Progetto Finale, reti logiche

Politecnico di Milano

Prof. Gianluca Palermo – anno 2020-2021

Francesco Maria Tranquillo

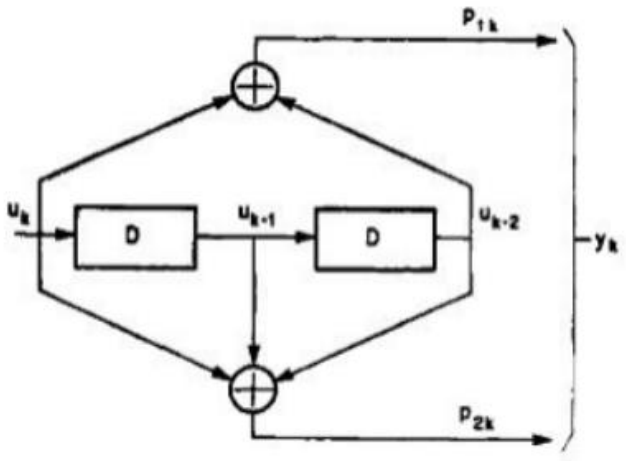
POLITECNICO
MILANO 1863

Indice

Introduzione

Obiettivo del progetto

Implementare un convolutore in grado di interfacciarsi con una memoria, con indirizzamento al byte, nella quale sono memorizzate la quantità di parole da codificare e suddette parole.
Il modulo riceve in ingresso una sequenza continua di W parole, ognuna di 8 bit, e restituisce in uscita una sequenza continua di Z parole, ognuna da 8 bit. Ognuna delle parole di ingresso viene serializzata; in questo modo viene generato un flusso continuo U da
1 bit. Su questo flusso viene applicato il codice convoluzionale ½ (ogni bit viene codificato con 2 bit) secondo lo schema riportato in figura; questa operazione genera in uscita un flusso continuo Y. Il flusso Y è ottenuto come concatenamento alternato dei due bit di uscita.
Utilizzando la notazione riportata in figura, il bit uk genera i bit p1k e p2k che sono poi concatenati per generare un flusso continuo yk (flusso da 1 bit). La sequenza d'uscita Z è la parallelizzazione, su 8 bit, del flusso continuo yk.



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

Specifiche generali

I dati salvati in memoria sono parole da 8 bit; il numero di parole da codificare è memorizzato in posizione 0 e suddette parole sono salvate a partire dalla posizione 1. Dopo aver codificato il flusso di parole il modulo dovrà salvarle in memoria a partire dalla posizione 1000.

Il modulo inizierà l'elaborazione a seguito di un innalzamento del segnale di ingresso i_start, il quale rimarrà alto fino a che il segnale di uscita i_done non verrà portato a 1, a quel punto start e done verranno portati a 0 uno dopo l'altro e il modulo rimarrà in attesa di un nuovo segnale di start. Il segnale di reset (i_rst) invece verrà usato per portare il modulo nello stato iniziale. Solitamente verrà innalzato poco prima della prima elaborazione, ma se alzato successivamente dovrà interrompere qualsiasi elaborazione e portare il modulo ad uno stato iniziale.

Diagramma dei segnali di start e done:

