# Prova Finale (Progetto di Reti Logiche)

Alice Anselmi (10702686)

A.A. 2021/22

Docente: Prof. William Fornaciari

# Indice

**1 Introduzione: specifiche del progetto** **3**
1.1 Descrizione generale . . . . . 3
1.2 Specifica dettagliata . . . . . 3

**2 Architettura** **4**
2.1 Porte e segnali . . . . . 4
2.2 Stati . . . . . 5

**3 Risultati sperimentali** **7**
3.1 Report di sintesi . . . . . 7
3.2 Simulazioni . . . . . 7
3.2.1 Testbench 1 . . . . . 7
3.2.2 Testbench 2 . . . . . 8
3.2.3 Testbench 3 . . . . . 8
3.2.4 Testbench 4 . . . . . 8

**4 Conlcusioni** **9**

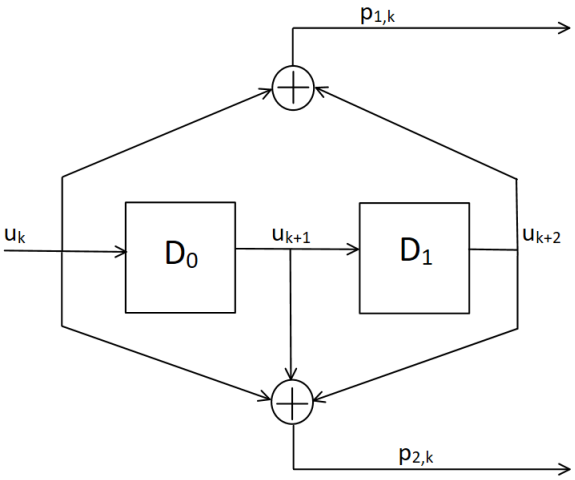
# 1 Introduzione: specifiche del progetto

## 1.1 Descrizione generale

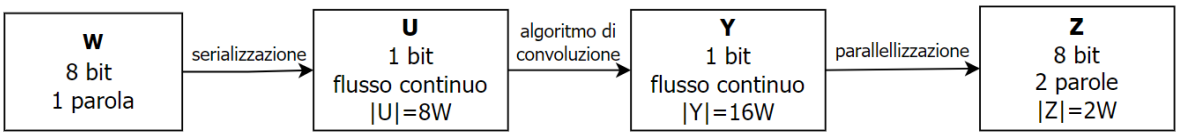
Obiettivo del progetto è implementare in VHDL (linguaggio di descrizione dell'hardware) un circuito che riceve un flusso continuo di parole dalla RAM e, tramite l'applicazione di un algoritmo di convoluzione con tasso di trasmissione $\frac{1}{2}$, genera una sequenza di byte di lunghezza doppia. Tale flusso non può superare la quantità di 255 byte, e il circuito deve essere in grado di svolgere ogni operazione necessaria con un periodo di clock di almeno 100 ns. Il software utilizzato per la stesura, implementazione, simulazione e sintesi del circuito è *Xilinx Vivado Webpack*, con dispositivo FPGA configurato Artix-7 xc7a200tfbg484-1.

## 1.2 Specifica dettagliata

Il contenuto della RAM è una sequenza di parole (al massimo 255) di 8 bit, a partire dall'indirizzo 0 della memoria; il componente accede alla memoria ed il flusso viene elaborato e trasformato secondo il seguente procedimento.
Ciascuna parola W viene serializzata per ottenere un flusso continuo di bit U che costituiscono l'input dell'algoritmo di convoluzione, di cui sono qui sotto illustrati lo schema circuitale e la macchina a stati finiti. Tramite l'algoritmo, che prevede l'utilizzo di due flip-flop D e due porte XOR, si ottengono due bit $p1_k$ e $p2_k$ da ogni bit del flusso $u_k$. A ogni ciclo di clock si concatenano $p1_k$, $p2_k$, $p1_{k+1}$, $p2_{k+1}$ e così via, arrivando ad ottenere un flusso di 16 bit Y. L'ultimo passaggio prevede quindi la formazione di 2 byte Z, corrispondenti alla parallelizzazione dei primi 8 Y e degli 8 seguenti.
Il processo descritto viene ripetuto per ogni parola del flusso in input, quindi, una volta terminata l'acquisizione, l'elaborazione e la scrittura dell'output per ogni parola, il componente si resetta riportando al valore di default tutti i parametri ed è pronto per una nuova elaborazione.
Di seguito lo schema dell'algoritmo di convoluzione:




Il processo completo può invece essere rappresentato nel seguente modo:



# 2 Architettura

Il componente progettato è una macchina a stati finiti regolata da un ciclo di clock e dotata di stato di reset. Il numero totale di stati è 11; di seguito una descrizione dettagliata di porte e stati del componente.

