A partite dall'indirizzo indicato in `i_add` sono presenti i dati utili per la realizzazione del filtro, che sono disposti nel seguente modo

- In posizione `i_add` e `i_add + 1` sono presenti i valori di K_1 e K_2, che concatenati danno luogo al valore `K`, il numero di ingressi su cui il filtro dovrà eseguire la funzione descritta in precedenza
- In posizione `i_add + 2` è presente il byte `S`, il cui bit meno significato dà l'informazione sulla tipologia di filtro differenziale da applicare.
- Da `i_add + 3` a `i_add + 17` ci sono i coefficenti C_j, con `S = 0` sono utlizzati i primi 7 byte, altrimenti gli ultimi 7
- Infine da `i_add + 18` a `i_add + 17 + K` ci sono i `W` byte da elaborare

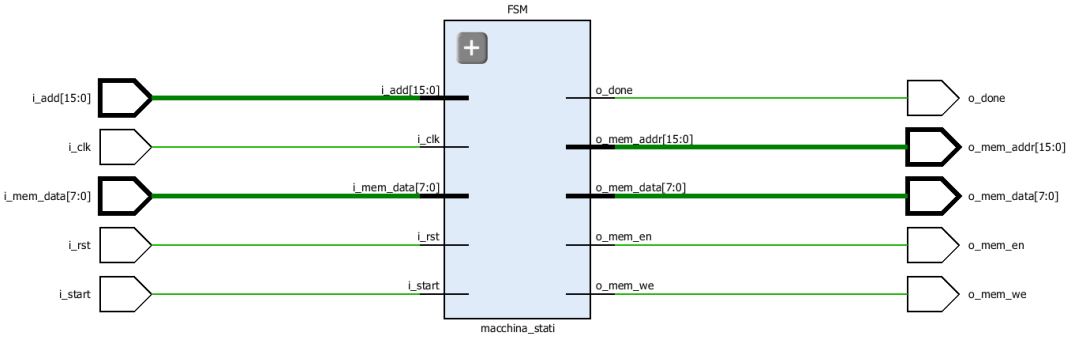
Offset	**Dato**
0	K_1
1	K_2
2	S
3	C_1
...	...
17	C_{14}
18	W_1
...	...
17 + K	W_k

1.5 Segnali dell'interfaccia

Come da specifica, la fase di acquisizione dati e elaborazione degli stessi avviene quando viene alzato il segnale `i_start` e fornito un valido indirizzo in `i_add`. Da notare che per far funzionare correttamente il dispositivo, esso deve essere prima inizializzato con un segnale di reset, qualora arrivasse un comando di start, senza che essere stato preceduto da uno reset almeno uno volta, il comportamento del filtro non e' determinato. L'utente potrà leggere la sequenza in uscita solo dopo aver osservato che il dispositivo stesso abbia alzato il segnato `o_done`, comunicando la riuscita elaborazione.

2 Design

2.1 Architettura



Abbiamo deciso di sviluppare il componente in un'unica entity stando però attenti a non sacrificare la leggibilità del codice o la sua chiarezza. Per questo abbiamo cercato di creare quanti più process possibili, sia sincroni che sequenziali.

- E' presente il processo sicrono che gestisce la Macchina a Stati Finiti (FMS), il cui diagramma degli stati è specificato in seguito. Questo processo si occupa, in base allo stato corrente e a pochi altri parametri, la prossima operazione che il dipositivo deve eseguire
- Per ogni registro è presente un processo che si occupa esclusivamente di gestirlo con i dati corretti. Questo è valido sia per i registri nei quali vengono salvati i dati in input utili per l'elaborazione del prossimo dato in uscita, sia per il salvataggio dei risultati parziali utili durante il calcolo della funzione specificata in precedenza
  L'unica eccezione è durante la fase di inizializzazione dei parametri `K`, `S`, `C`, in cui questi 3 dati vengono salvati nei relativi registri in un unico process.
- E' presente un processo combinatorio che si occupa di gestire il dialogo con la memoria, quindi di caricare su `o_mem_addr` l'indirizzo corretto, che corrisponde a
  - `address` in fase di lettura
  - `address` + offset in fase di scrittura

  alzare `o_mem_en` e, in base all'operazione richiesta, abilitare o meno l'opzione di scrittura con `o_mem_we`
- E' presente un process combinatorio per la gestione di `o_done`, che notifica quando un'elaborazione è terminata