Interfaccia del componente

Il componente avrà 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(16 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;
```

dove:

- i_clk è il segnale di CLOCK in ingresso generato dal Test Bench;
- i_rst e il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START, o che la riporta nello stato iniziale;
- i_start è il segnale di START generato dal Test Bench;
- i_data è il segnale (vettore) che arriva dalla memoria in seguito alla 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.

Architettura

Scelte progettuali

Il progetto è costituito prima dalla definizione di alcuni componenti base che andranno utilizzati dai moduli principali per estrapolare dalla memoria, elaborare e salvare, nuovamente in memoria, i dati elaborati.
I componenti base sono flip-flop di tipo T e D oltre che alcuni latch. Nella definizione di suddetti componenti verrà usato il costrutto "process(…)", ma, eccezion fatta per questi 3 componenti, il resto del progetto verrà definito completamente tramite una rappresentazione di tipo dataflow e structural.

I moduli principali sono: count_w, multiplexer, convolutore; essi si occupano rispettivamente di: sfruttare un contatore ed una memoria inziale per ottenere l'indirizzo di memoria da cui richiedere il byte che si deve elaborare successivamente, trasformare in un flusso continuo di bit il byte richiesto e ottenuto dalla memoria, così da poter presentare il flusso al convolutore, il quale elaborerà il flusso e lo ripresenterà alla memoria.

Per poter portare il componente ad uno stato iniziale, ogni volta che viene richiesta una nuova elaborazione dei dati, si può alzare il segnale di reset, oppure non appena viene portato il segnale di start da uno stato basso 0 ad uno alto 1, il componente sfrutterà 3 flip-flop per mandare un segnale di reset a tutti i moduli prima di alzare un segnale interno che farà partire l'elaborazione.