## 2.1 Porte e segnali

Porte di interfaccia:

- *i_clk*: segnale di clock che scandisce le transizioni della macchina a stati.
- *i_rst*: segnale di reset; se portato a 1 in qualsiasi fase del processo di elaborazione causa il reset dell'intera macchina.
- *i_start*: segnale di start che dà inizio all'acquisizione di dati dalla RAM.
- *i_data*: vettore di 8 bit che contiene il byte (proveniente dalla RAM) da elaborare.
- *o_address*: vettore di 16 bit che regola l'indirizzo di accesso alla memoria, sia in lettura che in scrittura.
- *o_done*: segnale di termine da alzare quando si è completata l'elaborazione di tutte le parole del flusso.
- *o_en*:segnale che, se portrato a 1, permette di accedere alla memoria sia in lettura che in scrittura.
- *o_we*: segnale che permette la scrittura di dati sulla RAM.
- *o_data*: vettore di 8 bit contenente il byte da scrivere in memoria.

Segnali:

- *is_first_round*: booleano che assume valore "true" se il componente è al primo avvio dopo che il segnale *i_start* è stato alzato, "false" altrimenti.
- *i_addr*: vettore di 16 bit contenente il valore da sommare a *o_address* per ottenere l'indirizzo di accesso alla RAM in lettura.
- *word_count*: vettore di 8 bit in cui viene memorizzato il numero di parole da elaborare, viene decrementato alla fine dell'elaborazione di ogni parola W.
- *w*: vettore di 8 bit contenente la parola W in ingresso dalla memoria.
- *pos_w*: intero che indica la posizione, nel vettore w, del bit $u_k$ da inserire nel processo di convoluzione. Parte da 7 e viene decrementato fino a 0.
- $u_k$: bit ottenuto dalla serializzazione di W (elemento in posizione *pos_w* di W).
- $q_0$, $q_1$: bit che indicano i valori assunti dalle uscite dei due flip-flop D conenuti nel circuito di convoluzione.
- $p1_k$, $p2_k$: bit che contengono i due risultati dell'algoritmo di convoluzione.
- *z*: vettore di 8 bit che contiene il risultato della concatenazione e parallelizzazione dei $p1_k$ e $p2_k$.
- *pos_z*: intero utilizzato per tenere conto della posizione in cui inserire i $p_k$ all'interno del vettore z. Inizializzato a 7, alla fine della scrittura di una parola Z arriva a 0.
- *is_end*: bit che segnala la fine dell'elaborazione e scrittura di entrambi i byte Z ottenuti da una parola W.
- *o_addr*: vettore di 16 bit, è il valore da sommare a *o_address* per ottenere l'indirizzo di accesso alla RAM in scrittura.

## 2.2 Stati

- *RST*: stato di reset della macchina in cui i segnali vengono riportati al valore di default. All'inizio del processo di elaborazione la macchina si trova in questo stato, e vi rimane finchè non viene alzato il segnale *i_start*; inoltre vi ritorna ogni qual volta si abbia *i_rst*=1, in qualsiasi punto dell'elaborazione si trovi il processo. La FSM rimane in questo stato finchè non si dà inizio all'acquisizione dei dati: a quel punto si passa allo stato *READ_PREP*.

- *READ_PREP*: questo stato serve alla FSM per prepararsi a leggere il dato presente in memoria, pertanto *o_en* viene portata a 1. Vengono anche riportate al loro valore di default le variabili *pos_w* e *pos_z* in quanto la macchina ritorna a questo stato ogni volta che termina la scrittura delle due parole Z per leggere un nuovo byte. Vi è una clausola (non rappresentata nello schema della macchina) che tratta il caso particolare in cui si è ritornati a *READ_PREP* dopo la lettura del primo dato in memoria, ovvero il numero di parole da elaborare, e questo è pari a 0: in tal caso la macchina non deve più fare nulla quindi va nello stato FINISH.