2.2 Segnali utilizzati

Prima di procedere con l'elenco dei segnali usati dal filtro si fa notare la creazione di due tipi scecifici per gli array, entrambi di lunghezza 7

```
type vettore_type is array(0 to 6) of std_logic_vector(7 downto 0);
type intermedio_type is  array(0 to 6) of std_logic_vector(17 downto 0);
```

Table 1: Segnali utilizzati

Nome	Tipo	Informazioni
`state`	`stateType`	Memorizza lo stato della FSM

Nome	Tipo	Informazioni
`lettura`	`letturaType`	Indica in quale registro salvare il dato letto dalla memoria
`address`	`std_logic_vector(15 downto 0)`	L'indirizzo di memoria dove risiede il prossimo dato da analizzare
`K1`	`std_logic_vector(7 downto 0)`	K1, gli 8 bit più significativi di K
`K2`	`std_logic_vector(7 downto 0)`	K2, gli 8 bit più significativi di K
`K`	`std_logic_vector(15 downto 0)`	K, il numero di dati da elaborare
`S`	`std_logic`	Tipo di filtro, '0' per filtro di ordine 3, '1' per filtro di ordine 5
`C`	`vettore_type`	in C_i è presente l'iesimo coefficiente con $i \in [0, 6]$
`nums`	`vettore_type`	Lo shift register dei numeri in ingresso da analizzare
`contaC`	`std_logic_vector(2 downto 0)`	Contatore da 0 a 6 per la lettura dei coefficienti
`contaNum`	`std_logic_vector(2 downto 0)`	Contatore da 4 a 6 per la lettura dei primi 3 numeri da analizzare
`moltiplicati`	`intermedio_type`	Risultati parziali, dati dal prodotto di `nums` e `C`
`sommato`	`std_logic_vector(18 downto 0)`	Somma di `moltiplicati`
`sommaDivisioni`	`std_logic_vector(18 downto 0)`	Risultato intermedio della normalizzazione
`numeroFinale`	`std_logic_vector(18 downto 0)`	Risultato della normalizzazione
`remaining`	`std_logic_vector(15 downto 0)`	Contatore decrementale da K a 0, che inidica la quantita di numeri ancora da analizzare

2.3 Lettura dalla memoria

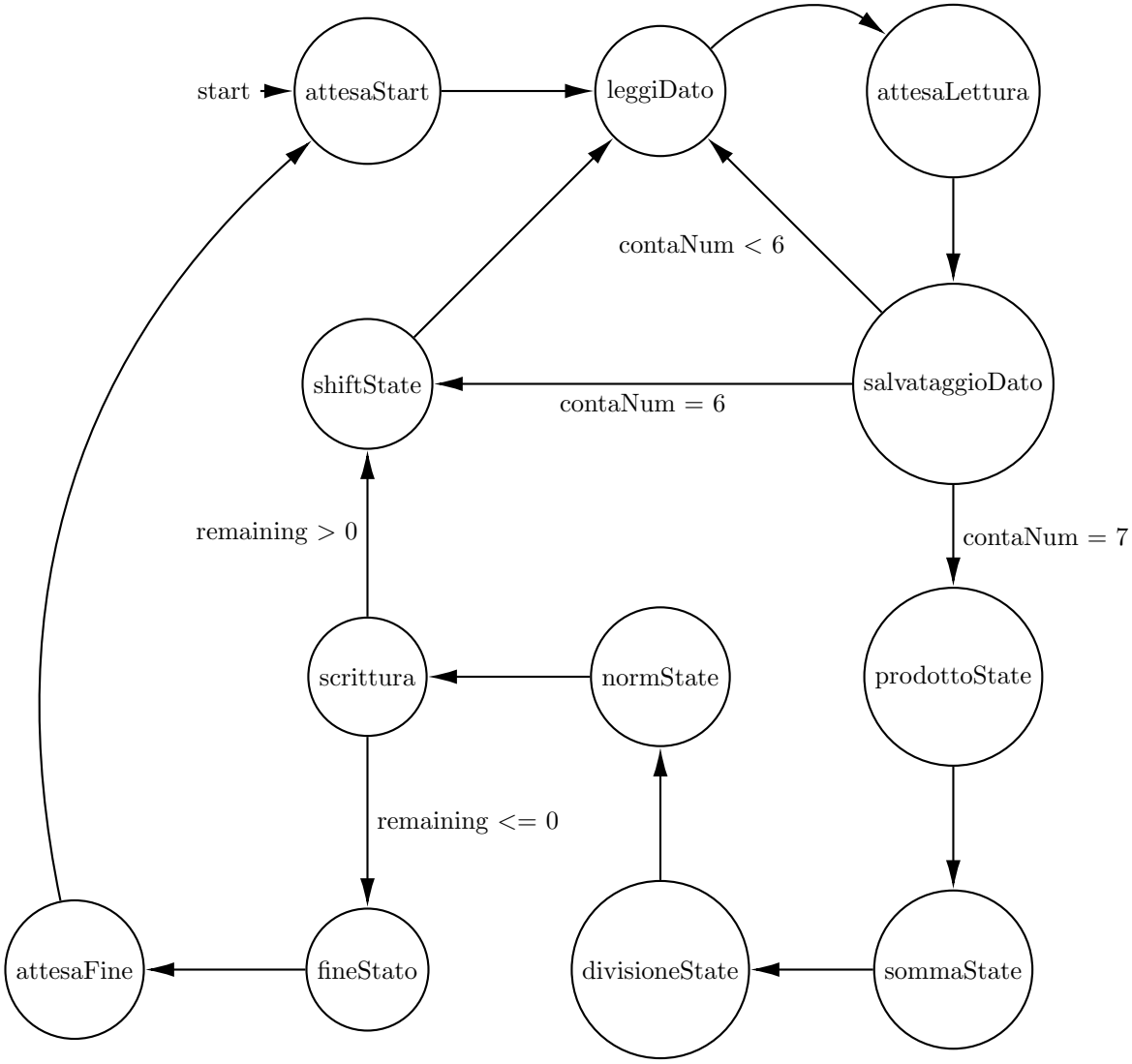
Il dato letto dalla memoria viene salvato nel relativo registro grazie al segnale `lettura`, Questo e' un segnale particolare in quanto è tipizzato `letturaType`, unica altra enumerazione oltre a `stateType`.
Quando il dispositivo è in attesa che l'elaborazione cominici, `lettura` è posto a `leggiInizio` Di seguito sono elencati i campi possibili, con il relativo significato e numero di iterazioni