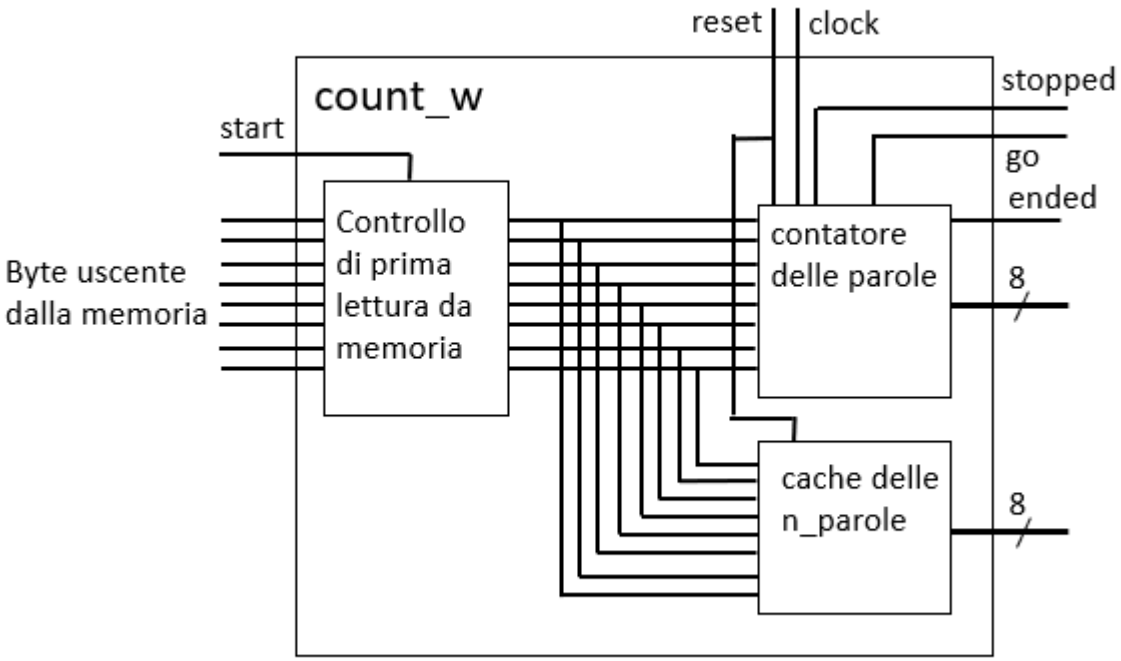
Componenti base

Come già accennato precedentemente il componente sfrutta 2 tipi di flip-flop, T e D, nei vari moduli principali oltre che a 8 latch, i quali vengono usati, solo sul fronte di discesa del clock, come memoria temporanea per salvare ogni byte generato dal convolutore prima di poterlo presentare in memoria. Nonostante sarebbe stato possibile sostituire i latch con dei flip-flop (già definiti) sono stati scelti i primi per evitare di dover attendere un ulteriore fronte del clock, dovendo così interrompere gli altri moduli; quindi grazie ai latch il convolutore presenta il dato ad un fronte di discesa del clock ed esso viene salvato (su quel fronte basso) nei latch senza dover attendere un fronte alto.

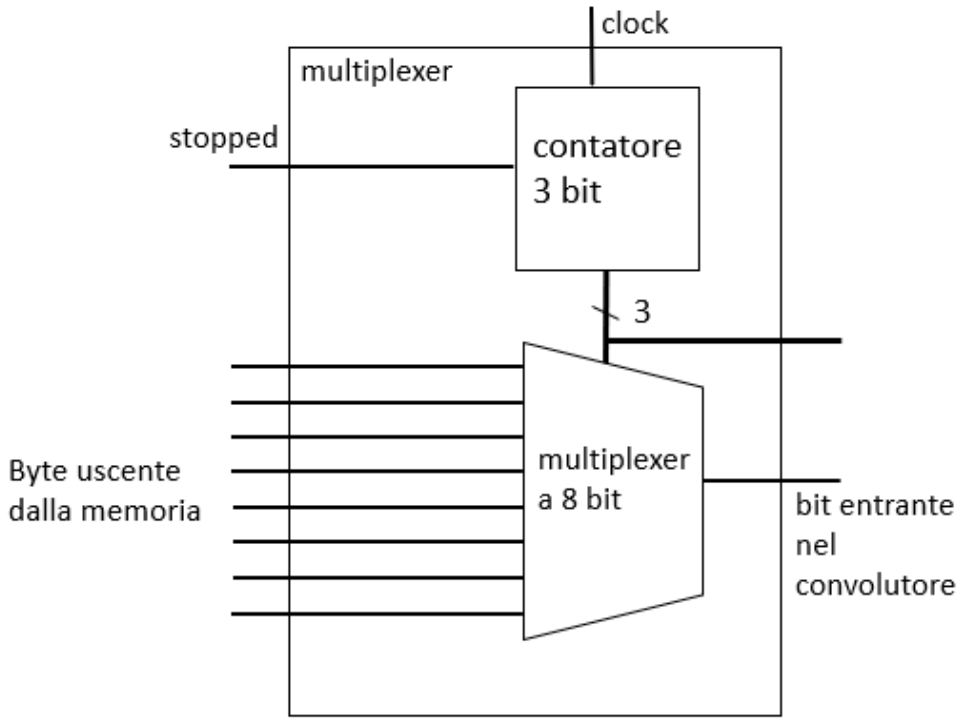
Una spiegazione più approfondita dell'utilizzo dei latch verrà presentata il seguito durante la definizione dei moduli.

Blocchi costitutivi

- ***count_w***: componente addetto a inizializzare un contatore delle parole che andranno elaborate dal convolutore.
  Esso riceve in ingresso il clock, il segnale di reset, il segnale di start, un vettore da 8 bit, che verrà usato per salvare il numero di parole da computare, e un segnale di go (approfondito successivamente); in uscita invece presenta 2 segnali (ended e stopped) e 2 vettori da 8 bit.
  Il primo vettore da 8 bit presenterà in uscita, durante tutta l'elaborazione, il numero di parole da codificare. Questo numero è salvato in una cache temporanea, composta da flip-flop di tipo D, che viene sovrascritta solo all'inizio di ogni computazione tramite l'ausilio di segnali interni (gestiti da altri 2 flip-flop) che indicheranno la prima lettura da memoria, quindi quella per cui l'indirizzo è 0.