- *READ_WAIT*: stato intermedio che permette alla memoria di ricevere il segnale di lettura e di "preparare" il dato da acquisire.

- *READ_WORD*: qui viene eseguita la lettura del dato contenuto nella memoria in base al valore di *is_first_round*: se è true la macchina è stata avviata per la prima volta e va quindi letto il numero di parole da elaborare per poi tornare a *READ_PREP* per leggere il prossimo dato, se è false viene acquisita e memorizzata la parola W da 8 bit e si inizia l'elaborazione.

- *BIT_SEL*: in questo stato avviene la serializzazione della parola W: si seleziona il bit in posizione *pos_w* e lo si memorizza in $u_k$. In parallelo avviene l'avanzamento del circuito di convoluzione: se $u_k$ non è UNDEFINED (valore a cui sono inizializzati $q_0$ e $q_1$), va in ingresso al flip-flop $d_0$ e dunque nel prossimo ciclo di clock si avrà $q_{0,k+1}=u_k$; allo stesso modo, se $q_0$ non è UNDEFINED entra nel flip-flop $d_1$ e dunque avremo $q_{1,k+1}=q_{0,k}$.

- *PK_CALC*: vengono calcolati $p_{1,k}$ e $p_{2,k}$ secondo lo schema dell'algoritmo di convoluzione utilizzando i valori attuali di $u_k$, $q_0$ e $q_1$. Per quanto riguarda le uscite dei flip-flop, se il processo si trova ancora nei cicli iniziali e $q_0$ e $q_1$ sono ancora UNDEFINED, vengono trattati come se "non fossero presenti" - per esempio, nel caso in cui $q_1$='U', si ha $p_{1,k}=u_k$ invece che $p_{1,k}=u_k$ xor $q_1$. In parallelo continua ancora l'esecuzione del circuito di convoluzione.