Table 2: `lettura`

Nome	Dato Salvato	Numero di letture
`leggiK1`	`K1`	1
`leggiK2`	`K2`	1
`leggiS`	`S`	1
`leggiC`	`C`	7
`leggiPrimi`	`i primi 3 numeri da analizzare`	3
`leggiNum`	`gli altri numeri da analizzare`	`K - 3`

2.4 Macchina a stati Finiti

Il dispositivo implementa una macchina a stati finiti (FSM) per la corretta gestione dell'input e l'elaborazione. Di seguito è presentato il suo diagramma degli stati



2.5 Descrizione degli stati

2.5.1 attesaStart

Il disposivo ha appena ricevuto il segnato il reset oppure un'elaborazione è terminata. In ogni caso il componente è pronto per ricevere il comando di start e iniziare una nuova sequenza di input da filtrare. In questo stato tutti i registri sono resettati.

2.5.2 leggiDato

Viene chiesto un dato alla memoria, sia che si tratti di un segnale per la fase di inizializzazione che un numero da convertire. La macchina a stati scrive in `o_mem_addr` l'indirizzo dove risiede il dato richiesto e imposta alto il segnale `o_mem_en` in modo da abilitare la memoria

2.5.3 attesaLettura

E' uno stato di attesa in cui il componente attende che il dato richiesto alla memoria in `leggiDato` sia disponibile per essere usato. E' un vincolo imposto dalla lettura della memoria stessa.

2.5.4 salvataggioDato

Il dato letto dalla memoria viene salvato in un apposito registro, scelto in base al valore corrente di `lettura`. Questo è lo stato dove viene aggiornato il valore di `lettura`, in modo tale che alla prossima iterazione il dato letto vengo salvato nel registro corrispondente.
Viene aggiornato il valore di `address`, minimizzando il numero di letture. Normalmente questo valore viene incrementato, ma in base al tipo di filtro, solo 7 dei 14 coefficiente presenti in memoria sono effettivamente usati e quindi è necessario chiederli.

In base alla fase corrente dell'elaborazione cambia il suo prossimo stato. In particolore

- durante la fase di inizializzazione, in cui `K1`, `K2`, `S` e `C` devono essere letti, lo stato successivo sarà `leggiDato` per l'acquisizione del prossimo parametro
- dopo aver caricato il `nums` con i primi 3 valori da elaborare finisce la fase di inizializzazione e prima di leggere il quarto numero e iniziare la conversione si porta in `shiftState`
- durante da fase di calcolo dell'output lo stato seguente è `prodottoState`

2.5.5 shiftStato

In questo stato il dispositivo si occupa dello shift dei numeri in input sul relativo registro `nums` di una posizione verso sinistra.