  Il secondo vettore da 8 bit, invece, è collegato ai pin di uscita di un contatore composto da flip-flop di tipo T che, partendo dal valore salvato nella cache temporanea, decrementerà di 1 ad ogni ciclo, tranne quando il segnale stopped verrà alzato.
  Il segnale stopped verrà portato a 1 ogni volta che il contatore decrementa di 1 e verrà riportato a 0 quando un blocco esterno alzerà il segnale di go.
  Grazie ai 2 vettori presentati in uscita il componente principale sarà in grado di calcolare facilmente, facendo la sottrazione fra i 2, l'indirizzo del byte che si deve richiedere alla memoria.



- ***Multiplexer***: componente che, ricevendo in ingresso il byte letto dalla memoria, si occuperà di serializzarlo per poterlo presentare al convolutore.
  Questo componente al suo interno contiene un ulteriore contatore composto da flip-flop di tipo T che partendo da 8 decrementa di 1 ad ogni ciclo di clock. Il vettore a 3 bit in uscita dal contatore verrà usato come codice del multiplexer per selezionare uno fra gli 8 ingressi che rappresentano il byte contenuto nella memoria; ad ogni ciclo di clock quindi il multiplexer presenterà in uscita, un bit per volta partendo dal più significativo, tutti i bit del byte contenuto all'indirizzo di memoria da elaborare correntemente.

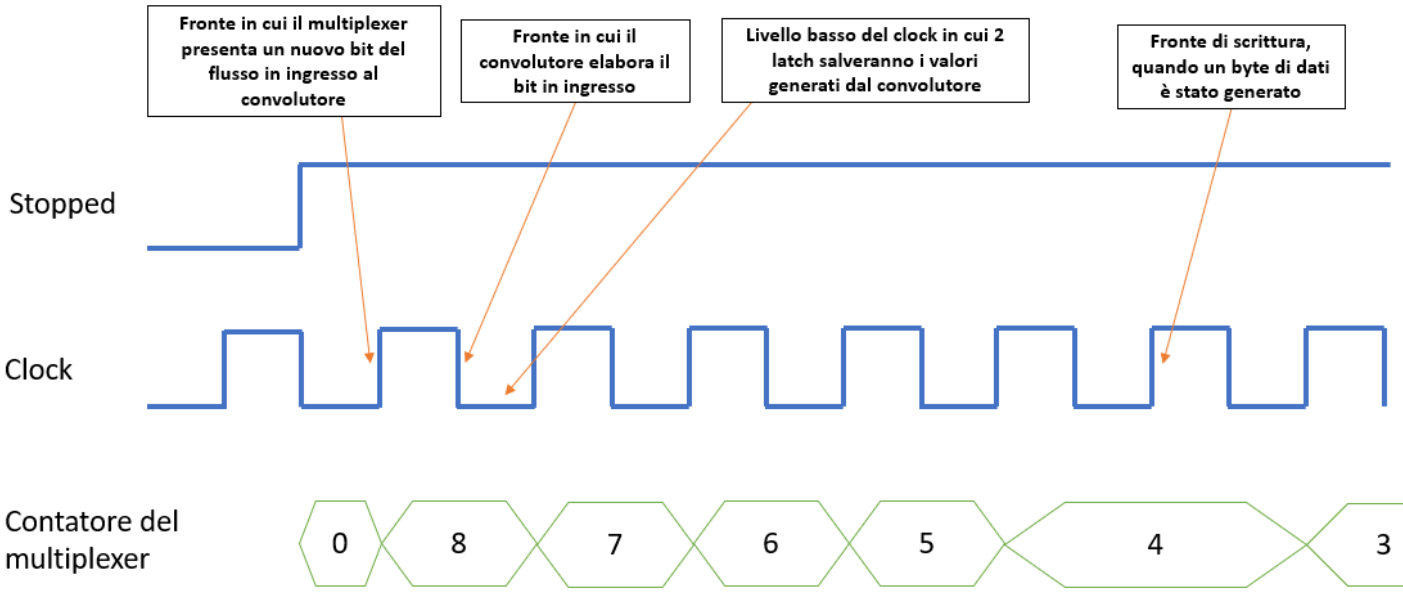


- ***Convolutore***: rappresenta il componente addetto a elaborare il flusso di bit ricevuti in ingresso. Tramite 3 flip-flop D che si occupano: uno di salvare l'ingresso (per evitare che sia trasparente) e gli altri 2 di svolgere l'effettivo comportamento del convolutore.
- ***Calcolatore***: in questo componente vengono collegati il convolutore e il multiplexer, inoltre viene generato il segnale di go e il byte da presentare alla memoria per poter essere salvato.