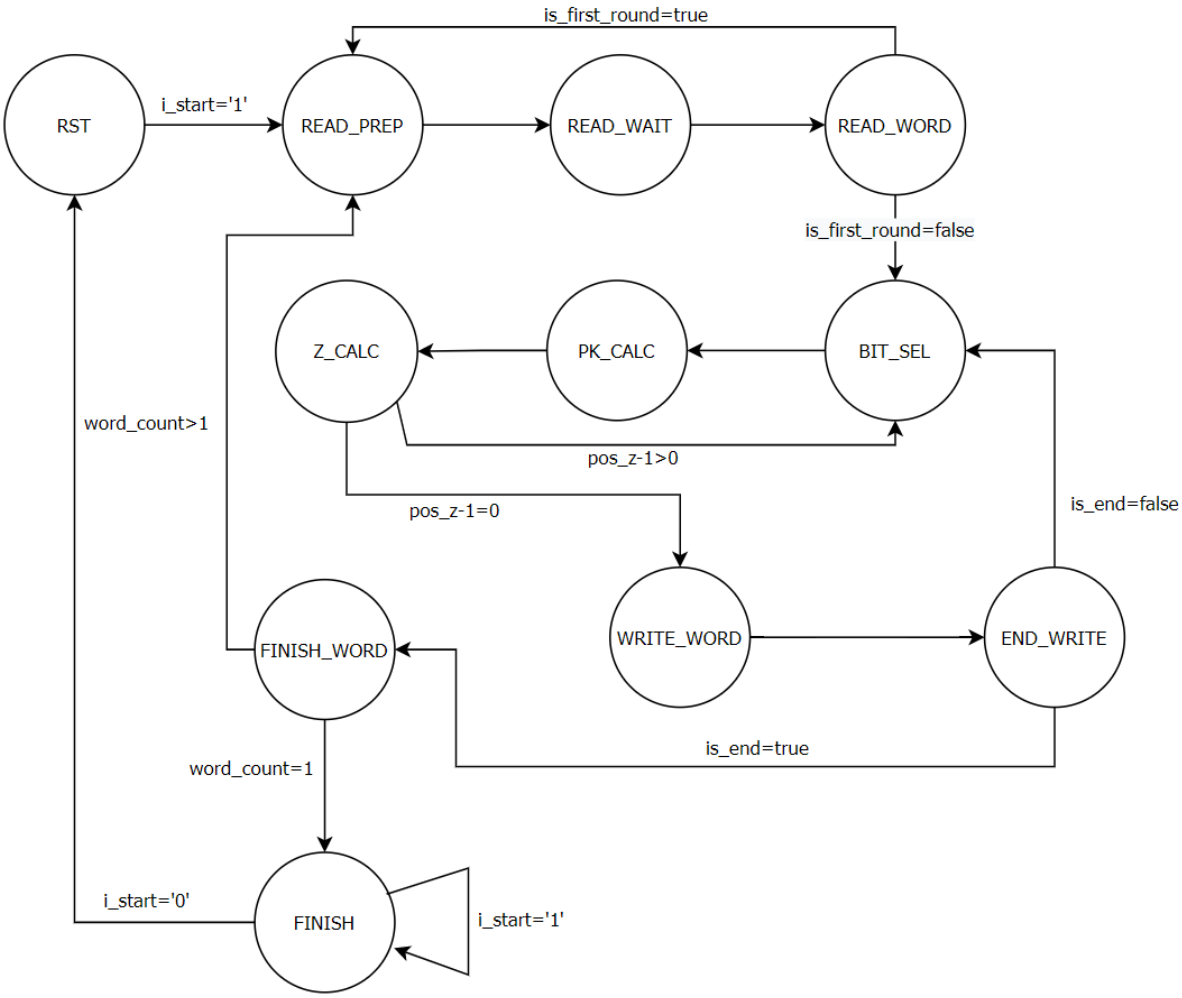
- *Z_CALC*: qua si memorizzano i risultati del codificatore convoluzionale nel vettore z, nelle posizioni *pos_z* e *pos_z*-1 (secondo la specifica, i bit vanno memorizzati nella parola a partire dal più significativo, dunque si va dalla posizione 7 alla posizione 0). Se i bit scritti erano gli ultimi della parola si passa alla fase di scrittura di z, altrimenti si prosegue con la convoluizione e con il calcolo dei $p_k$.

- *WRITE_WORD*: stato di scrittura del dato in memoria. *o_en* e *o_we* vengono alzati e il byte z viene copiato in *o_data*.

- *END_WRITE*: in questo stato si abbassa il segnale di scrittura e si incrementa *o_addr* in vista della prossima write, quindi i casi sono due: se la parola scritta nello stato precedente era il secondo vettore z (*is_end*=true), allora la computazione della W in ingresso è terminata e si va allo stato seguente; altrimenti si riporta z al valore di default e la macchina torna allo stato *BIT_SEL* per continuare l'elaborazione dei 4 bit rimanenti di W.

- *FINISH_WORD*: stato in cui si controlla se tutte le parole memorizzate nella RAM sono state elaborate; in tal caso, si passa alla fase di terminazione, altrimenti si decrementa il contatore *word_count* e si ritorna a *READ_PREP* per ricominciare il processo con una nuova W.

- *FINISH*: questo è lo stato di terminazione in cui si aspetta che *i_start* venga abbassato per resettare la macchina e cominciare un nuovo processo.


is_first_round=true
RST
i_start='1'
READ_PREP
READ_WAIT
READ_WORD
is_first_round=false
Z_CALC
PK_CALC
BIT_SEL
word_count>1
pos_z-1>0
pos_z-1=0
is_end=false
FINISH_WORD
WRITE_WORD
END_WRITE
is_end=true
word_count=1
i_start='0'
FINISH
i_start='1'

# 3 Risultati sperimentali

## 3.1 Report di sintesi

Gli strumenti di report mostrano i seguenti risultati:

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 227 | 134600 | 0.17 |
| FF | 157 | 269200 | 0.06 |
| IO | 38 | 285 | 13.33 |

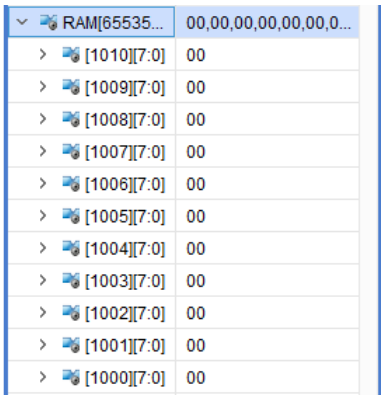
dove LUT sta per LookUpTable e FF rappresenta il numero di flip-flop utilizzati. Il componente non presenta nessun latch nella sintesi.