2.5.6 prodottoState

In questo stato inizia la vera e propria elaborazione del numero che poi sarà scritto in memoria come output. In `moltiplicati` vengono salvati i 7 risultati delle moltiplicazioni fra `C` e `nums`. In particolare

$$moltiplicati(i) = C(i) \cdot nums(i) \qquad \forall i \in [0, 6]$$

Per semplicità della gestione del codice sorgente gli indici di `C` sono nel range [0, 6] e non [-3, 3]. per passare da una notazione all'altra è sufficiente sottrare 3

2.5.7 sommaState

In questo stato i 7 valori presenti in `moltiplicati` vengono sommati e il risultato salvato in `sommato`

$$sommato = \sum_{i=0}^{6} sommati(i)$$

2.5.8 divisioneState

In questo stato in base al tipo di filtro vengono calcoli i valori intermedi utili per la normalizzazione dell'output e successivamente sommati in `sommaDivisioni`. Come da specifica:

- per il filtro di ordine 3

$$sommaDivisioni = \lfloor \frac{sommato}{16} \rfloor + \lfloor \frac{sommato}{64} \rfloor + \lfloor \frac{sommato}{256} \rfloor + \lfloor \frac{sommato}{1024} \rfloor$$

- per il filtro di ordine 5

$$sommaDivisioni = \lfloor \frac{sommato}{64} \rfloor + \lfloor \frac{sommato}{1024} \rfloor$$

2.5.9 normState

Qualora `sommato` sia negativo, eseguendo le divisione descritte precedentemente si commette un errore pari a -1 per ciascuna divisione effettuata. In questo stato si tiene conto di ciò per avere un risultato tanto più prossimo al valore reale.

2.5.10 scrittura

E' lo stato in cui l'output della fase di elaborazione viene comunicato alla memoria per poter essere salvato. Per fare questo viene posto in `o_mem_data` e attivata la memoria in modalità scrittura (alzando i segnali di `o_mem_en` e `o_mem_we`).
Come da specifica, qualora il numero convertito fosse fuori dal range consentito $[-128, 127]$, il dato fornito alla memoria è prima ristretto al dominio.
Lo stato seguente dipende dalla fase di elaborazione, se questa fosse finita (e il dipositivo avesse analizzato tutti i numeri) si andrebbe in `fineStato`, se cosi non fosse si procederebbe con l'acquisizione di un nuovo input portandosi in `shiftState` e decrementando `remaining`

2.5.11 fineStato

L'elaborazione è avvenuta con succeso e si comunica la riuscita alzando il segnale `o_done`.

2.5.12 attesaFine

E' necessario attendere un ciclo di clock fra la fine dell'avvenuta elaborazione e l'inizio della successiva. In questo stato *o_done* è tenuto alto in modo che il chiamante possa abbassare `i_start`

3 Analisi TestBench

Il modulo creato è stato provato su diversi testbench, ognuno dei quali verifica un particolare aspetto e ne assicura la correttezza come da specifica, tenendo in considerazione i casi limite. In seguito è stato eseguito un testbench finale che racchiude in un'unica esecuzione tutti gli scenari possibili.
Per ogni testbench vengono descritti i casi analizzati, la corretta riuscita e il tempo di elaborazione richiesto.

3.1 I Testbench

Questo è il testbench fornito insieme alla specifica. Si tratta di un filtro di ordine 3, con `K` = 24. Garantisce che i parametri K_1 K_2 S e i primi 7 valori di `C` vengano letti correttamente dalla memoria, e che il modulo sia capace di produrre l'output, seppur con `K` limitato.