  Oltre ad occuparsi di unire i blocchi multiplexer e convolutore, questo componente si occupa di bloccarne il clock in ingresso quando necessario tramite l'ausilio di alcuni segnali impostati da vari flip-flop. In particolare il convolutore dovrà fermarsi quando il segnale di go diventerà alto (portando così quello di stopped di count_w a 0) in attesa del flusso di bit generato dal successivo byte, contenuto in memoria, che

andrà elaborato. Inoltre dovrà interrompersi anche nel caso in cui un byte fosse già stato generato e quindi attendere fino a che non venga salvato in memoria.
Il segnale di go, invece, verrà posto a 1 solo quando il convolutore avrà finito di elaborare tutto un byte e dopo che i 2 byte generati sono stati salvati in memoria, all'opportuno indirizzo.

Per salvare il byte generato dal convolutore vengono usati, come accennato precedentemente, dei latch, i quali vedranno l'enable posto a 1 solo sul livello basso del clock e solo quando il contatore contenuto nel multiplexer indicherà uno dei 2 valori di interesse per quel latch; ad esempio i primi 2 latch, nei quali verranno salvati i 2 bit meno significativo, saranno trasparenti solo quando il clock è basso e il contatore del componente "multiplexer" è 100 oppure 000.
Grazie all'utilizzo dei latch non appena il convolutore avrà esposto il dato in uscita (sul fronte di discesa del clock) esso verrà salvato e reso disponibile prima del fronte di salita immediatamente successivo, quando il bit del flusso in uscita dal multiplexer cambierà; evitando così di dover aspettare un altro fronte del clock e generare ulteriori segnali che dovranno interrompere i vari componenti.

Diagramma di flusso del calcolatore:



- ***project_reti_logiche***: il componente principale, composto dal calcolatore e dal count_w, oltre che da altri flip-flop usati per gestire vari segnali.