## 3.2 Simulazioni

### 3.2.1 Testbench 1

Il primo testbench tratta il caso limite in cui viene dato in input un word_count pari a 0 e quindi la memoria resta invariata: di seguito un frammento di codice del testbench e il contenuto della RAM che rimane invariato.

```
23
24    signal RAM: ram_type := (0 => std_logic_vector(to_unsigned(  0  , 8)),
25                             others => (others =>'0'));
26
27    component project_reti_logiche is...
40
41    begin
42    UUT: project_reti_logiche...
54
55    p_CLK_GEN : process is...
60
61    MEM : process(tb_clk)...
74
75    test : process is
76    begin
77        wait for 100 ns;
78        wait for c_CLOCK_PERIOD;
79        tb_rst <= '1';
80        wait for c_CLOCK_PERIOD;
81        tb_rst <= '0';
82        wait for c_CLOCK_PERIOD;
83        tb_start <= '1';
84        wait for c_CLOCK_PERIOD;
85        wait until tb_done = '1';
86        wait for c_CLOCK_PERIOD;
87        tb_start <= '0';
88        wait until tb_done = '0';
89        wait for 100 ns;
90
91        assert RAM(1000) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
92        assert RAM(1001) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
93        assert RAM(1002) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
94        assert RAM(1003) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
95        assert RAM(1004) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
96        assert RAM(1005) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
97        assert RAM(1006) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
98        assert RAM(1007) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
99        assert RAM(1008) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
100       assert RAM(1009) = STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)) report "TEST FALLITO" severity failure;
101
102       assert false report "Simulation Ended! TEST PASSATO" severity failure;
```




### 3.2.2 Testbench 2

Nel secondo testbench si analizza il caso in cui la sequenza di parole nella RAM sia della massima lunghezza (255 byte): il programma passa il test.

### 3.2.3 Testbench 3

Il componente passa qualsiasi in test in cui vi sia almeno un reset, riuscendo a gestire casi di reset asincrono nel mezzo dell'esecuzione del processo, come nel caso trattato dal seguente codice.

```
121 test : process is
122 begin
123     wait for 100 ns;
124     wait for c_CLOCK_PERIOD;
125     tb_rst <= '1';
126     wait for c_CLOCK_PERIOD;
127     wait for 100 ns;
128     tb_rst <= '0';
129     wait for c_CLOCK_PERIOD;
130     wait for 100 ns;
131     tb_start <= '1';
132 
133     wait for c_CLOCK_PERIOD;
134     wait for 50 ns;
135        tb_start <= '0';
136        tb_rst <= '1';
137        wait for c_CLOCK_PERIOD;
138        tb_rst <= '0';
139        tb_start <= '1';
140     wait until tb_done = '1';
141     wait for c_CLOCK_PERIOD;
142     tb_start <= '0';
143     wait until tb_done = '0';
144     wait for 100 ns;
145     i <= "01";
```

### 3.2.4 Testbench 4

Infine, viene ovviamente passato anche qualsiasi test in cui non vi siano reset, eseguendo correttamente elaborazione delle parole e scrittura dell'output in memoria.

# 4 Conlcusioni

Per concludere, il componente progettato soddisfa tutte le specifiche ed è stato ideato in modo da gestire qualsiasi casistica particolare. Se nella RAM vi sono dati già memorizzati all'indirizzo 1000 e seguenti, vengono sovrascritti solo i byte strettamente necessari alla conversione delle parole in ingresso, mentre il resto della memoria rimane invariato. Inoltre, il report mostra che lo slack è di 93.272 ns, ben oltre le richieste della specifica: ciò significa che la FSM e gli stati che la compongono sono ben progettati e ottimizzati - la parte del componente che implementa l'algoritmo di convoluzione era sicuramente quella più delicata e, allo stesso tempo, quella con più margini di miglioramento nelle tempistiche.