Il testbench viene superato, stampando il seguente messaggio
```
Failure:  Simulation Ended!  TEST PASSATO (EXAMPLE)
Time:  6270 ns
```

3.2 II Testbench

Si tratta di un filtro di ordine 3, con `K` = 65535, il massimo consentito da specifica. Garantisce che i parametri K_1 K_2 S e i primi 7 valori di `C` vengano letti correttamente dalla memoria, e che il modulo sia capace di produrre l'output per qualsiasi valore di K

Il testbench viene superato, stampando il seguente messaggio
```
Failure:  Simulation Ended!  TEST PASSATO (EXAMPLE)
Time:  6553270 ns
```

3.3 III Testbench

Si tratta di un filtro di ordine 5, con `K` = 65535, il massimo consentito da specifica. Garantisce che i parametri K_1 K_2 S e gli ultimi 7 valori di `C` vengano letti correttamente dalla memoria, e che il modulo sia capace di produrre l'output per qualsiasi valore di K

Il testbench viene superato, stampando il seguente messaggio
```
Failure:  Simulation Ended!  TEST PASSATO (EXAMPLE)
Time:  6553270 ns
```

3.4 IV Testbench

Si tratta di un filtro di ordine 3, con `K` = 7. Garantisce che la fase di normalizzazione avvenga correttamente per i numeri positivi. In particolare verifica che qualsiasi numero maggiore di 127 venga reso esattamente 127, il massimo valore di output consentito da specifica e che anche in condizioni limite il numero di bit scelto per la rappresentazione dei dati sia adeguato.

Il testbench viene superato, stampando il seguente messaggio
```
Failure:  Simulation Ended!  TEST PASSATO (EXAMPLE)
Time:  2870 ns
```

3.5 V Testbench

Si tratta di un filtro di ordine 5, con `K` = 7. Garantisce che la fase di normalizzazione avvenga correttamente per i numeri negativi. In particolare verifica che qualsiasi numero minore di -128 venga reso esattamente -128, il minimo valore di output consentito da specifica, e che anche in condizioni limite il numero di bit scelto per la rappresentazione dei dati sia adeguato.

Il testbench viene superato, stampando il seguente messaggio
`Failure: Simulation Ended! TEST PASSATO (EXAMPLE)`
`Time: 2870 ns`

3.6 VI Testbench

Verifica il corretto funzionamento del dispositivo quando deve eseguire due iterazioni, la prima per un filtro di ordine 3 e la seconda per un filtro di ordine 5. Verifica che tutti i registri siano reinizializzati e che i parametri della configurazione del filtro cambino quando una nuova esecuzione viene avviata.

Il testbench viene superato, stampando il seguente messaggio
`Failure: Simulation Ended! TEST PASSATO (EXAMPLE)`
`Time: 4630 ns`

3.7 VII Testbench

Verifica il corretto funzionamento del dispositivo quando deve eseguire 6 iterazione (sia di ordine 3 che di ordine 5), ognuna della quali con `i_add` diverso. Molte volte durante la fase di esecuzione viene comunicato il comando di reset e si verifica che il modulo sia capace di gestirlo correttamente.

Il testbench viene superato, stampando il seguente messaggio
`Failure: Simulation Ended! TEST PASSATO (EXAMPLE)`
`Time: 4965430 ns`

4 Sintesi

Verificato il corretto funzionamento del dispositivo in modalità `Behavioral` si precede con la sintesi e di seguito si riportano i dati significativi

4.1 Utilization

Table 3: Utilizzo delle risorse logiche

Site Type	**Used**	**Fixed**	**Available**	**Util%**
Slice LUTs*	858	0	134600	0.64
LUT as Logic	858	0	134600	0.64
LUT as Memory	0	0	46200	0.00
Slice Registers	350	0	269200	0.13
Register as Flip Flop	350	0	269200	0.13
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Da notare che il componente sintetizzato non fa uso di registri latch

4.2 Timing

	Setup	Hold
Worst Negative Slack (WNS)	13.036 ns	—
Total Negative Slack (TNS)	0.000 ns	—
Worst Hold Slack (WHS)	—	0.159 ns
Total Hold Slack (THS)	—	0.000 ns
Number of Failing Endpoints	0	0
Total Number of Endpoints	686	686

All user specified timing constraints are met.

Dalla tabella è possibile evincere che i paramentri sono rispettati e che il Worst Nagative Slack (WNS) è positivo

5 Conclusioni

Il successo di tutti i testbench (sia in fase `Behavioral` che in fase `Post-Synthesis`) verificano il corretto funzionamento del modulo progettato. Tenendo presente il fatto che questo vada poi sintetizzato su FPGA, abbiamo fatto attenzione all'assenza di registri latch e allo stesso tempo abbiamo cercato di minimizzare Lookup Up Table (LUT) e flip-flop, senza però sacrificare leggibilità e chiarezza del codice sorgente. Stesso discorso vale per il numero e la funzionalità degli stati della FSM, dove ad ogni stato corrisponde un'azione specifica che deve essere eseguita.