  Il componente principale si occupa, oltre che a unire i 2 blocchi sopracitati, di generare 3 segnali importanti: il primo è quello di reset, dato dal reset esplicito del

componente tramite i_rst o dall'innalzamento dell'ingresso i_start, che riporterà tutti i moduli allo stato iniziale prima dell'inizio della computazione.
Il secondo è il segnale di start, alzato poco dopo che l'ingresso i_start viene portato a 1, per permettere al segnale di reset di inizializzare il componente.
Il terzo è il segnale di done, alzato non appena il contatore presente in count_w raggiunge lo 0.
Inoltre, come introdotto precedentemente, questo modulo si occuperà di richiedere alla memoria il byte da elaborare tramite un indirizzo che potrà assumere, grazie ad un ulteriore multiplexer, due valori distinti:

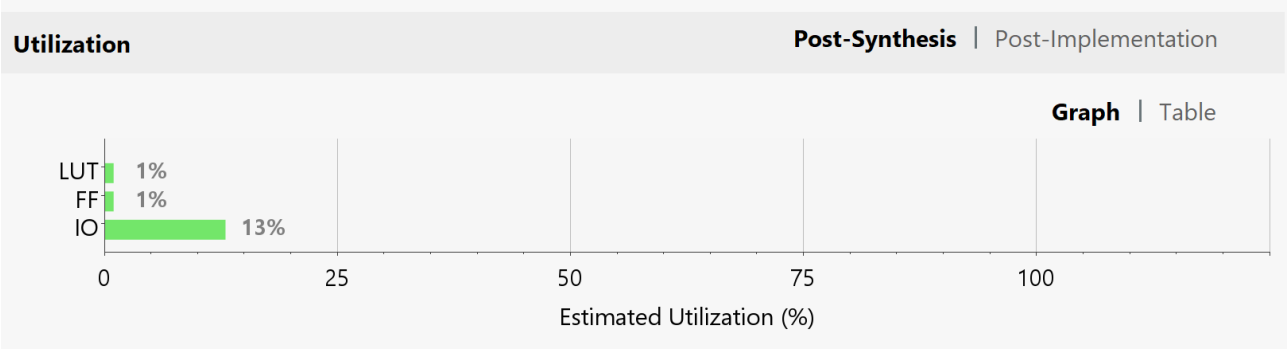
- la differenza fra la cache temporanea e il contatore del blocco count_w, che daranno l'indirizzo di memoria del byte che deve essere elaborato successivamente
- l'indirizzo di memoria in cui andrà salvato il byte generato dal convolutore, dato dall'indirizzo di memoria del byte che si sta elaborando moltiplicato per 2 e a cui si somma 998 o 999 (dato che per ogni byte di memoria ne vengono generati 2).

Per poter scegliere fra i 2 indirizzi il multiplexer si baserà sul contatore a 3 bit del blocco "multiplexer", il quale indicava il bit correntemente dato in input al convolutore. Quindi nel caso in cui il contatore assuma il valore 4 o 0 (a meno che non sia stato appena inizializzato) il multiplexer sceglierà l'indirizzo di scrittura corretto (quindi quello in cui viene sommato 998 se il contatore è arrivato a 4 e quello in cui viene sommato 999 se il contatore è arrivato a 0). In tutti gli altri casi verrà selezionato l'indirizzo dato dalla sottrazione dei 2 vettori in uscita dal blocco count_w.

Risultati sperimentali

Report di sintesi

Il componente progettato è sintetizzabile e su una FPGA di tipo xc7a200tfbg484-1 ha il seguente utilization report:



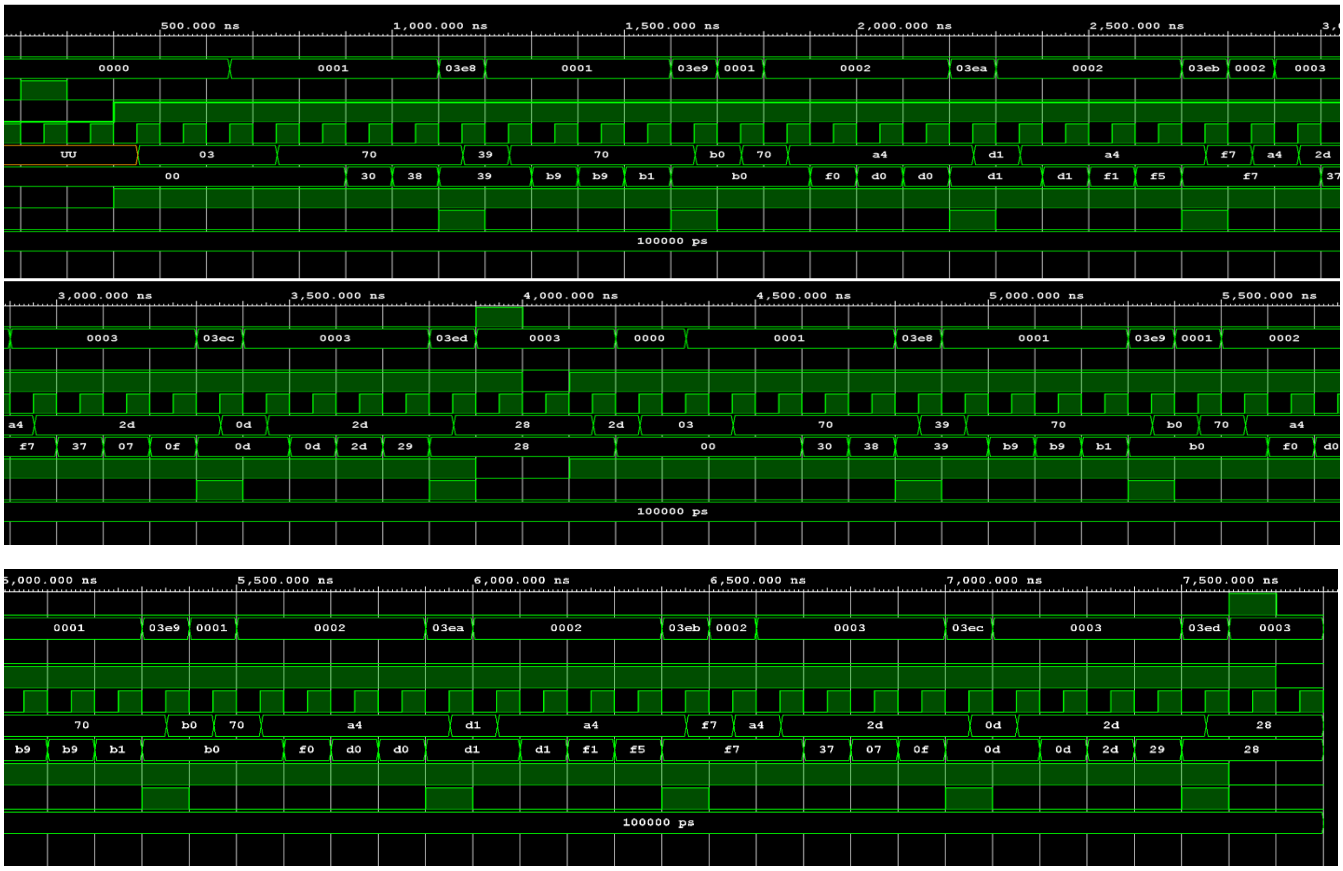
Inoltre utilizza 78 Look Up Table e 60 Flip Flop.

Testing

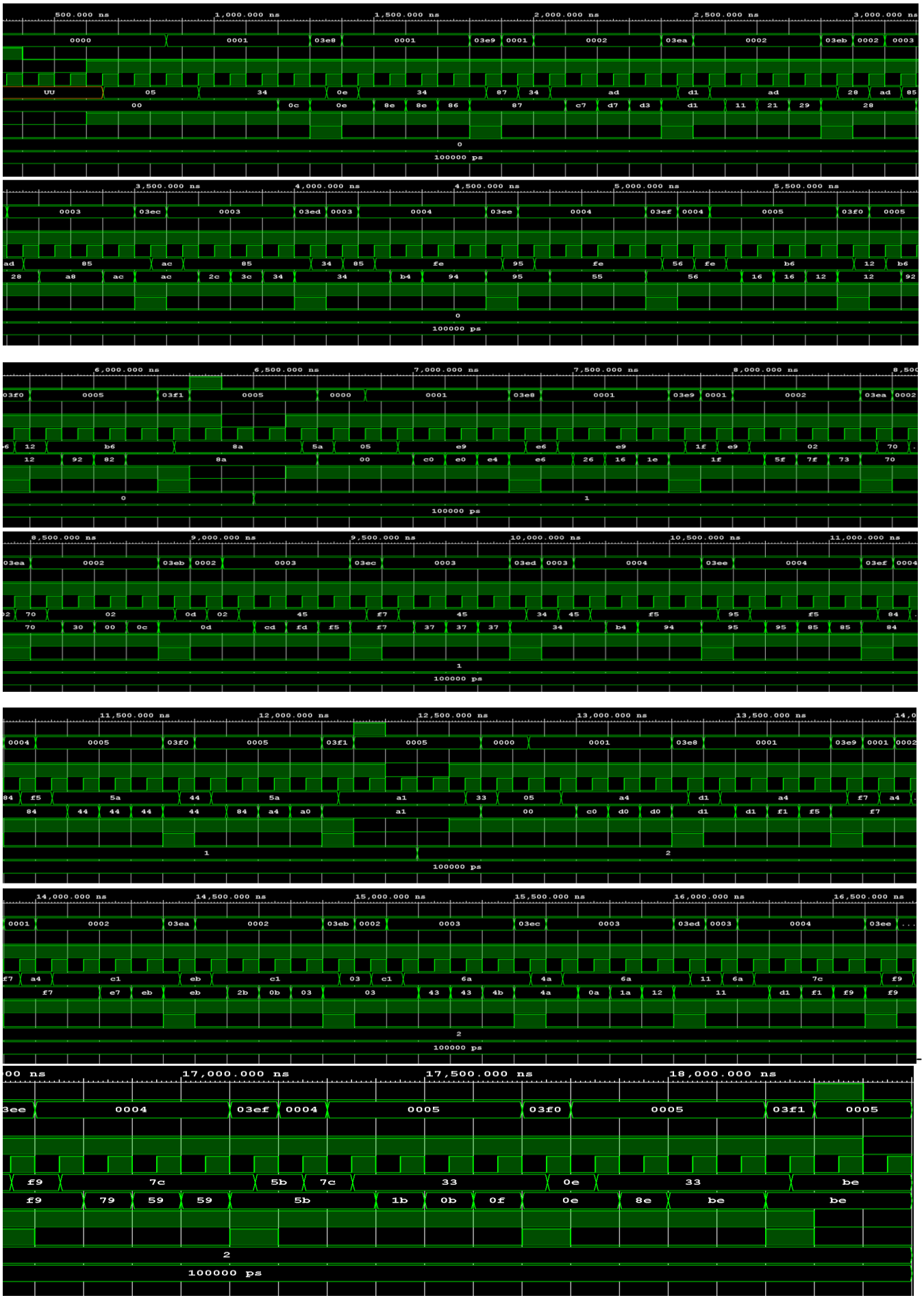
Di seguito veranno elencati alcuni dei test più critici a cui è stato sottoposto il componente, assieme a delle immagini che riportano l'andamento dei segnali di ingresso e uscita.

Esso ha passato con successo sia la simulazione di tipo behavioural che post-sintesi. (Nelle immagini a partire dall'alto i segnali rappresentati sono i seguenti: o_done, o_address, i_rst, i_start, i_clk, i_data, o_data, o_en, o_we, periodo del clock)

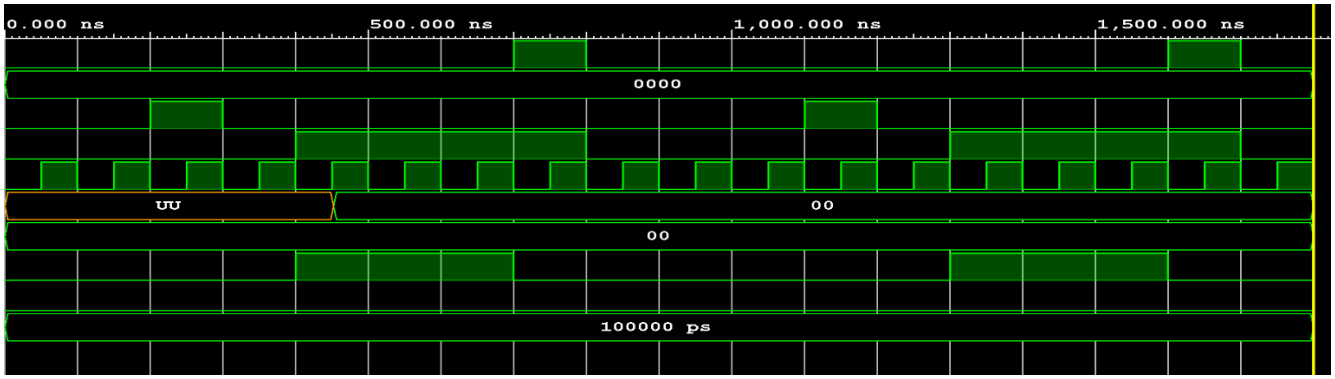
- Primo test, 2 elaborazioni dei dati sulla medesima Ram.
  Dopo aver caricato nella Ram i valori [3, 112, 164, 45] elencati partendo dall'indirizzo 0, ciò che ci si aspetta in uscita sono i valori [57, 176, 209, 247, 13, 40] elencati a partire dall'indirizzo 1000



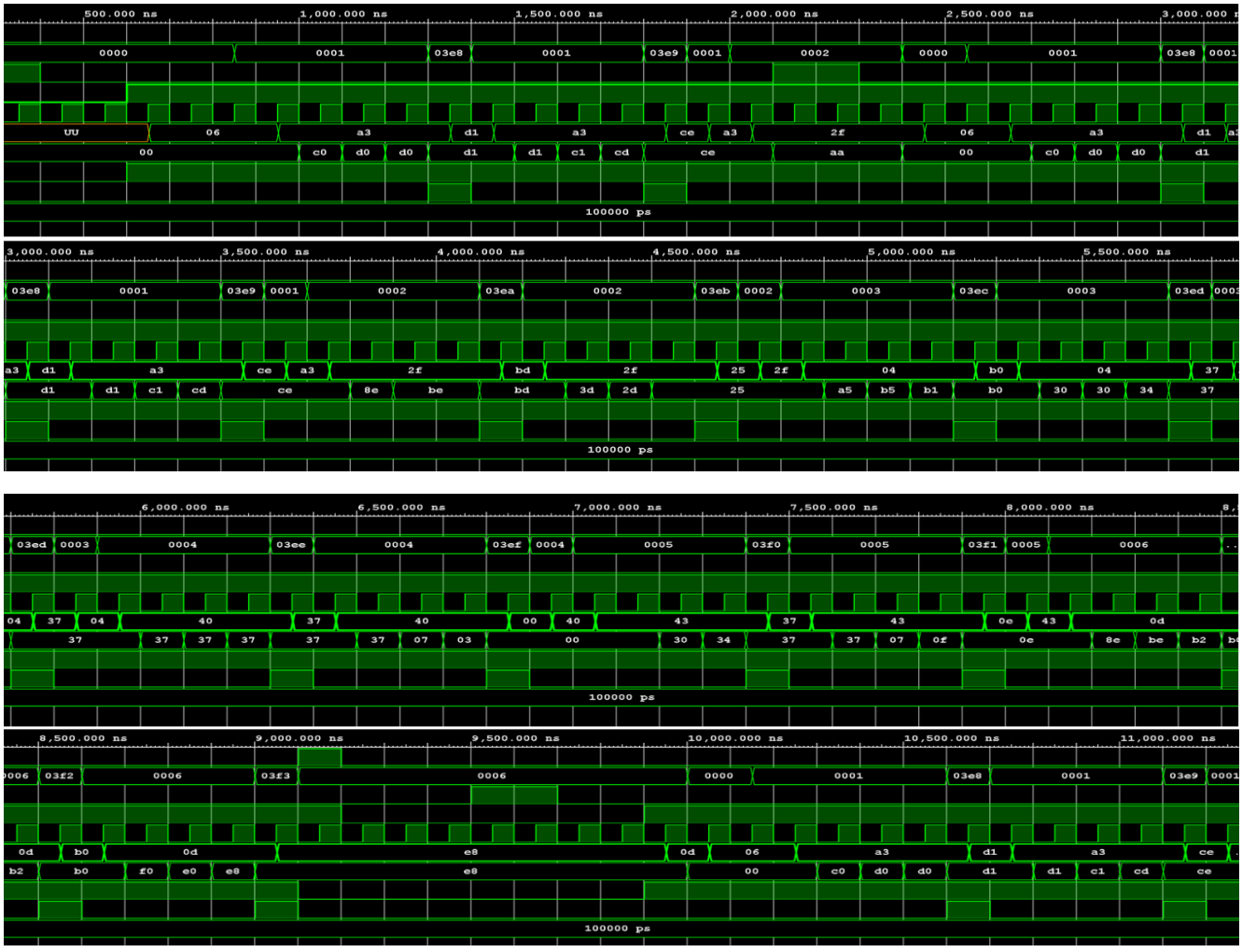
- Secondo test, 3 computazioni una di seguito all'altra.
  In questo test vengono eseguite 3 elaborazioni sui dati presenti nella Ram. Ad ogni elaborazione i valori contenuti nella Ram cambiano
  Prima elaborazione: valori di input [5, 52, 173, 133, 254, 182]
  valori di output [14, 135, 209, 40, 172, 52, 149, 86, 18, 138]
  Seconda elaborazione: valori di input [5, 233, 2, 69, 245, 90]
  valori di output [230, 31, 112, 13, 247, 52, 149, 132, 68, 161]
  Terza elaborazione: valori di input [5, 164, 193, 106, 124, 51]
  valori di output [209, 247, 235, 3, 74, 17, 249, 91, 14, 190]

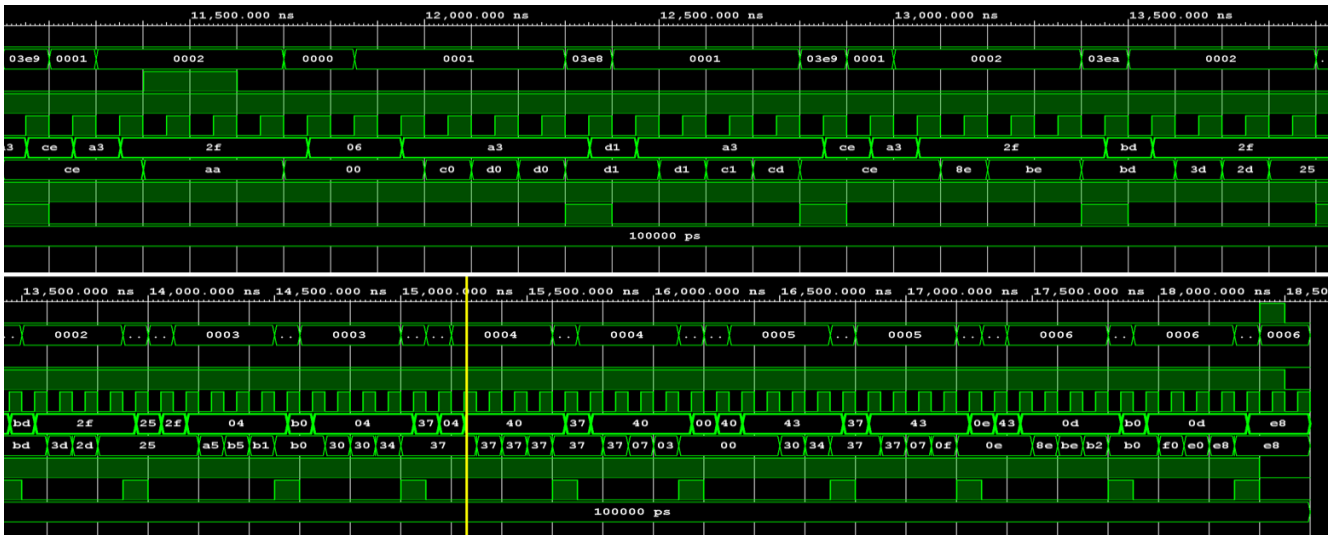

500.000 ns
1,000.000 ns
1,500.000 ns
2,000.000 ns
2,500.000 ns
3,000.000 ns
100000 ps
3,500.000 ns
4,000.000 ns
4,500.000 ns
5,000.000 ns
5,500.000 ns
100000 ps
6,000.000 ns
6,500.000 ns
7,000.000 ns
7,500.000 ns
8,000.000 ns
100000 ps
8,500.000 ns
9,000.000 ns
9,500.000 ns
10,000.000 ns
10,500.000 ns
11,000.000 ns
100000 ps
11,500.000 ns
12,000.000 ns
12,500.000 ns
13,000.000 ns
13,500.000 ns
100000 ps
14,000.000 ns
14,500.000 ns
15,000.000 ns
15,500.000 ns
16,000.000 ns
16,500.000 ns
100000 ps
17,000.000 ns
17,500.000 ns
18,000.000 ns
100000 ps

- Terzo test, 0 parole da elaborare e 255 (ovvero il massimo) parole da elaborare.
  (Per problemi di dimensione del secondo fra questi due test non è stato possibile caricarne l'immagine)



- Quarto test, reset asincroni alzati durante la computazione dei dati.
  In questi test il segnale di reset viene portato a 1 durante l'elaborazione al fine di verificare che la macchina torni in una configurazione iniziale ricominciando la computazione dei dati.





Conclusioni

Il componente è stato sottoposto, rispettando sempre le specifiche, sia ai casi di test sopra elencati, sia ad ulteriori simulazioni più lunghe e con valori di input casuali, ma anche solo variando gli input e gli output attesi per i medesimi test elencati prima. Si è cercato, inoltre, di sfruttare il più possibile sia i fronti di salita che di discesa del clock per ottimizzare il più possibile (nei limiti di come è stata concepita la sua struttura) il tempo impiegato per elaborare i dati.
Dopo aver finito gran parte della sintetizzazione del componente mi sono reso contro delle varie ulteriori alternative tramite cui lo si poteva definire (usando process e macchine a stati finiti). Tuttavia una descrizione puramente dataflow e strutturale ha dato permesso di manipolare con più libertà il componente che si stava realizzando, nonostante la gestione dei segnali scambiati e dei clock dei vari componenti probabilmente si è complicata a causa di questa scelta.